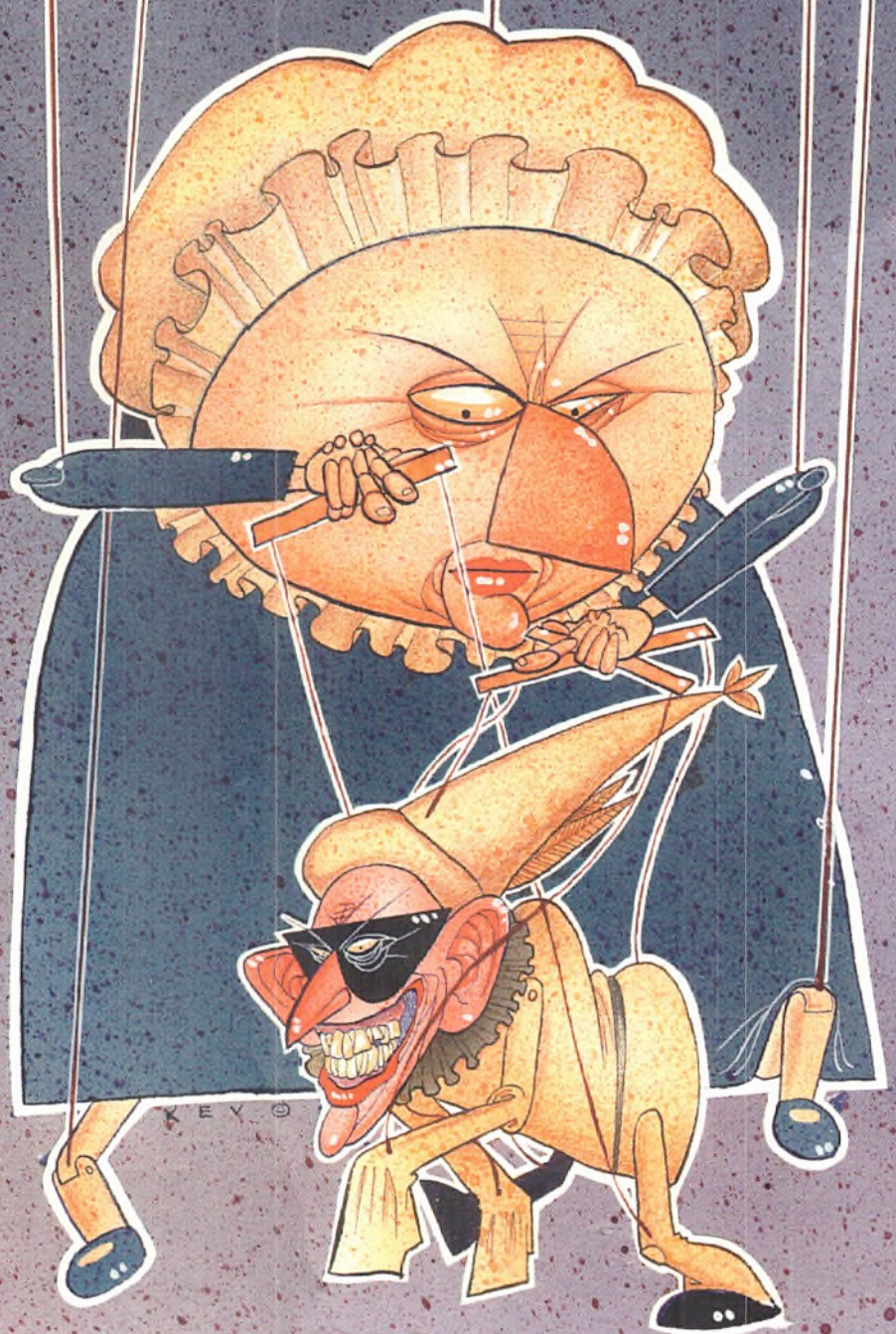
A supplement to EXE Magazine, October 1995

Visual PROGRAMMING

OLE in control



Powerful. Efficient. Reliable.

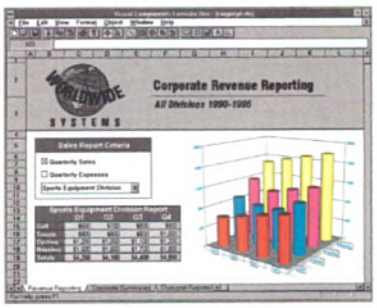
Introducing OLE Custom Controls for Windows 95 applications

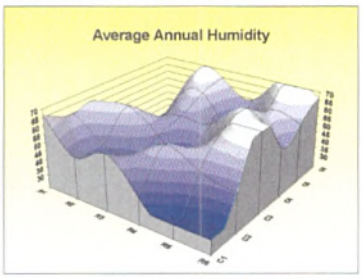
Cutting edge development tools for Windows developers

Formula One™ 3.0

£195

- The best spreadsheet component just got better!
- **NEW!** Now available as a 16-bit and 32-bit OLE Custom Control.
- **NEW!** Read and write Microsoft Excel 5.0 worksheets.
- **NEW!** Support for Excel-style workbooks including the tabbed worksheet interface; include up to 256 worksheets in a workbook! Plus, use three-dimensional cell references.
- Enhanced calculation engine. The fastest spreadsheet component just got faster!
- Larger color palette.
- **NEW!** New worksheet functions including SUMIF, COUNTIF, CONCATENATE, ROUNDUP, and ROUNDDOWN.
- **NEW!** New events including Mousedown, Mouseup, Mousemove, ObjGotFocus, ObjLostFocus, and Modified.
- Enhanced printing support.
- **NEW!** Autofill worksheets with days, dates, numbers, or create your own custom fills.
- **NEW!** Draw chart objects directly on worksheets.
- **NEW!** Data validation rules.
- Royalty-free run time distribution.





First Impression® 2.0

£195

- Now, get even more charting and graphics power!
- **NEW!** Now available as a 16-bit and 32-bit OLE Custom Control.
- **NEW!** New chart types include XYZ, Contour and 3D Surface charts.
- **NEW!** New data grid allows you to view and interactively change chart data.
- **NEW!** Chart Wizard takes you step-by-step from raw data to finished chart. Plus, it includes an easy-to-use gallery of chart types.
- **NEW!** Complete object-based API.
- The only charting component that gives you a photo-realistic rendering engine. Create 3D charts in true perspective.
- Royalty-free run time distribution.

VisualSpeller™ 2.0

£129

- **NEW!** Now available as a 16-bit and 32-bit OLE Custom Control.
- **NEW!** Improved and streamlined API.
- Includes comprehensive standard dictionary with more than 100,000 entries.
- Easily create custom dictionaries with built-in dictionary maker.
- Royalty-free run time distribution.

Visual Components propels you into the next generation of Windows-based application development with the introduction of our OLE custom controls. Available as 16-bit and 32-bit products, reusable OLE components from Visual Components are the perfect tools for building state of the art, cutting edge applications in Visual Basic, Visual C/C++, Access, Visual FoxPro, and other OCX supported environments.

Look to Visual Components for robust development tools that provide unmatched functionality and the reliability you require for mission critical applications. It won't take long for you to understand why we're absolutely the leader in component software.

VISUAL COMPONENTS™

The Leader in Component Software

CALL

+44 1892 834343

LENEXA HOUSE, 11 ELDON WAY,
PADDOCK WOOD, KENT
TN12 6BE, UK

TEL: +44 1892 834343 • **FAX:** +44 1892 835843
BBS: +44 1892 835579

Visual Basic 4.0

Version 4.0 of the ubiquitous Windows development tool Visual Basic has been long awaited. **Roland Perera** finds out if Microsoft has finally delivered the goods.

Visual Basic is back. After a protracted gestation period - nearly a year in beta testing - the latest incarnation of Microsoft's development system for Windows has arrived. The hype surrounding Windows 95 has seen many opportunists, Microsoft included, riding the crest of a marketing tsunami. Software companies have seized the chance to revamp their old Win16 applications into all-new, but strangely not-so-new, 32-bit versions. Cynical thoughts like these lead me to treat the arrival of Visual Basic 4.0 with some suspicion.

So what's on offer? Visual Basic 4.0 is available in Standard and Professional Editions, like its predecessor, and a new 'Enterprise' Edition, bundled with software and features suitable for corporate client/server development. See Table 1 for a summary of the main components of the Standard, Professional and Enterprise editions.

All three editions provide 16- and 32-bit versions of the development system. I should point out that Visual Basic is not a cross-platform tool - it's not possible to target Win16 from the 32-bit IDE, for example.

I'm going to start by looking at the core of Visual Basic 4.0 - what's supplied with the Standard Edition. Later on I'll take a cursory peek at some of the bits and bobs that come with the other editions.

Going 32-bit

The significance of VB 4.0's arrival for most people will be the opportunity finally to develop Visual Basic client and stand-alone programs that run natively under a 32-bit operating system. Until now many companies developing client/server applications have been forced either to run Windows 3.x on the client or to choose a different programming tool. Unfortunately, porting an existing 16-bit VB application to 32-bit is not necessarily straightforward.

Under the 32-bit environment, Visual Basic eXtensions (VBXs) are not supported. Instead, Visual Basic 4.0 automatically replaces any VBX controls it recognises with corresponding 32-bit OLE Control eXtensions (OCXs) when a project is loaded. The OCX control must be 100% compatible with the VBX it is to replace. Entries in VB's configuration file (VB.INI) map VBXs to OCXs and identify the control's type library. The OLE controls included in VB 4.0 are functionally identical to the VBXs that shipped with Version 3.0, but unlike VBXs, exist in both 16-bit and 32-bit forms. If your VBX doesn't have a corresponding OCX, you'll have to wait until third parties catch up.

With the 16-bit IDE, VBXs *are* supported, although their use is deprecated - none are actually supplied with the product. When a VB 3.0 project is first loaded into VB 4.0, the default behaviour is to replace any recognised VBXs by 16-bit OCXs. (Usefully, recognised 32-bit OCXs are also replaced by their 16-bit counterparts, allowing the same project to be used to target both platforms.) Table 2 lists the new standard OLE controls that come with Visual Basic.

Standard Edition	Professional Edition	Enterprise Edition
Visual Basic IDE	← All this, plus...	← All this, plus...
Visual Basic for Application engine	Additional 3D controls	Remote Data control and Remote Data Objects
Enhanced Jet engines	ODBC 2.5	Remote OLE Automation
Data Access Objects	OLE Automation servers	Visual SourceSafe 4.0
Data Manager utility	Office Developers Kit	Component Manager
Setup Wizard	Crystal Reports 3.0	
	Help compiler	
£69 (upgrade £39)	£329 (upgrade £119)	£649 (upgrade £369)

Table 1 - What do you want to buy today?™

Data access

The Jet 2.5 and 3.0 database engines (16-bit and 32-bit respectively) are faster, and have seen other much-needed improvements, including a new Data Access Objects (DAO) programming model, cascading updates and deletes, and programmatic access to referential integrity. Jet 2.5 and 3.0 are functionally compatible, so you can build both 16-bit and 32-bit database applications from the same code.

It's now also possible to have your data-bound controls linked to *native* drivers for various databases, including Oracle and SQL Server. No longer do you have to incur the additional overhead of ODBC. Interestingly, your VB application is still portable across database platforms, since the control itself forms an abstraction layer (similar in concept to ODBC). This layer has its own overhead, but as OCXs are implemented as *in-process* servers (DLLs) this should be quite low.

The Gooey

Interface-wise, Version 4.0 is hardly a quantum leap of improvement over earlier releases. Property sheets would probably be

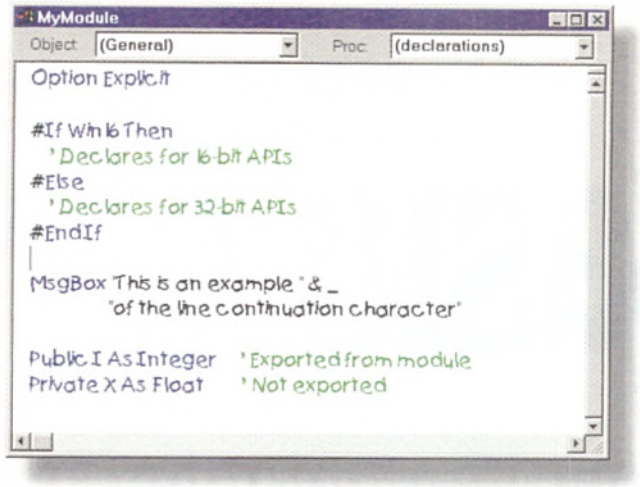


Figure 1 - New language features (and a silly font)

Visual Basic 4.0 is not Object Basic... yet

more intuitive than a list box for selecting the object whose code is to be edited. The syntax colouring feature would be more handy if keywords were coloured as soon as they were typed: instant feedback is one of the reasons syntax colouring is useful at all. Currently colouring only occurs when you move the cursor off the current line. However, the colours used are customisable, as is the font used by the editor (as you'll see from the screen captures).

Common activities such as designing forms, editing properties, selecting controls and watching expressions are much the same as they were in VB 3.0, with the addition of right-button context menus for common tasks like editing a watch or stepping to the cursor. Whilst on the subject, it would be handy to have context-sensitive help available from the right mouse button as well as from the *F1* key. I found myself looking for this when using both the Control toolbox and the Properties window.

Plus ça change...

VB Project files now have the extension .VBP rather than the old, overloaded .MAK suffix. A requirement for 100% backward compatibility with VB 3.0 means that the language itself has to remain fairly static. Even so, Visual Basic for Applications, the language for programming Microsoft Office, is now the core engine for Visual Basic 4.0. See Figure for some examples of new features.

Conditional compilation is now supported to allow developers to target different platforms from the same source code. The lack of a line continuation character, an omission that occasionally detracted from program readability, has at last been remedied. (Oddly enough, Visual Basic for *DOS* has sported such a character since its inception.)

As with Visual Basic 3.0, a variable, constant or procedure declared `Private` in the Declarations section of a module can only be referred to from within the module in which it is defined. Version 4.0 provides more language support for controlling the visibility of declarations in the form of the new keyword `Public`, which forces a declaration to be available to the entire application. `Private` and `Public` declarations are an important consideration when designing class modules (see below).

A Pascal-esque `With` statement also makes an appearance, syntactic sugar to makes code that repeatedly uses the same object more legible. See Figure 2 for an example. A disclaimer in the Help file makes me question the safety of Visual Basic: *Don't jump into or out of With blocks. You may get errors or unpredictable behavior if statements in a With block are executed, but either the With or End With statement is not executed.* In other words, the `With` statement is there to be used, but there's no compiler support to make sure you use it safely!

Collections

Visual Basic 4.0 provides a couple of new constructs to make handling *groups* of objects easier. VB 3.0 supplied a variety of built-in object 'collections', such as `Forms` (a global variable holding the application's loaded forms) and `Controls` (a property of a `Form` object enumerating its controls). Now you can define your own group of objects, by creating and maintaining `Collection` objects - generic containers that can hold objects of any type, with support for LIFO, indexed and hashed access to their elements.

A new `For Each...Next` control structure has been added to the language to make iteration over elements of a collection easier.

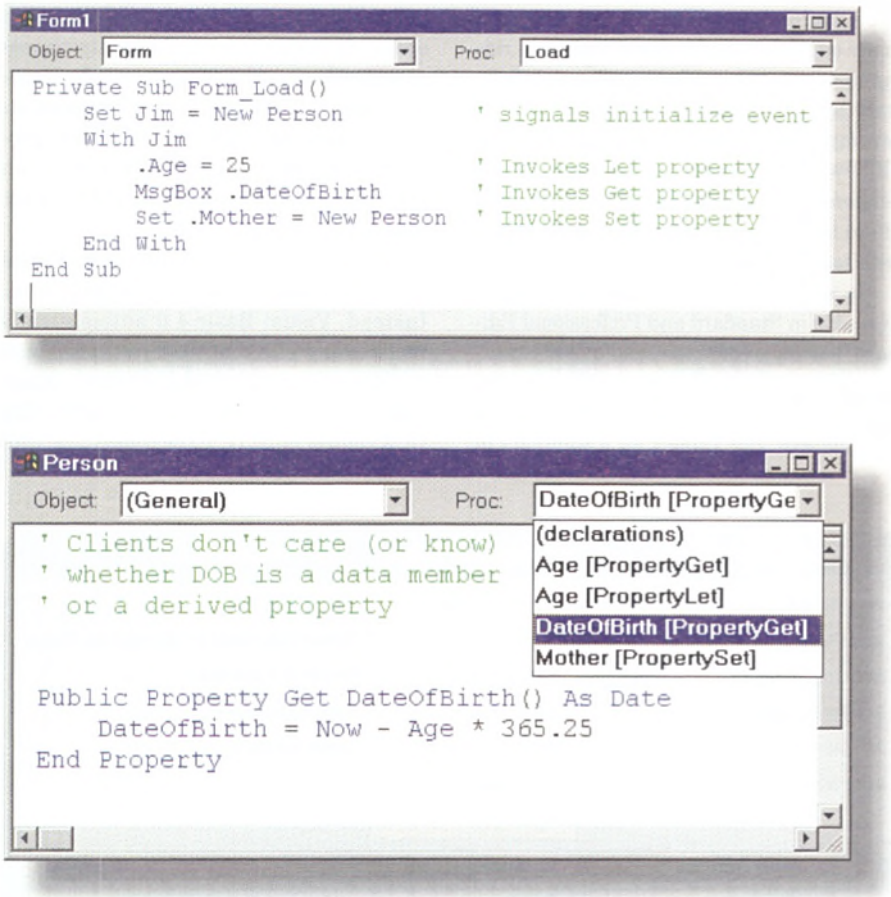


Figure 2 - Properties allow interface to hide implementation

This construct can be used to iterate over elements of an array as well.

It's a *class* thang!

Don't get too excited - VB 4.0 is *not* Object Basic...yet. But perhaps the most significant enhancement to the language offered in Version 4.0 is the notion of a *class module*. VB 3.0 allowed the programmer to define types, similar to Pascal `RECORD`s or C `struct`s, using the `Type...End Type` notation. There were no means of restricting access to the data contained in type instances, nor of defining initialisation and tidy-up routines, or other functions associated with a particular type. In other words, as with Pascal `RECORD`s and C `struct`s, there was no way of providing encapsulation, nor of defining constructors, destructors or ordinary methods.

Class modules are a step in an object-oriented direction for Visual Basic. A `ClassModule` object - stored in a corresponding `.CLS` file - is a design-time entity, similar to a standard `.BAS` module, that defines a *class*. (See Figure 2 for a taster of what's involved.) Classes behave pretty much like standard VB object types; for example, they can be instantiated with the `New` keyword in exactly the same way as built-in objects. When a new instance is created, a special `Initialize` event for the class is triggered. Similarly, when an object is destroyed, the class' `Terminate` event occurs. By supplying handlers (event procedures) for these events in the class module, the programmer can customise object construction and destruction.

As one would expect, the programmer can define `Public` and `Private` data, methods and properties for the class. A *method* is a function that can be applied to a particular instance of a class, using the familiar `Object.Method` syntax you'll have used with standard VB objects like `Form`s and `TextBox`es. Each method has an implicit `Me` parameter (analogous to C++'s `this` pointer) which refers to the object for which the method is being called.

User-defined properties are new in VB 4.0. Fans of object-oriented terminology will fondly recognise them as *accessors* and *updaters* (or *mutators*). In Visual Basic, a property is an attribute of an object that appears to clients of the class to be a data member, but which may in fact be implemented in some other way. As a simple example, consider the `Person` class of Figure 2. Two properties of a `Person` are `Age` and `DateOfBirth`. In implementation terms, only one of these two attributes has to be stored explicitly in a `Person` object: `Age` can be derived from `DateOfBirth`, and vice-versa. However, clients of the class need only know that a `Person` has an `Age` and a `DateOfBirth`, and should remain ignorant of how that information is obtained.

A property is implemented as a special kind of `Public` procedure. `Get` property procedures are called to retrieve the value of a property when required by client code; `Let` procedures are invoked when client code assigns to a property; `Set` procedures are called when an object reference is assigned to a property with the `Set` keyword. Examples of code that will invoke all three kinds of property procedure are given in Figure 2.

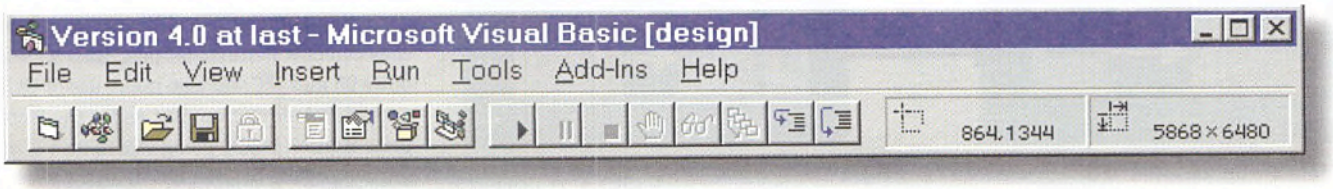
OLE Automation servers

Judicious use of class modules, properties and methods undoubtedly has great potential for qualitative leaps of organisation in VB programs. But the class module has still another role to play in the Microsoft scheme of things. Visual Basic 4.0, at least in its Professional and Enterprise guises, now allows the creation of OLE Automation servers, applications that expose part of their functionality through a standard OLE-Automation-compliant interface. When you create a `ClassModule` object (via the *Insert* menu), you'll notice that like any other Visual Basic object it has a set of built-in properties. As well as an innocuous `Name`, a `ClassModule` contains two other properties: `Instancing` and `Public`. By setting `Public` to `True`, the class is exported from your application as part of its OLE Automation interface.

Simply setting *StartMode* to 'OLE Server' in the Project sheet of your Options dialog box and typing a name by which your server application will be known causes an entry to be made in the Windows (or Windows NT) Registry next time you build the project. (The server can be installed in another machine's Registry by running the executable with the `/REGSERVER` command-line argument.) Thereafter, classes exported from your application will be instantiable and useable from any program that supports OLE Automation! The server will be loaded automatically by the OLE subsystem when a client request arrives.

Classes exported from your application will be useable from any program that supports OLE Automation!

The way class instantiation requests are handled depends on the `Instancing` property of the associated `ClassModule`. If set to 'Creatable MultiUse', multiple instances of the class can exist in the same server process. 'Creatable SingleUse' forces each request for object creation to create a new copy of the server running as a separate process. 'Not Creatable' means that the class



is *dependent*, ie can only be created from within the server itself and not from an external application.

It's also possible to build an 'in-process' OLE Automation server - essentially a DLL that runs in the same address space as the client process that loads it.

What's the significance of all this? For a start, VB programmers have never been able to share binary code, at least not without recourse to third-party products that allow a VB executable to export functions similarly to a DLL. Just as important, with the current trend towards distributed architectures, is the ability to partition applications into separate processes...

...enter Enterprise Edition

The very existence of an Enterprise Edition suggests that Microsoft sees Visual Basic playing a bigger role in the distributed systems of the near future. Until now Visual Basic has always been the 'client' in client/server, with its emphasis on prototyping and GUI development. The Enterprise Edition also includes *Remote Automation*, middleware that allows a client to access its OLE server over the network and which is likely to be part of Microsoft's forthcoming Cairo operating system. For the first time programmers have the opportunity to write distributed applications entirely in Visual Basic.

A Remote Automation client uses an OLE 'proxy' to communicate with its server. Only the proxy knows that the server resides on a remote machine rather than locally. The proxy employs Remote Procedure Calls (RPCs) to communicate with an Automation Manager process running on the server, a multithreaded application which routes each incoming client call onto its appropriate Automation server via an OLE stub. Figure 3 summarises this architecture.

Even though the Automation Manager is multithreaded, and so is capable of handling more than one client request at the same time, OLE Automation servers themselves are *not*. Multiple client calls to the same server are effectively serialised, potentially impacting on the responsiveness of the system. The manual suggests circumventing this problem (if it is a problem for your application) by setting the `Instancing` property of your classes to 'Creatable MultiUse' as described above, forcing each object into its own instance of the server. This solution places too many unnecessary architectural constraints on an application for my liking - for example, if each object resides in its own server process, it is more difficult for objects to share data (such as static class data).

Existing VB programmers out there will be happy to know they can now write 32-bit programs for Windows 95 and NT

The Enterprise Edition also includes Visual SourceSafe 4.0, a version control package for workgroup development that can be integrated with Visual Basic's IDE. Visual SourceSafe is a product in its own right which there isn't space to cover properly here. My first impression is that the user-interface is eminently useable, although it lacks support for long filenames in my beta copy. Both the user and administrator versions of Visual SourceSafe are supplied in 16-bit and 32-bit forms. The administrator version can import PVCS and Microsoft Delta databases.

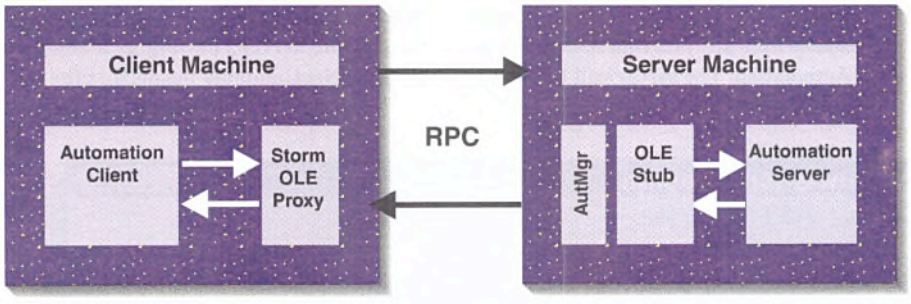


Figure 3 - Remote OLE Automation

Control	Description
`SSTab`	Tabbed property sheets (16- and 32-bit)
`ImageList`	Loads and stores bitmap images and icons
`ListView`	Display items in one of four views (Small Icons, Large Icons, List and Report)
`ProgressBar`	Shows the progress of a lengthy operation
`RichTextBox`	Like a `TextBox` but supports different fonts, point sizes, styles and colours
`Slider`	Slider with optional tick marks
`StatusBar`	Bar consisting of up to 16 panels of text and graphics
`TabStrip`	Strip of tabs containing text and graphics
`Toolbar`	Collection of `Button` objects
`TreeView`	Hierarchical list of labels with optional bitmaps

Table 2: - New OLE controls: all except `SSTab` are 32-bit only and require Windows 95 or NT 3.51

The verdict

Visual Basic was originally designed for rapid prototyping of front-ends. It sacrificed rigorous semantics and run-time efficiency for ease of use, an objective it achieved by providing the programmer with a highly interactive code-test-debug cycle and abundance of GUI-centric features.

VB 4.0's more disparate goals have met with only partial success. As far as the development environment is concerned, there has been little change. Even with the timely addition of class modules, the language itself is still leagues behind its closest competitor, Borland's Object Pascal (part of the Delphi programming environment). The adaption of VBA as the core language means true portability across Microsoft applications.

By adding the ability to create OLE Automation servers to VB 4.0, Microsoft is trying to scale Visual Basic up to a development environment suitable for writing server applications. In my opinion this is unlikely to succeed. Business-critical servers have certain requirements - robustness, speed and graceful degradation as the number of clients increases for starters. The first of these requires language support in the form of static checking of errors and proper type-safety; the second true compilation to native code. The third requires multithreading. VB 4.0 provides none of these.

Overall, however, Visual Basic 4.0 certainly betters Version 3.0. Existing VB programmers out there will be happy to know they can now write 32-bit programs for Windows 95 and NT and also make use of OCXs and OLE Automation. But as for the Enterprise Edition boldly going where no VB program has gone before - not likely!

Visual Basic Visual FoxPro Visual Objects
Delphi C/C++ Clipper PowerBuilder Lotus Notes

From desktop to enterprise, for the latest versions of the best developers' products and add-ons, call us today.

3D Graphics Tools	£115.00
3D Widgets	£75.00
†Aware/VBX	£110.00
ButtonMaker	£70.00
Calendar Widgets	£90.00
†CodeBasic	£129.00
CodePrint Pro	£79.00
Crescent Enquiry	£230.00
Crescent NetPak Prof	£129.00
†Crescent PDQComm	£95.00
Crescent PowerPak Pro	£625.00
†Crescent QuickPak Pro	£140.00
Crescent VBDD Datadict.	£129.00
Crescent Xref for VB	£79.00
Crystal Reports Pro	£295.00
Crystal upgrade from VB Pro	£149.00
†Custom Control Factory	£40.00
DataDirect Dev Toolkit 2	£495.00
DataDirect MultiLink/2	£289.00
DataDirect ODBC Pack	£345.00
†Data Widgets	£90.00
Desaware Cmn Dlg Toolkit	£30.00
†Designer Widgets	£90.00
Distinct TCP/IP SDK Vis.Ed.	£179.00
†First Impression	£125.00
First Impression OCX	£195.00
†Formula One	£125.00
Formula One OCX	£195.00
FXTools/VB Professional	£255.00
Gantt/VBX	£189.00
†Grid VBX	£70.00
†Graphics Server SDK OCX	£199.00
†HighEdit SDK	£195.00
InfoSleuth	£115.00
†ImageKnife 2.0 Pro	£275.00
†ImageStream	£125.00
Integra/VDB for VB Desktop	£199.00
†LeadTools 5 OCX Prof.	£365.00
†List & Label (VB)	£275.00
M.4 VB	£155.00
†MediaKnife 1.1	£275.00
†MicroHelp Comms Library	£109.00
MicroHelp Compression Plus	£170.00
MicroHelp Fax Plus	£170.00
†MicroHelp Muscle	£95.00
MicroHelp Network	£79.00
MicroHelp Spellpro	£95.00
MicroHelp Uninstaller	£49.00
MicroHelp VBTools 4	£95.00
†MicroHelp VBXRef	£79.00
†MicroHelp VBViewer	£70.00
Microsoft VB 4 Professional	£349.00
Microsoft VB 4 Enterprise	£689.00
MS VB4 Pro V. Upgrade	£115.00
MS VB4 Pro C. Upgrade	£209.00
MS VB4 Enterprise V. Upgrade	£379.00
Open Mail System (VBX)	£189.00
Rocket xBase	£195.00
Schedule/VBX	£189.00
†Spread VBX	£165.00
†SpyWorks	£115.00
†Tab Pro	£70.00
†ToolThings	£125.00
†TrueGrid Professional	£129.95
VBA Companion from Apex	£65.00
VBAssist	£125.00
VB-Cert	£99.00
†VBCompress	£95.00
†VBDB Code/Form Generator	£149.00
†VB Language Manager Pro	£195.00
VB ProjectWorks Pro 3 User	£225.00
VBtrv	£185.00
Version Stamper	£120.00
Versions/VB 1.1	£75.00
Visual Developers Suite Deal	£205.00
†Visual Speller	£125.00
†VisualWriter Pro	£155.00
†VSFlex OCX	£89.00
†VSVBX OCX	£40.00
†VSView OCX	£89.00

†Maestro Pro for Visual Basic	£695.00
†Maestro Desktop for VB	£195.00
†Notes Netware Server	£345.00
†Notes Starter Pack	£749.00
†Notes VIP Designer	£695.00
†Transporter (S server)	£2375.00

32-BIT NEWS!

OCX Versions. You're may be familiar with your new 32 bit Microsoft languages by now and wondering about availability of your favourite add ons. The following are available for both 16 and 32 bit development (ie inclusive in one package): GraphicServer; First Impression, Formula One, Calendar Widgets, Designer Widgets, LeadTools 5, VSFlex, VSVBX, VSView. More 32 bit OCXs are coming on stream every week; phone us for the latest or check out the OCX grid on our bulletin board.

Visual Basic 4 has arrived! We managed to get our hands on 50 Pro copies at the Microsoft launch party - did they disappear fast! We hope to have plenty of Pros and Enterprises in stock as you read this. Do note that the Pro version upgrade now costs £115 and includes paper docs. We also expect the Enterprise upgrade to include paper docs. There should be a Microsoft Computer-Based Training (CBT) on CD for VB4 coming out. It will cost £79.00.

Visual Foxpro upgrades have been selling steadily. Upgrade prices are for both version and competitive upgrades! We are waiting for the 'Mastering Visual Foxpro' CBT CD. It also will sell for £79.00

Upgrades. Check out the price lists for V (version upgrade) and C (competitive upgrade).

Apollo from SuccessWare	£115.00
†Async Professional	£135.00
Conversion Assistant Standard	£65.00
Conversion Assistant Database	£119.00
Delphi Client/Server	£799.00
Delphi CS upgrade from DT	£649.00
Delphi Desktop Comp. Upg	£129.00
Delphi RAD Pack	£129.00
Delphi Desktop plus RAD	£249.00
Distinct TCP/IP SDK Std	£375.00
†Graphics Server SDK	£199.00
ImageKnife Pro Version 2	£275.00
†InfoPower VCL + Source	£199.00
†InfoPower VCL	£150.00
†LeadTools Pro 5.0 DLL	£725.00
OMS Open Mail System	£379.00
†Orpheus	£135.00
Mobius Business Builder	£195.00
Mobius Drawkit	£95.00
Mobius WinG Sprite Kit	£95.00
Rocket from SuccessWare	£195.00
Silverware Win Comms Kit	£219.00

Calendar Widgets OCX	£90.00
CommTools	£225.00
Cryptor Windows DLL 6 user	£299.00
Cryptor DOS PLB 1 User	£99.00
Foxfire Developer's Edition	£299.00
FoxFix for DOS and WIN	£149.00
Graphics Server SDK	£199.00
Mac Rubel's Power Dev. Lib	£125.00
MIX (Single User)	£95.00
NetLib Network Library	£189.00
Raidar debugger	£175.00
ReFox Decompiler	£295.00
†SilverFox Comms (Win/ DOS)	£249.00
SymScript	£145.00
T-BASE Graphics	£399.00
The FEEL Editor	£85.00
Visual FoxPro Standard	£155.00
Visual FoxPro Professional	£389.00
VFP 3 Standard V/C Upgrade	£89.00
VFP 3 Pro V/C Upgrade	£229.00
Xilights Editor Enhancement	£99.00

Barcode Library for Windows	£249.00
Barcode Library for DOS	£389.00
BugTrak 2 (1 user)	£159.00
dBest Barcodes for Windows	£345.00
DemoQuick Express	£230.00
†DemoQuick Simulation Plus	£390.00
†Doc-To-Help Help Developer	£265.00
Quicture plus Smooth Scaling	£99.00
†ED for Windows v 3.5	£145.00
ED for DOS and Windows	£199.00
†InfoModeller DeskTop	£339.00
InfoModeller Server	£1385.00
†InstallSHIELD 3 Windows 3.1	£319.00
InstallSHIELD 3 Win'95, NT	£495.00
MS Office 95 Pro (Incl. Access)	£449.00
MS Office 95 Pro V. Upgrade	£449.00
MS SourceSafe 95 (platform)	£349.00
MS SourceSafe 95 Upgrade	£79.00
Microsoft Test 95	£419.00
Microsoft Test 95 V. Upgrade	£159.00
MS Windows 95 Upgrade 3.1	£69.00
MS Windows 95 Upgrade DOS	£125.00
MultiEdit Pro + Evolve	£155.00
PC-Install for DOS+Windows	£179.00
†R&R Rep Wtr Win/xBase 6.0	£195.00
†R&R Rep Wtr Win/SQL 6.0	£295.00
†SOS Help! for Win Info Author	£195.00
†Versions for Windows (single)	£195.00
Wise Installation System (Dev.)	£137.00

Borland C++ v 4.5	£275.00
Borland PowerPack	£69.95
Borland Visual Solutions Pack	£59.00
†CodeBase or CodeBase++	£235.00
CodeBase multi-platform	£699.00
Distinct TCP/IP SDK Pro	£525.00
Distinct TCP/IP SDK Std	£375.00
Formula One 32 bit	£195.00
Greenleaf Comm++	£195.00
Greenleaf Database Library	£195.00
†High Edit SDK	£195.00
Integra/VDB for C++ (Desktop)	£199.00
Integra/VDB for C++ (Server)	£499.00
†LeadTools 5.0 32 bit DLL	£965.00
List & Label (C/C++)	£389.00
MS VC++ Subscription 4.0	£349.00
MS VC++ Subs 4.0 V. Upgrade	£179.00
VBtrv for C++	£275.00
Watcom C/C++ (CD)	£199.00
Zinc App Framework	£call

Advanced Developers Toolkit	£385.00
Component Pack	£189.00
Desktop 4	£489.00
Enterprise	£3195.00
Erwin for PB Enterprise	£2395.00
FUNCky for PowerBuilder	£175.00
†PowerFrame App. Fr. Library	£305.00
†PowerFrame App. Sec. Lib.	£1150.00
PowerFrame Navigator Object	£99.00
PowerFrame Object Analyser	£99.00
PowerFrame TabFolder	£99.00

Adv. Xbase Server SDK	£69.00
Adv. Xbase Server 10 user	£1195.00
App:Convert2VO()	£145.00
App:HelpConstructor()	£145.00
CA Visual Objects	£399.00
Class:VBX	£219.00
†Graphics Server SDK	£199.00
LightLib Business Pro	£299.00
Rocket from SuccessWare	£195.00
Silverware Win Comms Kit	£219.00

Adv. Xbase Server SDK	£69.00
Adv. Xbase Server 10 user	£1195.00
Bandit Linkable Reporter	£199.00
†Blinker Linker	£179.00
CA Clipper 5.3 Comp. Upgrade	£125.00
CA Clipper 5.3 Upgrade	£85.00
CA Exospace	£95.00
†Class(y)	£109.00
†Clip-4-Win	£195.00
†Comix RDD	£99.00
†dGE Graphics	£189.00
†Escape Printer Library	£159.00
†Fast Text Search for Clipper	£70.00
†FAXual.lib	£79.00
Flexfile	£99.00
†FUNCky	£169.00
NetLib Network Library	£189.00
†NovLib Library	£189.00
†R&R Report Writer/DOS	£195.00
†Rescue Decompiler	£259.00
Scripton PostScript Library	£129.00
SilverClip SPCS Comms	£249.00
SOS Help! for Clipper	£165.00
†Summer '93 Code Optimiser	£159.00
†Telepathy Clipper Comms	£159.00
†T-BASE Graphics DOS or Win	£399.00
WBrowse for Clip-4-Win	£69.00

Borland InterBase Win NT	£880.00
Borland InterBase Novell NLM	£880.00
Borland InterBase 8 user dev.	£2509.00
Borland InterBase 16 user dev.	£4873.00
Borland Paradox Win	£349.00
Borland Visual dBASE	£255.00
Microsoft Access 95	£275.00
Microsoft Access 95 V. Upgrade	£89.00
MS SQL Server NT 6	£720.00
MS SQL Server NT 6 Upgrade	£429.00
MS SQL NT Client (1)	£120.00
MS SQL NT Client (1) Upgrade	£69.00
MS SQL Server 6 Client (20)	£1950.00
MS SQL 6 Workstation	£385.00
MS SQL 6 Wkst Upgrade	£199.00
Watcom SQL Developers' Ed.	£189.00
Watcom SQL Server 6 User	£529.00
Watcom SQL Server 16 User	£1085.00
Watcom SQL Server 32 User	£1965.00

†We have demos on our BBS for these products. Call 0181 747 1979, connect 2400 up to 28,800 baud, 8 data no parity 1 stop, to pick them up.

Call us today
0181 994 4842

QBS Software Limited
10 Barley Mow Passage, London W4 4PH
Ph: +44 181 994 4842, Fax: +44 181 994 3441
BBS: +44 181 747 1979

*Prices, correct at time of going to press, are subject to change and do not include shipping and VAT.

Visual dBASE
Borland's best kept secret?

Borland has released Visual dBASE - a Windows development complete with compiler. **Bob Rimmington** has a look at this latest attempt to recapture the glory days of Ashton-Tate.

A year ago a Windows database programmer had little choice. There was Visual Basic and Access but not much else. They had merits but lacked many of the features a programmer needs. Suddenly, this has changed and products have recently been arriving in quick succession. Delphi, Visual FoxPro, Visual Objects (VO) and now Visual dBASE (VdB). Out of these, only VO is certainly a first generation product (and it shows). VdB 5.5 is an upgrade of dBASE for Windows v5.0 with a host of new facilities, one of the most important being the long-awaited compiler.

dBASE has been around a long time, indeed some might say from the beginning of time itself as it pre-dates the first PC. The original version II evolved into the widely used and popular III+. There it languished until the ill-fated version IV arrived. Ill-fated as initially it was full of bugs and when these were sorted many still found it too cumbersome. Meanwhile, various competitors offered extra features, enough to spawn a more-or-less common language known as xBASE. Unfortunately the *x* could stand for Excessive Exceptions, as each rival added its own variations to the core language.

Doing it your way

Last year, Borland released v5.0 for DOS and a corresponding Windows v5.0, now replaced by VdB. It has all the sophistication we expect today and yet, remarkably, it is still largely compatible with earlier versions. A simple 14 year old dBASE II program ran completely unchanged. VdB does this with what I will call *Preferred Programming Interface* or *PPI*, not to be confused with Windows' MDI (Multiple Document Interface). There seems to be an alternative way of doing nearly everything. One can pick from menus, click on buttons or type in commands. For example, a table can be opened through the normal File menu, selected from a navigator window by clicking or opened from a command panel by typing `USE <filename>` just as in all previous dBASE versions. This is rather neat. Newcomers can virtually ignore the compatibility and do it all (well, most) by clicking, dragging and the use of objects while old hands can move from their familiar methods at whatever pace suits them.

Another illustration of PPI is what Borland calls 'two-way tools'. This allows a screen to be designed visually and any subsequent changes to the code it generates are then echoed back on the screen. In fact in many instances it could be called a 'three-way tool'. A button can be positioned by dragging, repositioned by editing the figures in the property inspector or changed again by altering the code. This is a typical example of PPI. The first two ways of editing change the code while you watch. The source code does need to be saved before it takes effect. Alas, only changes are two way and any extra lines of code vanish if re-saved from the design screen.

Database options

The traditional DBF table is the default option, but Paradox files are also handled directly and all the varieties of SQL, Access, Btrieve and others can be connected to through ODBC. The Client/Server option includes a local Interbase server, Borland's SQL Links and various Interbase and SQL guides, much as with the similar Delphi pack. SQL commands can be embedded in normal dBASE code and links set between different types of database.

Indexing options depend on the type of database. Paradox files allow only primary and secondary indexes by one or more fields. Far more scope is available for DBF files. Although the traditional NDX files are still available, the MDX production index files are best for most purposes. They can hold up to 47 different index sequences created using any combination of fields or part fields. Indexes may also be conditional, such as `INDEX ON TOWN FOR COUNTRY="UK"`.

Programming by design

The VdB interface resembles Delphi, but has additional command and navigator panels (see Figure 1). These neatly handle any interactive tasks and the selection of required directories and files. The starting point of any new program is a form. There is the now inevitable *expert* to assist. Even experienced programmers will find this useful as it alleviates the tedium of creating typical data-entry screens by placing fields from a selected table on the form. The positions and any descriptive text can then be changed if required through the designer window. Alternately, fields can be dragged from a field palette. Controls (such as an OK button) can be dragged from a controls palette on to forms and menus created (including a right-mouse button pop-up) and added. Properties may be modified and snippets of code attached to events. This can be done by typing in a *codeblock* (for readers unfamiliar with Clipper, codeblock is a data type to store commands or expressions) or by popping up a procedure editor. Typically some code will be attached to the `OnClick` event of a button. This could be a simple cancel instruction or a complex file processing routine.

With so many separate panels, the design-mode screen is very crowded – it's a pity that Borland has not used the Delphi trick of combining the speedbar buttons and components palette in one tabbed tool bar. Most users will

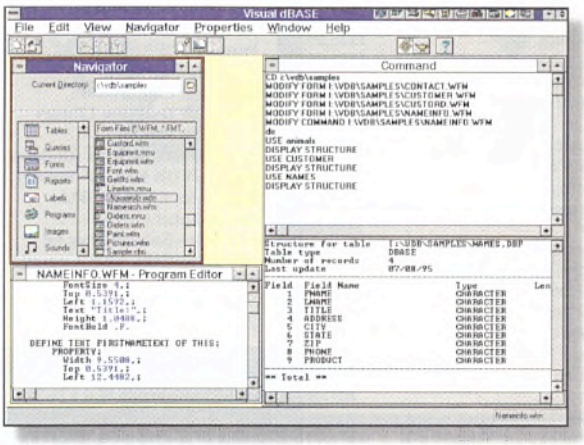


Figure 1 - Navigator and command windows

want to rearrange the panels to fit it all in (start planning for that 17" screen). See Figure 2.

Programming by code

A plus for programmers moving to Windows is that *all* the code generated by VdB is available for editing. Indeed, if you were a masochist, you could enter it all manually. What happens is that, as a form is designed and controls added, all the associated source code is placed in a WFM file. This will normally be called by linking it to an event but, as with the traditional PRG file, it can be called with a `DO ...` command. So every line of generated code is available to the programmer, with just one exception: VBX controls. Using a VBX attaches a BFM file containing binary information. For most non-visual tasks, the traditional PRG and procedure files are still used.

Is access to all the code actually useful? Yes, visual tools cannot do everything. For example, a text component puts text on a form, such as a heading. Adding `+CTOD(DATE())` to the relevant line of code is an easy way, perhaps the only way, of appending a date to the text. Comments too must be typed in. The code for a button may be self-evident, but six months later I may need to know why it is there. So new code may be written or existing DOS routines blended in.

The language itself preserves most of the traditional features but now includes a preprocessor, better scoping, objects, arrays and codeblocks. Memory variables, or *memvars*, can now be defined as `Local` and `Static` as well as `Public` and `Private`. However, predefining is still optional and typing continues to be a free and easy affair. Only your own discipline will inhibit assigning a number to a character variable. As seen before, PPI comes into play providing an alternative way of doing the same thing: creating objects to encapsulate memvars. Access to the generated code does help to understand objects and to see how they can be used. It is a good way of breaking the ice and introducing the benefits. See Table 1 for a list of standard classes.

The array itself is another example of our PPI rule. The command `DECLARE MyArray[10,2]` creates an array and `MyArray = NEW ARRAY(10,2)` creates an object called `MyArray` of class `ARRAY`. The result of both methods is identical. It can be passed as a parameter to functions such as `AFILL()` or the corresponding object method used. VdB does in fact include a second array class, `AssocArray`, that uses character strings as the index. See Table 2 for examples of arrays. Data in array elements can be of any type, including array types.

Something new, something old

A new feature is visual inheritance. The basic forms, controls and other objects can be modified, saved and then used in just the same way as supplied objects. A form can be designed with a heading, standard buttons and colours as a master template for an application. Any changes made to the master then apply to all other forms derived from it. In effect, it acts like a wordprocessor's stylesheet. Forms and dialog boxes include a page 0 on which can be placed any buttons and text that should appear on every page. Pages can be tabbed or scrolled. This new multi-page facility opens up all sorts of possibilities, for example a switch to another page via a 'Search' button to select an existing record. Controls, such as a frequently used validation routine to attach to a field entry, can be created and saved for reuse. However, such new controls cannot be used outside VdB. Forms can also be made *modal*, as normally used for dialog boxes. This may be needed to prevent a user from using Alt-Tab to switch to other software during entry of critical data.

For each new form or control there is a default set of properties. Depending on their type, properties can be changed by entering data, double-clicking, selecting from a list or, in some cases, visually editing them. Two examples are choosing a font with all its attributes and setting colours from a palette. Yet the old ways are still there: you can still enter `SET COLOR TO G/N`.

VBX controls can be imported (although version 1.0 only and no OCXs yet) and placed in the controls palette. Three examples, including a timer, are supplied. Borland has its own Visual Solutions Pack of VBX controls at extra cost (£69). It comes with a full manual plus three thin booklets describing how to incorporate the controls into VdB, Delphi and C++.

Array	*Menubar*
Assocarray	*Object*
*Browse**	*OLE**
*Checkbox**	*OLEautoclient*
*Combobox**	*Paintbox**
DDElink	*Popup*
DDETopic	*Pushbutton**
*Editor**	*Radiobutton**
*Entryfield**	*Rectangle**
Form	*Scrollbar**
*Image**	*Shape**
*Line**	*Spinbox**
*Listbox**	*Tabbox**
Menu	*Text**

**** can be selected from the Controls Palette***

Table 1 - List of standard classes

The *old* is represented by the command window. It allows all those little ad hoc tasks so beloved of clients to be done in the familiar way. Some program lines can be pre-tested this way and then pasted into the source code. A query designer includes various pick-and-paste facilities, an expression builder and a means of visually selecting related fields. If this is used with the command window open, all the generated code can be viewed, modified if needed and again pasted into a program. More PPI.

Where VdB really starts to score is in a typical complex application that needs a lot more than entry forms, queries and smart reports. The maturity that comes with 14 years of development means that there are commands

dBASE - A product with a history

dBASE is one of the oldest microcomputer development products around. First launched in 1980 as Vulcan, and subsequently as dBASE II (as is well known, there never was a dBASE I), Ashton-Tate's product grew to dominate its market in the early and mid-1980s. However, in the later part of the decade, near-compatible products such as Fox Software's FoxBase and Nantucket's Clipper began to make dBASE look old-fashioned and slow. Ashton-Tate reacted by fighting a look-and-feel suit against Fox, and by setting hundreds of developers to work on a new dBASE release.

In February 1989, *.EXE Magazine* published Bob Rimmington's review of dBASE IV. In the light of subsequent events, the article makes interesting reading. In his summing up, Bob said:

Perhaps Ashton-Tate should borrow BR's slogan, 'We're getting there'. It has now, on balance, caught up with Clipper and the like in respect of the facilities but still lags behind on speed [it ran at half the speed of Clipper in Bob's benchmark tests - Ed]... *It is puzzling how a package with so many obvious blemishes could have slipped through the quality control net - assuming there is one.*

dBASE IV became notorious for its bugginess and, despite a 1.1 release which went some way towards addressing the problems, Ashton-Tate was bought by Borland soon afterwards.

dBASE II vs. the Bilge Pumps.

and functions to handle most of those awkward and obscure client specifications.

Security and multi-user

For data entry, the default VdB method is to write straight to the fields in the table using the provided protection facilities. There is a `BeginTrans()` function with a `Commit()` or `RollBack()` option on completion (or failure). Alternately with DBF tables there are `BeginAppend()`, `IsRecordChanged()`, `SaveRecord()`, `AbandonRecord()` functions. You could still use an array or variables if you do not trust these, or build a scatter/gather routine. Alternatively use dummy files that mirror real ones executing a `COPY` or `APPEND` on completion. VdB includes various other safety options such as login, table and field security, and data encryption. A referential integrity check automatically protects records in linked files. Rules can be defined for Paradox or DBF files by adding `INTEGRITY` parameters to the `SET RELATION` command.

VdB also includes automatic file locking for multi-user systems and suggests that the `Rlock()` and `Flock()` functions need only be used to test for successful locking. No doubt experienced programmers will want to check reliability thoroughly especially in common situations such as a new invoice header record, a set of detail records, changes to a stock file or a set of audit trail records. However the automatic file locking and direct table entry do, in combination, offer a potential saving of many hundreds of lines of code in a large application.

Debugging is a dream...

At first sight, the debugger looks much like any other with source code, watch, break and call-stack windows. It also has its own toolbar, a handy addition. However, there are two less common features. First, if a coding error can be corrected within the offending line then it can be done on the fly, the source code file is updated and program execution resumed from that point. This is a great boon when testing database programs as it can avoid the repeated entry of test data in the form just prior to the error point. Setting break points can jump the program to the right place but will not automatically enter data.

The second feature is the `SET COVERAGE` command. When set `ON`, it writes to a file showing cumulative information on how many times, if any, program execution enters each logical block including control structures such as `IF`, `DO WHILE`, `DO CASE`, `FOR`, `SCAN`, `LOOP`, etc. So, it both acts as a profiler, and (even more valuable) identifies sections of code not used during a test run. That should help spot those elusive bugs that can evade normal testing. Unfortunately it appears only to work with PRG files, so WFM files created by the designer need to be renamed for this test. See Table 3 for an example of output.

...but printing can be a pain!

VdB does not handle Windows printing directly, but through the included Crystal Reports (version 3.0). It is not an ideal solution. It has a different look and feel, it lacks the speed of VdB and, though it can produce attractive output, there are too many limitations. Basic listings and forms are easy to design but the approach is quite different to VdB itself. It can also be slow. Only DBF and DB (Paradox) file types appear to be supported. Other database packages are bundled with ReportSmith or CA-RET. Messages on CompuServe forums suggest that none are popular.

The answer must surely be to include all the necessary printing tools in VdB itself. Some reports are just too complex for an add-on package. What if user responses are required while printing? Once again, though, PPI does come to the rescue! VdB still includes all the `SET PRINT ON` and `@ x,x SAY ...` commands and these by-pass the Windows print driver. So the sort of print routines used in DOS dBASE still work. Not ideal, but it does solve some problems.

How does it go?

Performance. Borland says VdB is 2 to 4 times faster than version 5.0. Well, it is faster; by how much depends on what you are doing. Screen updates certainly feel quicker and that is what concerns most users. Some simple tests of file processing (see Table 4) show only about 10% speed improvement, but it's much closer to Clipper times than I had expected. I think processing time is less important: users can do other things during a lengthy file update, so a few extra minutes will not usually matter. In one important respect, VdB now really flies. A last minute addition was a *SpeedFilter* feature. Long overdue in dBASE, Fox has had much the same for quite a while. This really speeds up queries, even without a filter. For example, a `COUNT` for `"Smith"` in an unindexed *Name* field of a 12,000 record file took 12 seconds the first time. Subsequent counts for Smith and any other name were virtually instant. With an index, it was instant the first time. A simple `SET FILTER TO ...` can now be used with browse as matching records appear at once and PageDown is similarly immediate. So no more need to code work-arounds setting bounds as an alternative to a filter.

So far programs have proved bug free, but the VdB desktop software is not 100% robust. There is a steady GDI leak. With just VdB loaded, 68% free resources was reported but this would drop after a couple of hours use, even if all tables and forms were closed. In theory, it could drop below the minimum 57% seen; in practice an error message of the type 'Please close files and exit' also arose after every two- or three-hour session.

Traditional usage

```
DECLARE MyArray[6,2]
AFILL(MyArray,1) && = function
? MyArray[3,2] && Returns 1
```

Array as an object

```
MyArray = NEW Array(6,2)
MyArray,Fill(1) && = method
? MyArray[3,2] && Returns 1
```

Sparse arrays (only needed elements are created)

```
oMyArray= NEW Object()
oMyArray[1930] = "Born"
oMyArray[1950] = "Married"
oMyArray[1990] = "Retired"
oMyArray[2005] = "Died"

? oMyArray[1990] && Returns "Retired"
```

Associative arrays
(these index on labels instead of numbers)

```
PartNumber = NEW AssocArray()
PartNumber["ABC123"] = "Widget"
PartNumber["DEF234"] = "Widget Holder"
PartNumber["GHI345"] = "Widget Fixings"
PartNumber["JKL3456] = "Widget Carton"

? PartNumber["DEF234"]
          && Returns "Widget Holder"
```

Table 2 - Array examples

Windows 95 and all that

VdB and programs written with it support basic Windows 95 features such as long file names, extended file attributes and the latest Microsoft GUI standard. Two new OLE classes are now included and the two DDE ones retained.

VdB maintains the dBASE tradition of proper user guides. No longer the ring binders, slip case and quality paper of III+ in its heyday but at least a full set of printed books are supplied. 'Acrobatic' (what an appropriate name!) on-disk manuals are just copies, thank goodness. The printed documentation includes a *User Guide*, a *Programmers Guide* and a com-

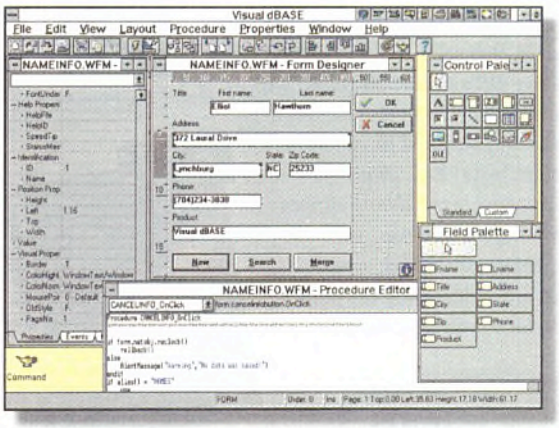


Figure 2 -Designer screen

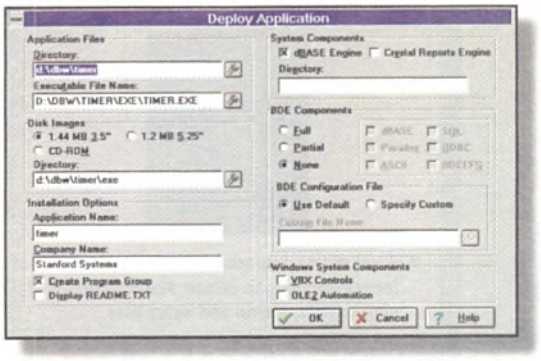


Figure 3 - Compiler Dialog Box

prehensive *Language Reference*, plus *Quick Reference* and Crystal Reports booklets: 2,000 pages in all. A lot to read, but it is well presented, well explained, well illustrated and sports plenty of examples. An SQL Links guide and three Interbase guides are included with the Client/Server pack.

The compiler

A 'compiler' (with Resource Workshop) is included in the Client/Server pack. It is a bit of a misnomer, as all source code files auto-compile into object files when first run. The so-called compiler actually assembles an EXE file and builds installation disks. The end result looks good, but getting to this point did reveal a few odd features.

The compiler Readme file says instructions are in the *Programmers Guide*. Not so: there are no printed instructions, only some help screens. The Workshop has an Acrobat *User Guide*, according to the CD contents card. Again not so: there's only a brief Readme file that clearly assumes it is part of the Delphi package! Perhaps the Workshop is thought to be too intuitive to need a manual, yet Borland provided a 300-page guide for Turbo Pascal for Windows users. There is a separate facility to create help files.

```
Block 1, line 38 to 52, was exercised 1 times.
Block 2, line 56 to 99, started 1 times and finished
0 times.
Block 3, line 102 to 127, was exercised 1 times.
Block 4, line 134 to 140, was exercised 1 times.
Block 5, line 144 to 147, was exercised 3 times.
Block 6, line 151 to 154, was exercised 3 times.
Block 7, line 159 to 168, was exercised 1 times.
Block 8, line 175 to 176, was exercised 0 times.
Block 9, line 187 to 188, was exercised 0 times.
Block 10, line 198 to 204, was exercised 2 times.
Block 11, line 206 to 208, was exercised 2 times.
Block 12, line 210 to 211, was exercised 0 times.
Block 13, line 221 to 364, was exercised 1 times.
Block 14, line 373 to 378, was exercised 0 times.

9 blocks out of 14 blocks tested.
Test coverage is 64.29% complete.
```

Table 3 - Example of coverage report on sample program DBCLOCK.PRG

The compiler adds two options to the program menu, 'Build Executable' and 'Deploy Application'. Strangely, this menu appears only when a file is open for editing, although the Build and Deploy actions assume compiled and tested files. The dialog boxes are deceptive, implying that various types of file can be accessed from individually specified drives and directories. Not quite so. Although a normal compile will link in any procedure and similar files from another directory an EXE build will not. However, it may well include unwanted files (such as source code). Similarly, the Deploy operation will put all the files it finds in the specified directory onto the install disks. The solution is first to create project and application directories. All the required object files should be placed in the former and the Build option will put these into an EXE file. All files required by the user (the EXE, DBF and MDX files, and any thing else relevant) should then be placed in the application directory. The Deploy option will then build disk or CD-ROM images comprising your development files and all supporting files (see Figure 3). The latter includes a 1.35 MB DB55RUN file and various DLLs which means that even a trivial program will need two 1.44 MB disks.

Creating the directories and copying the files is best done outside VdB using File Manager or similar. If you have a library of your own PRG, procedure and custom control files, you will need to know every file used and move them to the project directory. It is really all a bit clumsy: it reminded me of early DOS systems. Perhaps Borland were too ashamed to set this out in printed instructions for all to see!

Installation of the generated application is visually slick. It compares well with purchased software and includes the Borland car dashboard progress screen to add the professional touch. A *splash* screen and a startup icon can be included. The program then runs much as it would in VdB itself. This might be expected as the object files, embedded in the EXE, are presumably working through the runtime file. So no change to performance.

Summing up

Time to nail one's colours to the mast. Many programmers, with a background of dBASE and Clipper, are looking for the right route into Windows. Visual Objects has an attractive approach and potential, but is not easy to learn and really needs machines better specified than most now in use. Delphi is good too, similar in many ways to VdB, but the Pascal language is not for everyone, the documentation is poor and it lacks some data manipulation tools. VdB is not perfect, the print facilities could be better and there are limitations when compared to Clipper. However, it is easy to use and it does have those interactive and command-line options. There are the splendid debugging tools and, of course, the invaluable PPI. Is it just possible that Borland has produced a worthy successor to dBASE III+ at long last? If they have, they must feel rather embarrassed by it, the launch itself has been anything but *visual*! ■

Bob Rimmington is an independent xBASE and Clipper consultant. He can be contacted through Stanford Systems on 01444 236352. Email: rsr@stnfrdbh.demon.co.uk (or CompuServe: 70630,1365).

VdB on its own costs £349 (upgrade £59); the compiler costs £349 and the Client/Server version which includes the compiler costs £499 (upgrade from any version of dBASE for £199). Borland is on 0800 212727.

Program uses 3 files

File A has approximately 2200 records with a 'Code' field.
File B has approximately 12500 records with a 'Code' field.
File C has four fields and initially nil records.

Program indexes file B on Code.

Then loops through every record in file A.
Appends record to C for each match found in B.
Writes to each new record in C:

- *Field 1 replaced with Code*
- *Field 2 replaced with 100 char string extracted from fields in A and B and then concatenated*
- *Field 3 replaced with a calculated figure*
- *Field 4 replaced with a calculated date*

Overall, 12500 records written to file C.

The program and DBF files were held on a RAM disk to eliminate effect of hard disk performance. The code for each test is identical with timing from a start point. Eg, in VdB, a Go button is clicked to start processing.

Times recorded:

dBASE III+	*9.45 minutes*
Clipper 5.2d	*1.54 minutes*
dBWin 5.0	*2.49 minutes*
VdB 5.5	*2.27 minutes*
VdB 5.5 compiled	*2.27 minutes*

Table 4 - Performance figures

DISTINCT TCP/IP VISUAL EDITION

- **Create VB TCP/IP applications without making a single DLL call**
- Upcalls and polling are now presented as VB events delivering incoming data, status change and errors
- Access to Windows Sockets, FTP file transfer and TelNet connectivity

£265

VIDEOSOFT VSFLEX

- **Flexarray Control for displaying tabular data, compatible with VB Grid Control. Similar to a spreadsheet allowing display, sort, merge and format string and picture tables**
- Individual cell formatting and cell merging
- Development layout designer for easy grid design
- Row and column moving at runtime
- FlexString regular expression engine providing search and replace capabilities to parse input strings.

£105

GREY MATTER

Prigg Meadow, Ashburton
Devon TQ13 7DF

Prices do not include VAT or other local taxes, but do include delivery in the UK and Europe. Please check prices at time of order as ads are prepared some weeks before publication. This page lists some products – call us for a complete price list.

ORDER BY PHONE WITH YOUR CREDIT CARD

TEL: (01364) 654100

FAX: (01364) 654200

IMAGEMAN/VB

- **A replacement data-aware picture control for VB**
- Industry standard file formats, TIFF, PCX/DCX, BMP, DIB, IMG, WPG, GIF, WMF, EPS, Targa and JPEG
- Full support for TWAIN compliant input devices
- Dithering, brightness, gamma, colour, reduction, scrolling and panning, rotation, zooming with "rubber banding"
- High quality printer output

£239

MICROSOFT VISUAL BASIC ENTERPRISE

NEW V4.0

- **Professionally create Windows, Windows NT and Windows 95 applications with all the Professional features**
- Team RAD tool for three-tier, distributed client/server development
- Includes Microsoft Visual SourceSafe and Component Manager for large scale team development
- Remote Data Control for SQL Server and the ability to create and remotely execute OLE servers for true three-tier distributed solutions

£715

MICROSOFT VISUAL BASIC PROFESSIONAL

UPGRADE TO V4.0

- **Why buy a full product when you can upgrade?**
- Upgrade from Visual Basic Professional v3.0 to Visual Basic Professional v4.0

£115

- Upgrade from Visual Basic Professional v3.0 to Visual Basic Enterprise v4.0

£290

- Competitve upgrade to Visual Basic Professional v4.0

£218

DOC -TO-HELP

- **Easily create and maintain Windows on-line Help and printed documentation from a single source**
- Convert your documents into on-line Help with hypertext jumps, help macros, popup definitions & keyword searching
- Auto format headers, footers, sideheads, table formatting, even & odd pages, gutters, page numbering & much more
- Customisable template layouts you can layout, format and create with a simple point-and-click process

£280

GRAPHICS SERVER 4.0

- **Using the GRAPH.OCX in VB Pro? You're missing out on:**
- True 3D rotation
- Interactive toolbar
- Combination graphs
- Hot graphs for EIS drill-down
- Over 100 new properties
- **NEW in latest release 4.02**
- Data binding to VB Data control
- Graph templates you can save

£210

DESAWARE

SPYWORKS 4.0

- **Do virtually anything in VB that is possible in other languages**
- Provides advanced subclassing, windows, hooks, callbacks, debugging tools and more
- 16 & 32 bit OCX for VB 4.0

£115

VERSION STAMPER

- **Eliminate incompatibility problems when distributing component based applications**
- Embed into your executable version information about all of the DLL's and components used
- 16 & 32 bit plus 16 bit VBX

£115

STORAGE WORKS – VB 1.0

- **The ultimate storage & file manipulation toolkit for VB and other OLE clients**
- Work with OLE 2.0 structured storage files from within your applications
- Create resource files and more
- 6 & 32 bit plus 16 bit VBX

£115

PROGRAMMING TOOLS

Ada	Assemblers	Basic
C/C++	Comms	Cross Dev
Custom Controls	Database	Debuggers
Delphi	Editors	Fortran
Graphics	Linkers/Locaters	Modula-2
Pascal	Prolog	Smalltalk
SQL	Version Control	Visual
Programming	Windows	Xbase

Please phone for full price list

OLE Automation in Delphi

Delphi's VCL does not support OLE Automation. But **Phil Wigglesworth** has tracked down the right calls to put your Delphi application in charge of Microsoft Word.

OLE automation is a major extension to Microsoft's OLE (Object Linking and Embedding) API, which delivers the true power of the interface. Put straightforwardly, automation allows one program to call another and then use an API of functions within that other program.

Automation started 'simply' with Microsoft exposing a programmatic interface to get at the macro languages of its major applications. Word 6.0 offers a massive monolithic object called `Word.Basic` which provides access to 899 functions, broadly corresponding to the Word Basic language. The Excel team, in contrast, worked alongside the OLE team with a new object-oriented approach to produce many smaller objects bundled together in one huge Excel object. Each sub-object is of manageable size, so once the `Excel` object is loaded, the individual objects are fast. Microsoft twigged that this technology could be used to create other lightweight reusable components, and the concept of OCX was born.

To explore this strange new world, dubbed by Microsoft 'The Future', I chose to work with a tool *not* developed by Microsoft - not Visual Basic or Access - but Borland's Delphi.

From Pascal to OLE

The standard C header files for the OLE libraries have already been translated into Object Pascal in the form of two minimal OLE wrappers supplied with Delphi - `OLE2` and `Dispatch`. `OLE2` provides the basic object manipulations for Object Linking & Embedding, while `Dispatch` provides all the juicy bits that make automation come alive.

However, Borland doesn't provide any online help or manuals on the OLE functions, and the interface isn't for the faint hearted. Back to Microsoft documentation. A quick scan of the *Developer Network CD* reveals - online - the entire text of *Inside OLE 2* by Kraig Brockschmidt with a small footnote to the affect that the second edition will be added as soon as it is available. A check with the book shop, a flourish of that invaluable programmer's tool the credit card, and the second edition arrives on my desk the next day. Weighing in at a mere 1194 pages, this provides an excellent introduction to all the features of OLE.

... But I only want to call a few functions in another program! Can this be right? Fortunately not. This book is mostly about the *implementation* of objects to be linked, embedded and automated. The art of actually *calling* the objects is assumed to be left to the Microsoft heavyweights of VB and Access. However, Kraig is clear - if not concise - and scattered among the pages are all the tools required to build an automation controller.

Initialising OLE

Before starting any real programming, we need to initialise the OLE libraries with a call to the `OleInitialize` function - this takes an ubiquitous single parameter `pvReserved` that must be `nil` . Although OLE is available throughout Windows, you are initialising the library that will run inside your program space and this activates the appropriate DLLs. It returns a strange handle that can be turned into a boolean result using `SucceededHr`. This result must be stored. A failure only means that the OLE library has already been initialised - within your program - and no further initialisation is required. When we finish we call `OleUnInitialize` only if we got a successful result.

```
Unit WordBasi;
uses Ole2, Dispatch;

TWordBasic = class
  fInitialized: boolean;7
public
  constructor Create;
  destructor Destroy;
end;

constructor Create;
begin
  fInitialized := SucceededHr(
    OleInitialize (nil) ) ;
end;

destructor Destroy;
begin
  if fInitialized then
    OleUnInitialize;
end;

end.
```

Next, decide on the object you want to automate. Pick something you are familiar with as you need to recognise each step on your path to automating it. I have chosen the Word 6.0 `Word.Basic` object. You will need to identify the name of the object. Don't expect it to be self-evident. The `Word.Document` object

```
{ save CLSID in new private variable fClsId }
if FailedHr ( CLSIDFromProgId ( 'Word.Basic', fClsId ) then
  raise Exception.Create ( 'OLE Failed to convert ClsId from String' ) ;
```

Figure 1 - Creating a ClsId from ProgId

```
( save the IDispatch interface in a (new) private variable pDispatch )
if FailedHr ( CoCreateInstance ( fClsId, nil, CLSCTX_LOCAL_SERVER, IID_IDispatch, pDispatch) ) then
  raise Exception.Create ('OLE: Failed to create instance of object' ) ;
```

Figure 2 - Creating an instance of the object

```
const
  MaxArgs = 50;
type
  TWordBasic = class
    { must call OleUninitialize? }
    fInitialized: Boolean;
    { CLSID of our object }
    fClsId: CLSID;
    { automation i/face to object }
    pDispatch: IDispatch;
    { Language: 9 for English }
    fLocale: LongInt;
    { Parameters being passed }
    fParams:  DISPPARAMS;
    { Result when Function called }
    fVarResult : PVARIANT;
    { No Result when Proc called }
    fpNoVarResult : PVARIANT;
    { Did the call work? }
    fSucceeded: Boolean;
    { Exception info when fails }
    fExcepInfo: ExcepInfo;
    { Argument at fault if failed }
    fArgErr:  Integer;
    { Arguments being passed }
    fArg:  array [0..maxArgs-1]
       of VariantArg;
    { Named arguments being passed }
    fNamedArg: array [0..maxArgs-1]
      of DispId;
public
    constructor Create;
    destructor Destroy;
  end;

constructor TWordBasic.Create;
begin
  fInitialsized := SucceededHr (
              OleInitialize (nil) );
  if FailedHr ( CLSIDFromProgId (
     'Word.Basic', fClsId) ) then
    raise Exception.Create (
     'OLE: Failed to convert ClsId'+
      ' from String' );
  if FailedHr ( CoCreateInstance (
    fClsId, nil,
CLSCTX_LOCAL_SERVER,
     IID_IDispatch, pDispatch)) then
    raise Exception.Create (
     'OLE: Failed to create '+
       'instance of object' ) ;
  if pDispatch=nil then
    raise Exception.Create(
     'OLE: No IDispatch Created' );

  { country to use for
    interpreting object names }
  fLocale := 9;
  { set pointer to standard
     argument list }
  fParams.rgvarg := @fArg;
  fParams.rgdispidNamArgs :=
    @fNamedArg;
end;

destructor Destroy;
begin
  { release our IDispatch i/face }
  if pDispatch<>nil then
    pDispatch.Release;
  { release our use of OLE libs }
  if fInitialized then
    OleUnInitialize;
end;
```

Figure 3 - Core class for automating an object, including parameter definitions

appears OK until you discover that it doesn't support OLE Automation: for that you will require the `Word.Basic` object. As a general rule, use the RegEdit program (located in the Windows directory) to find objects with the */Automation* flag. These are normally the only formats that support OLE Automation.

But OLE can't work with anything as simple as a name. It requires a unique identifier called a `CLSID` (Class Identifier). This is a 16-byte number that is globally unique. The function `CLSIDFromProgId` can be called to convert an object name - a `ProgId` - to a corresponding `CLSID` - see Figure 1. If this call fails, then your `ProgId` is not in the Windows Registry. In our example, you need to make certain that Word 6 is fully installed on your system.

Creating the OLE object

We can now create the OLE object, using the value saved in `fClsId`, with the call to `CoCreateInstance` shown in Figure 2. In this code we instantiate a *local server*, which means it runs as a separate program rather than running as a DLL within our process space. In addition, we state that we are looking for an `IDispatch` interface (by supplying the `IID_IDispatch` argument) and that the result should be stored in the `pDispatch` variable.

There are a number of interfaces exposed by an OLE object, but the `IDispatch` interface is what interests us. An object which does not provide the `IDispatch` interface does not support automation. The `pDispatch` variable is of type `IDispatch`, a class defined in DISPATCH.PAS which exactly matches the OLE definition of the `IDispatch` interface. *(The definition is also given in DISPATCH.INT in the DELPHI\DOC directory - Ed.)*

It is at this point that your object gets fired up - Word in this case - if it is not already running. We all know how long it takes to start up Word, so make certain that you keep it loaded at all times.

Unfortunately, although `CoCreateInstance` may report a successful creation of an object instance, the object may not provide an `IDispatch` interface. So check that `pDispatch` actually refers to an `IDispatch` object before proceeding:

```
if pDispatch=nil then
  raise Exception.Create;
```

We now have a working object, but there is one more piece of OLE housekeeping before we can get to the meat. In the `Destroy` destructor, we need to add the line:

```
{ release our IDispatch interface }
if pDispatch<>nil then
  pDispatch.Release;
```

This tells OLE that we have finished with the interface. OLE keeps track of each interface as it is used and releases the whole object only when each of the individual interfaces have been released.

Finally, before we can call our first function, we need to set up some data structures. Figure 3 shows our core class for automating an OLE Automation server, including definitions for parameters we will need later.

Invoking a procedure

We now have an up-and-running intertface object that we can use to call any procedure or function the OLE application exposes. `CallProc` in Figure 4 wraps up the code required to call any procedure that does not *require* any parameters. In fact, most `TWordBasic` procedures do not require any parameters, and use their own defaults when none are supplied.

The `Invoke` function actually calls the OLE object's procedure. `MemId` identifies to the object which procedure is to be called. The `fpNoVarResult` variable, as part of our class, will have been initialised to `nil`. By de-referencing this variable, we are able to pass a `nil` pointer to the underlying function which, because that function checks for a `nil` value, causes no exception problems. `fExcepInfo` and `fArgErr` are used to handle any error.

A simple example of `CallProc` doing its stuff:

```
WordBasic := TWordBasic.Create;
WordBasic.CallProc (1);
MessageBox ( 'Help Displayed' );
WordBasic.Free;
```

```
procedure TWordBasic.CallProc ( MemId: MemberId ) ;
begin
  { Invoke as function with a return value }
  fSucceeded := SucceededHr (
    pDispatch.Invoke ( MemId, IID_Null, fLocale, DISPATCH_METHOD,
    fParams, fpNoVarResult^, fExcepInfo, fArgErr) ) ;
  if not fSucceeded then
    Raise Exception.Create ('OLE procedure  ' + IntToStr (MemId + ' failed' ) ;
end;
```

Figure 4 - Calling a procedure in the OLE Object

This calls the *Help* function in Word - not particularly exciting. So now let's create a new Delphi form and add a `TWordBasic` object to it. Create and free the object in the form's own `Create/Destroy` methods. Drop a button onto the form, change the caption to 'Print' and add the single line of code below to the `OnClick` handler:

```
WordBasic.CallProc (88);
```

Having compiled the program, load a document into Word, switch back to your form and press the button. Your document will be printed. Now we are motoring!

Parameter types

You can get a real kick from calling Word procedures directly from your Delphi program. Finally all those fancy words really mean something that you can actually code. But come on, be serious! Without parameters for your procedures, you are a little limited in the real world. So let's move up a gear and look behind those structures referenced in Figure 3.

Before we hit the code, I will give you a little theory and a touch of practical experience. OLE only knows about one type of variable. This keeps it simple - in theory - and totally messes you up - in practice. All variables are of 'variant' type, the union of all the types listed in Figure 5. Which means I fibbed, because you can pass any of the common types via a variant. Of course, you still need to pass a variable of the correct type as the argument, otherwise your object will give you an error.

This is not too bad for Word which has sensibly designed all its parameters to require variables of the appropriate type. But Excel will give you much more fun. You will notice that id 12 is `VT_VARIANT`. A bit weird. But worse, **all** Excel parameters - and I do mean all - are defined to be of type `VT_VARIANT`. So you have to pass a Variant structure pointing at another Variant structure that actually contains the data. Apparently the Excel object definitions and the OLE definitions were evolving at the same time. Both teams decided they needed variant structures for their parameters. Both teams implemented them. Deadlines didn't allow the Excel team to re-jig their code and we have to live with the result.

But fortunately, we are implementing the `WordBasic` object. Now, the Word team chose to expose the Word Basic language using roughly the original syntax for their parameters. So, while we still have integers when we would expect to pass boolean values, we can enjoy simple strings and integers elsewhere.

Ah! Simple strings. Perhaps a slight slip. OLE has invented a new type of string for us: the `BSTR` which looks very much like a null-terminated C style string. If you look in DISPATCH.PAS, it is even declared as a `PChar`. But as these can be long strings, searching for the length every time you needed to manipulate them would be very time consuming - and these strings must be copied across process boundaries for every OLE automation out-of-process server. So the designers of OLE automation added a `DWORD` length before each string. (Actually, `BSTR` stands for 'Basic String', and was included for compatibility with Visual Basic, promoting that language as the *de facto* standard for OLE automation controllers).

A set of routines is provided in DISPATCH.PAS to work with strings (see Figure 6). Each string is allocated out of global memory - not your buffer - and controlled by these API routines. The simple rule to follow is:

- If you allocated the string, remember to de-allocate it.
- If the system allocated the string, remember to de-allocate it.

This can be a bit of a pain when the system keeps allocating strings that you don't really want. Examine any data structures carefully, and check that the system is not in-filling them with unwanted - but persistent - strings.

Type	Id	Description
VT_EMPTY	0	Nothing
VT_NULL	1	Null
VT_I2	2	2 byte signed integer
VT_I4	3	4 byte signed integer
VT_R4	4	4 byte real
VT_R8	5	8 byte real
VT_CY	6	Currency
VT_DATE	7	Date
VT_BSTR	8	Binary String
VT_DISPATCH	9	IDispatch Interface
VT_ERROR	10	Scode
VT_BOOL	11	Boolean
VT_VARIANT	12	Pointer to Variant
VT_UNKNOWN	13	Iunknown Interface
VT_WBSTR	14	Wide Binary String
VT_I1	16	Signed Char
VT_UI1	17	Unsigned Char
VT_UI2	18	Unsigned Short
VT_UI4	19	Unsigned Short
VT_I8	20	Signed 64-bit integer
VT_UI8	21	Unsigned 64-bit integer
VT_INT	22	Signed machine integer
VT_UINT	23	Unsigned machine integer
VT_VOID	24	C style void
VT_HRESULT	25	HResult
VT_PTR	26	Pointer
VT_SAFEARRAY	27	Safe Array
VT_CARRAY	28	C style array
VT_USERDEFINED	29	User Defined Type
VT_LPSTR	30	Null terminated string
VT_LPWSTR	31	Wide null terminated string
VT_FILETIME	64	File Time
VT_BLOB	65	Length prefixed bytes
VT_STREAM	66	Name of the stream follows
VT_STORAGE	67	Name of the storage follows
VT_STREAMED_OBJECT	68	Stream contains an object
VT_STORED_OBJECT	69	Storage contains an object
VT_BLOB_OBJECT	70	Blob contains an object
VT_CF	71	Clipboard format
VT_CLSID	72	CLSID

Figure 5 - Variant types

Using a parameter

After wading through the documentation, the source headers, and generally getting lost, the use of parameters can be daunting. But I have already added the declarations to our class to cope with the parameters and to get started we only need to fill in a few simple fields.

As a first step, lets pass an integer to our `WordBasic` object. Add a new class procedure `Vline(I:Integer)` that moves the cursor by the number of lines supplied as a parameter.

```
procedure TWordBasic.Vline
  ( I: Integer );
begin
  with fArg[0] do
  begin
    iVal := I;
    VT := VT_I2;
  end;
  fParams.cArgs := 1;
  CallProc( 32820 );
end;
```

We fill in the first argument (zero based) with the actual value we want to pass and tell it the particular variant type we want to use. In all cases, we use the `Variant` definition - shown in Figure 7 - to select the appropriate variable to update. Remember that only the variable corresponding to the `vt` definition will be used by the OLE procedure. All other variables are mapped over the same physical memory space and are therefore invalid.

Having established our parameter, we need to tell `IDispatch` how many parameters

```
function SysAllocString(const p1: PCOLESTR): BSTR;
function SysReAllocString(var p1: BSTR; const p2: PCOLESTR): Integer;
function SysAllocStringLen(const p1: PCOLESTR; p2: Integer): BSTR;
function SysReAllocStringLen(var p1: BSTR; const p2: PCOLESTR; p3: Integer): Integer;
procedure SysFreeString(bstr: BSTR);
function SysStringLen(p1: BSTR): Word;
```

Figure 6 - BStr API

```
VARIANT = record
  vt: VARTYPE;
  wReserved1: Word;
  wReserved2: Word;
  wReserved3: Word;
  case Integer of
    VT_I2:              (iVal: Integer);
    VT_I4:              (lVal: Longint);
    VT_R4:             (fltVal: Single);
    VT_R8:             (dblVal: Double);
    VT_BOOL:             (vbool: BOOL);
    VT_ERROR:           (scode: SCODE);
    VT_CY:                (cyVal: CY);
    VT_DATE:             (date: DATE);
    VT_BSTR:           (bstrVal: BSTR);
    VT_UNKNOWN:     (punkVal: IUnknown);
    VT_DISPATCH: (pdispVal: IDispatch);
    VT_ARRAY:       (parray: SAFEARRAY);
    VT_BYREF or VT_I2:
                     (piVal: ^Integer);
    VT_BYREF or VT_I4:
                     (plVal: ^Longint);
    VT_BYREF or VT_R4:
                    (pfltVal: ^Single);
    VT_BYREF or VT_R8:
                    (pdblVal: ^Double);
    VT_BYREF or VT_BOOL: (pbool: ^BOOL);
    VT_BYREF or VT_ERROR:
                     (pscode: ^SCODE);
    VT_BYREF or VT_CY:   (pcyVal:  ^CY);
    VT_BYREF or VT_DATE: (pdate: ^DATE);
    VT_BYREF or VT_BSTR:
                     (pbstrVal: ^BSTR);
    VT_BYREF or VT_UNKNOWN:
                   (ppunkVal: ^IUnknown);
    VT_BYREF or VT_DISPATCH:
                (ppdispVal: ^IDispatch);
    VT_BYREF or VT_ARRAY:
                   (pparray: ^SAFEARRAY);
    VT_BYREF or VT_VARIANT:
                    (pvarVal: PVARIANT);
end;
```

Figure 7 - Variant Record Definition

```
procedure OleWordbasicProc.SetBStr(
  const s: String );
begin
  with fArg[fParams.cArgs] do
  begin
    bstrVal := SysAllocStringLen(
      @s[1], Length(s));
    VT := VT_BSTR;
  end;
  fParams.cArgs := fParams.cArgs+1;
end;

procedure OleWordbasicProc.SetInteger(
   i:Integer );
begin
  with fArg[fParams.cArgs] do
  begin
    iVal := i;
    VT := VT_I2;
  end;
  fParams.cArgs := fParams.cArgs+1;
end;

procedure OleWordbasicProc.SetBoolean(
  b:Boolean );
begin
  with fArg[fParams.cArgs] do
  begin
    vBool := b;
    VT := VT_BOOL;
  end;
  fParams.cArgs := fParams.cArgs+1;
end;
```

Figure 8 - Procedure wrappers for passing a parameter

we are passing (in this case one) by updating the field `fParams.cArgs`. The call to the appropriate `IDispatch` procedure - numbered 32820 - completes our code. I also clear the argument count at the end of the `CallProc` procedure:

```
fParams.cArgs := 0;
```

so that I can use `CallProc` without parameters by default.

Now, add a call to `WordBasic.Vline(10)` from a button `OnClick` handler to test the new code. Word now moves 10 lines down its document. Parameter passing is easy!

Behind the scenes, we had actually done a little preparation. In the `Create` constructor, we connected up the argument list to our parameter definition using the statement `fParams.rgvarg := @fArg`. This allows us to update the arguments without having to do extra housekeeping. It also by-passes the problems left by Borland's OLE definition, which passes a pointer to a single parameter definition in the `rgvarg` element. In the corresponding C definition, this causes no problem as you can pass a pointer to an array as easily as a pointer to a record. Pascal is a more type sensitive, but the `@` operator allows us to bypass it. However, keeping the assignment to a single line in the constructor allows us to fence off the dirty code and enjoy the benefits of that stronger typing.

Parameter wrappers

To simplify further the use of the parameters, one can place all the type specific code into a set of procedures. Figure 8 illustrates procedure wrappers for passing `STRING`, `INTEGER` and `BOOLEAN` parameters. In each case, I add the parameter and its type to the top of the stack and increment the stack pointer. We are using a fixed size stack - see `MaxArgs` in Figure 3 - and we do need to be careful we don't bust it. I initially used a 25 parameter limit but found many Word procedures actually have more parameters! The current limit of 50 appears to work well with Word, but if you are unfamiliar with your object then add range checking such as `if fParams.cArgs=MaxArgs then Raise Exception.Create( 'Ole: too many parameters' )` to the start of each of these wrapper procedures.

The string parameter is passed in a variant record pointing to a `BSTR` created using the `BStr` API discussed above. The `SysAllocStringLen` procedure takes an array of `char` and a length as its arguments, so for once we can get away without converting a Pascal string to a `PChar`. The role of `SysAllocStringLen is to` allocate a buffer of the required length and copy our string into it. We are not required to know the internal structure.

Other parameter types can be passed to an OLE procedure and forgotten, but woe betide the man who forgets his strings. I have therefore adapted the `CallProc` code to include string clean up code that generically frees any strings passed. In Figure 9 you can see the `for` loop stepping through checking for string parameters and calling `SysFreeString` for each one.

Parameters think backwards

Using the stack to save the parameters has the drawback that we must push them onto the stack in reverse order - the last parameter that you want to pass must be placed on the stack first. One of the friendlier OLE features is that you don't have to pass every defined parameter (but you do have to pass them in the order defined).

This allows you to prune the very large Word procedure definitions down to a manageable size, and create a whole host of procedures encapsulating useful functions. For example, I cut down Word's `FileOpen` procedure to meet my requirements. I took the 13 parameters that this procedure could use and removed the last 12 to give me the only one I generally use - the name of the file that I want to open. Because of the stack approach, nothing more is needed.

More, more, more

So much for my assault on Word procedures. But I have yet to cover how information is passed *out* of an automation object. Although OLE has, in theory, all the hooks to support parameters passed by reference (see the declarations at the end of Figure 7) I have yet to find any object that takes advantage of them. Certainly Word, Excel and MAPI don't appear to support them.

Fortunately, as well as procedures, OLE supports functions that return a single `Variant` variable back. The central support for functions is encapsulated in the `CallFunc` code included in Figure 9. You can then examine the `fVarResult` variable for the result. Don't forget that if a string is returned, it is your responsibility to copy the OLE `BStr` to a Pascal string and then free the string with a call to `SysFreeString` with a fragment of code such as:

```
Result := StrPas(fVarResult.bstrVal);
SysFreeString( fVarResult.bStrVal );
```

The OLE object will normally check whether you are calling a function or procedure and will flag an error if you have passed the wrong type. You can't just ignore the issue and always call it as a function - it will check that it has a `nil` pointer for the function return when it is expecting a call as a procedure.

I haven't covered the error structures being passed into the `Invoke` function in both `CallProc` and `CallFunc`. If you are getting

```
procedure WordBasic.CallProc(
  MemId: MemberId );
var
  i: Integer;
begin
  { Invoke as func with return val }
  fSucceeded := SucceededHr(
    pDispatch.Invoke(
      MemId, IID_Null, fLocale,
       DISPATCH_METHOD, fParams,
         fpNoVarResult^, fExcepInfo,
         fArgErr) );
  { clean up the strings }
  for i:= 0 to fParams.cArgs-1 do
  case fArg[i].VT of
    VT_BSTR: SysFreeString(
      fArg[i].bStrVal );
  end;
  { reset param count }
  fParams.cArgs := 0;
  fParams.cNamedArgs := 0;

  if not fSucceeded then
    Raise Exception.Create(
     'OLE function ' +
      IntToStr(MemId) + ' failed' );
end;

procedure WordBasic.CallFunc(
  MemId: MemberId );
var
  i: Integer;
begin
  { Invoke as a function with a
    return value }
  fSucceeded := SucceededHr(
    pDispatch.Invoke(
      MemId, IID_Null, fLocale,
       DISPATCH_METHOD, fParams,
        fVarResult, fExcepInfo,
          fArgErr) );
  { clean up the strings }
  for i:= 0 to fParams.cArgs-1 do
  case fArg[i].VT of
    VT_BSTR: SysFreeString(
      fArg[i].bStrVal );
  end;
  { reset parameter count
     ready for next time }
  fParams.cArgs := 0;
  fParams.cNamedArgs := 0;

  if not fSucceeded then
    Raise Exception.Create(
      'OLE function ' +
      IntToStr(MemId) + ' failed' );
end;
```

Figure 9 - CallProc & CallFunc with the String clean up code

exceptions raised because the object is returning an error, examine the strings in these structures to get excellent clues as to what is going wrong. Yet again, the system automatically creates these strings: it is your responsibility to free them with `SysFreeString`.

I also haven't covered *named parameters*. The big hint is that you add them to the parameter list, just like normal parameters, and add their `MemIds` to an array called `NamedArg` in our class. They must be added to our parameter list *before* the ordered parameters, but can be in any order. You must increment the `fParams.cNamedArgs` field to tell your object how many named fields you are passing. And, by the way, your 'named' argument is identified by a number - not a string. The number is the position of the argument in the parameter list, numbered from zero for the left most parameter.

IDispatch

Yes. I know. You thought you really didn't have to look at this. But it is your guide to the rest of the code that you might want to write. If the `IDispatch` interface is available on an object - and currently the object will support OLE Automation if and only if the interface is available - then you can depend upon the procedures highlighted in Figure 10 being available.

All OLE interfaces have the `QueryInterface`, `AddRef` and `Release` functions. The first can be used to locate any other interface - that is to say mini-API - available for this object. The other two are used to increment and decrement the number of copies of the interface that you are keeping in your program. When the object is created your requested interface counter is automatically incremented with a call to `AddRef`. But you must explicitly call the `Release` function when you have finished.

You may have been wondering where I obtained those magic numbers for my procedures. There are a number of ways of finding them out. A good search on CompuServe will give you clues; there are a number of tools supplied with the OLE SDK available on the *Microsoft Developer Network* CDs; or you can go straight to the object and ask it the question.

There are two whole interfaces devoted to telling you about what is available from your chosen object. The `ITypeInfo` interface gives details of what is available from this specific `IDispatch` interface. It will then point you onto the `ITypeLib` interface which gives a much wider range of information on the whole object. The `GetTypeInfoCount` function is only present to confuse your understanding of the `ITypeInfo` interface by suggesting that there may be more than one interface present. This should normally return one (assuming the feature is supported). Don't bother writing a load of code to step through the list. The `GetTypeInfo` function just needs a zero as the index; a `locale` that should be appropriate to the object and a variable to receive the interface.

Alternatively, you can use a reference book on the object and use a neat little function `GetIDsOfNames` that will directly translate the name to a `MemId`. This will sort out the function number but gives no hints on the parameters. Lest you rush off with a merry cry of 'I already have the documentation', note that, for instance, the *Word Developers Kit* states clearly that the parameters are not always defined in the order that OLE Automation requires them (thanks!).

The `ITypeInfo` interface is a better way to get your information and more flexible for the future. The `GetIDsOfNames` is designed for Visual Basic to run using late binding. However, if you can't find something in the type library that you think should be present, don't be afraid to experiment and see if `GetIDsOfNames` returns a suitable `MemId`. Sometimes this does work!

OLE is an exploration. A world to check out. Go one step at a time. Confirm each interface works as you expect. And then trust it like you would an API. Scatter your code with exceptions to be raised when something goes wrong. Build your code round `try .. except` blocks. Now go exploring with OLE. And enjoy. ■

Phil Wigglesworth is the Software Director at Estate Computer Systems the leading supplier of systems to the Property Industry with clients including Norwich Union, Grosvenor Estate, Wickes and Burfords. He has implemented many systems in Pascal and his own 4GL environment using the highly portable UCSD Pascal but has been working with Delphi since December 1994, leveraging its superb integration with Windows.

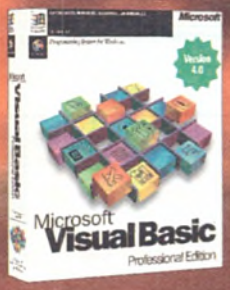
Inside OLE (Second Edition) by Kraig Brockschmidt. ISBN 1-55615-843-2 Microsoft Press.

```
IDispatch = class(IUnknown)
  function QueryInterface(const riid: IID; var ppvObj: Pointer): HResult;
        virtual; cdecl; export; abstract;
  function AddRef: Longint; virtual; cdecl; export; abstract;
  function Release: Longint; virtual; cdecl; export; abstract;
  function GetTypeInfoCount(var pctinfo: Integer): HResult;
        virtual; cdecl; export; abstract;
  function GetTypeInfo(itinfo: Integer; lcid: LCID;
            var pptinfo: ITypeInfo): HResult;
              virtual; cdecl; export; abstract;
  function GetIDsOfNames(const riid: IID; var rgszNames: PChar;
          cNames: Integer; lcid: LCID; var rgdispid: DISPID): HResult;
             virtual; cdecl; export; abstract;
  function Invoke(dispidMember: DISPID; const riid: IID;
         lcid: LCID; wFlags: Word; var pdispparams: DISPPARAMS;
           var pvarResult: VARIANT; var pexcepinfo: EXCEPINFO;
             var puArgErr: Integer): HResult;
                virtual; cdecl; export; abstract;
end;
```

Figure 10 - IDispatch

visual basic 4.0 and OLE controls for solutions

Visual Basic Standard Edition 4.0
Visual Development environment for Windows 95 that includes more than 20 prebuilt components. Build your own components in Visual Basic, which you distribute freely. *Every thing you need to build your own applications for Windows 95*
Visual Basic SE 4.0.....£79
Visual Basic SE 4.0 Upgrage.....£39

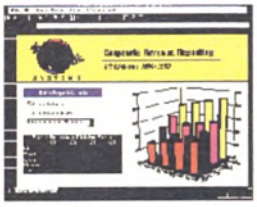


Visual Basic Professional 4.0
Load and re-compile 16-bit Visual Basic 3.0 applications. Create applications for 32-bit Windows 95 and Windows NT, and 16-bit Windows 3.x. Build OLE enabled applications quickly. Develop database applications fast using the new database engine.
Rapidly build applications for the entire Windows family
Visual Basic PE 4.0.....£359
Visual Basic PE 4.0 Upg from PE £119
Visual Basic PE 4.0 Upg from VB SE.....£229

Visual Basic Enterprise Edition 4.0
In addition to Professional Edition features: Create scaleable client-server applications for 32-bit Windows 95 and Windows NT, and 16-bit Windows 3.x. Develop fast client-server applications for Microsoft SQL Server and Oracle using the new remote data control
Build three-tier applications with remote OLE Automation support
Visual Basic EE 4.0.....£695
Visual Basic EE 4.0 Upg from VB PE.....£395

TO ORDER CALL 0171-833-1022
FAX 0171-837-6411
System Science, 1-6 Bradley's Close, White Lion St. London N1 9PN

SYSTEM SCIENCE



Formula One 3.0 £195
- NEW! Now available as a 16-bit and 32-bit OLE Custom Control.
- NEW! Read and write Microsoft Exel 5.0 worksheets.
- NEW! Support for multi-worksheet Excel-style workbooks including the tabbed worksheet interface.
- NEW! New worksheet functions including SUMIF, COUNTIF, CONCATENATE, ROUNDUP, and ROUNDDOWN

From Visual Components

Image Format Library
The AccuSoft OLE Custom Control provides the developer with a complete imaging toolkit that works with Visual Basic, Access and more. Supports 36 raster formats. Import, export, convert, compress, scan, display and print. Image processing includes: rotate, invert, zoom, pan, scroll, color reduction, thumbnails and more.

VBA Companion-£45 with Visual Basic
Simplify your OLE applications with VBA Companion. This powerful yet easy-to-use tool for viewing programmable objects accessible form OLE aware applications such as Excel, Access, and Visual Basic. This stand alone application allows the user to browse, search and print ALL the objects, their properties, methods and events within their entire system. It also facilitates the moving of an object's properties form one application into another.
By Apex Software

Calendar Widgets £99
Sheridan's reuable Components for Date & Time Management.
Calendar Widgets™ is perfect for visually displaying time and date related information.
- Visual Basic™ Custom Controls (.VBX) and 16- and 32-bit OLE Controls (.OCX)
- Four controls included for MonthView, YearView, DateCombo and DayView.
- Calendar Widgets is ideal for PIMs, data entry, billing, project management or any application requiring entry of time or dates

Prices are exclusive of VAT. Prices are subject to change. For upgrades, please add £8.00 shipping. Copyrights and trademarks are acknowledged. E & O.E.

OLE Custom Controls for developers

OCXs are all the rage, but their presence is still somewhat intangible. **EXE** found out what was available, from whom and for how much.

The VBX has had its day; the OCX is here to stay (for a while). With OLE paving Microsoft's path to Cairo and beyond, we'll be seeing more and more OLE controls liberally peppering our PC-based applications and operating systems. The transition to component-based software has not been painless, however. For the past couple of years, software vendors have been involved in a Red Queen-like race to keep abreast of Microsoft's evolving standards.

Basically, OLE Custom eXtensions (OCXs) are *in-process* 16-or32-bit OLE servers. They are similar in concept to dynamic-link libraries (DLLs) and, like VBXs, reside in the same address space as the calling program (avoiding the overhead associated with inter-process OLE Automation). The main advantage of OCXs over VBXs is that they comply with the open architecture of OLE's Component Object Model, and so can be used from any container application supporting version 2.0 of OLE. Add to that the fact that VBXs are not supported in 32-bit environments...

All products listed below should be available in both 16-bit and 32-bit form, except those marked * (16-bit only) or # (32-bit only).

OCX Product	*Publisher*	*Price*	*Suppliers*	*Availability*	*Description*
3D Graphics Tools	Micro System Options	£225	Q	1996	3D scenes, lights, cameras and objects
3D-Widgets	Sheridan	£85	Q	Dec 95	Form design tools, with enhanced menus, list/combo boxes and versatile panels
ALLText HT/Pro 4.0	Bennet-Tec	£335	G	Oct 95	Multi-font text box supporting OLE embedding and level 1 RTF
Aware/OCX	FarPoint	£110	G, Q	Oct 95	Set of data-aware controls. Memo control breaks 64 KB barrier
ButtonMaker	FarPoint	£70	G, Q	Current	Comprehensive button design plus Balloon control
Calendar Widgets 1.0	Sheridan	£87	G, P, Q, S	Oct 95	4 controls - MonthView, YearView, DataCombo and DayView
Chart FX 3.0	Software FX	£315	G	Current	Charting with Smart Detection, read-time charts, surface plots, rotation etc
Codebase et al	Sequiter	TBA	Q	1996	Access xBASE file and indices with xBASE-style functionality
Common Dialog Toolkit	Desaware	£35	Q	Current	Windows Common Dialogs; modify internal message streams. Embed VBXs into common dialogs
Communications Library 3.0	MicroHelp	£109	G, Q	Oct 95	Easy and powerful serial comms
Compression Plus 4.0	MicroHelp	£170	G, Q	Oct 95	PKZIP-compatible data compression
Controls for Btrieve OCX 3.0	Smithware	£185	G	Oct 95	Supports Btrieve
Crystal Reports Pro 4.5	Crystal	£285	G, P, S	Current	Report writer; reports can be generated and printed from within an application
Custom Control Factory 4.0	Desaware	£40	G, Q	Current	Animated buttons/toolbars, enhanced button style. 256/24-bit colour support, etc
Data Control for Btrieve	Classic Software	£169	P	Current	Supports Btrieve 6.x and 5.x, with control over indices, variable-length records and more
DataTable NT	Protogen	£299	P	Current	High-performance grid with caching, selectable cells, resizeable columns/rows, column totalling, etc
DataWidgets 2.0	Sheridan	£90	Q	Dec 95	DB controls including a Data Grid
Dazzle OCX	TeraTech	£420	G	Q4 95	Image control with zoom, panning, palette and grey scale support
dBarcode	dL Technology	£139	d	Nov 95	Creates barcode images; options to include text, select colours and generate checksums
Designer Widgets 2.0	Sheridan	£87	G, P, Q, S	Current	Create floating toolbars, use index tabs plus improved control over form appearance
Distinct TCP/IP SDK	Distinct	£265	G	Nov 95	TCP/IP kit including telnet, ftp, SMTP send, POP receive components
Drag-IT/OCX	Performix	£240	G	Dec 95	Lets users show data relationships by drawing with a custom toolbox
DynaZIP-16 3.0*	Inner Media	£175	G	Oct 95	Read/write v. 2.04G ZIP files from your application. Includes DZ_EASY interface
DynaZIP-32 3.0#	Inner Media	£210	G	Oct 95	32-bit version of DynaZIP-32
EnQuiry 2.0	Crescent	£299	Q	Oct 95	Client/server development tool that builds forms and queries without coding
First Impression 2.0	Visual Components	£99	G, P, Q, S, V	Current	Photo-realistic charting tool, featuring Chart Wizard, true perspective, 3D light sources, etc
Formula One 3.0	Visual Components	£99	G, P, Q, S, V	Current	Excel 5-compatible spreadsheet with tabbed worksheets, 3D cell references, autofill, etc
FxTools Standard/Pro	Image FX	£290	G, Q	1996	Display images, text and shapes using special effects
Gantt/OCX	AddSoft Inc	£189	G, Q	Nov 95	Visually interactive chart control with over 100 properties and 40 events
Graphics Server 4.02	Bits Per Second	£199	B, G, P, Q, S	Current	Presentational and 'hot' graphs and charts with 3D rotation, curve fitting and more
Grid/OCX	FarPoint	£70	G, Q	Oct 95	Data-aware grid control
HighEdit 3.5	heilerSoftware	£195	B, Q	Oct 95	Configurable word processor that supports RTF, print preview, data fields and more
Image Format Library 16*	Accusoft	£415	G	Current	Imaging with 36 raster forms. Import, export, convert, compress, scan, display and print
Image Format Library Pro Gold#	Accusoft	£1470	G	Current	2-6 times faster than 16-bit version
ImageKnife 1.0#	Media Architects	£375	G, Q	Feb 96	Image handling up to 24-bit colour with Twain, JPEG, FIF support and more

OCX Product	Publisher	Price	Suppliers	Availability	Description
Integra	Integra Bus. Solutions	£625	Q	1996	ODBC client/server tool that builds forms/queries without coding, plus programming interface
Intelligent Paper	London Software	£125	Q	Jan 96	Scrollable container that manages child controls, with alignment, word-wrap and more
Leadtools	Lead Technologies	£365	E, G, Q	Oct 95	JPEG toolkit, with dithering, display optimisation, unlimited image size, and more
LeadTools (DLL Add-on)	Lead Technologies	£145	Q	Oct 95	Add-on for DLL version of LeadTools
MediaKnife 1.0#	Media Architects	£299	G, Q	Nov 95	Create multimedia applications with special effects, fads, sprites, hotspots and more
MetaDraw	Bennet-Tec	£240	G	Oct 95	Image control to view/create/manipulate metafiles. Hotspots and programmatic control
MicroHelp Muscle/Win	Microhelp	TBA	G	Q1 96	Pop-up routines, dialogs, sundry controls, system routines, Windows services and much more
MyData Control 1.0	Apex	£115	G	Oct 95	Data control for VB to allow you to use any bound control with any data source, including text files
NetPak 2.0	Crescent	£145	G, Q	Current	Access to a broad array of network services
OLE Tools 5.0	Microhelp	£140	G, P, Q	Oct 95	Set of OLE custom controls
PDQComm 3.0	Crescent	£145	G, Q	Current	Easy and powerful serial comms. Multithreaded
PICS NT	Protogen	£299	P	Current	Date/calendar, time, numeric input using your choice of odometer, and more
PowerPak Pro 2.0	Crescent	£1,075	Q	Nov 95	More than 40 custom controls and 400 functions for advanced development
ProMath OCX	TeraTech	£130	G	Q4 95	Trigonometry, integration, complex numbers, Bessel functions, stats, non-linear equations etc
QuickPak Pro 4.0	Crescent	£180	G, Q	Current	30 custom controls (eg calendar), plus 100s of DLLs and sample programs
QuickPak Scientific 3.0	Crescent	£145	G, Q	Current	Non-linear equations, partial differential equations, integration/differentiation, etc
Resource Manager/OCX	AddSoft Inc	£195	G	Q4 95	Integrated with Schedule/OCX to monitor project resources forwards and backwards in time
Schedule/OCX	AddSoft Inc	£189	G, Q	Oct 95	Resource scheduling with year, month, week and day at a glance
sp-Assist	Sheridan	£450	Q	Current	Multi-user SQL Server development tool
Spread/OCX	FarPoint	£165	G, Q	Oct 95	Spreadsheet control
SpyWorks 4.0	Desaware	£115	G, Q	Current	Subclassing. Access Windows API. Intercept Windows messaging. Debugging tools
Storage Works	Desaware	£100	G, Q	Current	Access to any OLE 2 Structured Storage Technology-based file. Simplified access to Registry
Tab/OCX	FarPoint	£45	Q	Oct 95	Flexible Tab interface
TabPro/OCX	FarPoint	£70	G, Q	Oct 95	Flexible Tab interface with organiser look
Tlist 3/Pro	Bennet-Tec	£195	G	Current	Enhanced outline control - supports any no. of images, variable-sized items, picture-oriented view etc
TrueDBGrid	Apex	£159	Q	Nov 95	Database grid from the makers of TrueGrid and VB 4.0's Data Grid control
TrueGrid 4.0	Apex	£155	G, S	Q4 95	Data-bound grid, with split grid, outlining, multiple select, individual cell colours and fonts
VB/Link OCX	Brainstorm	TBA	G	Q1 96	Access Lotus Notes
VBAssist 4.0	Sheridan	£135	G, Q	Current	Application design productivity enhancement tools plus form generation
VBlite OCX	TeraTech	£130	G	Q4 95	150 routines and properties for printing, comms and arrays in Visual Basic
VersionStamper	Desaware	£100	G, Q	Current	Embed into your EXE info on all DLLs/custom controls used, with version and date info
VideoPlay 1.0#	Media Architects	£59	G, Q	Nov 95	Video playback using Video for Windows, Quicktime or MPEG
Visual Speller 2.0	Visual Components	£99	G, P, Q, S, V	Current	100,000-word spell checker (American English) with custom dictionary builder
Visual Voice for Windows 95#	Stylus Innovation	£495+	e	Oct 95	Telephony toolkit supporting fax-on-demand, Interactive Voice Response and voice mail
Visual Voice for Windows NT#	Stylus Innovation	£695+	e	Oct 95	Telephony toolkit supporting fax-on-demand, Interactive Voice Response and voice mail
Visual Writer Pro 3.0	Visual Components	£129	V	Current	RTF-compatible text editor, with dialogs for paragraph/character formatting and search/replace
VSFlex	VideoSoft	£89	G, Q	Current	Create Ad-hoc reports by visually pivoting grid columns. Pattern-matching text search/replace
VS-OCX	VideoSoft	£40	G, Q	Current	Elastic controls for resolution-independent forms. Includes IndexTab and Awk controls
VSView	VideoSoft	£89	G, Q	Current	Replacement printer object providing headers/footers, preview, page numbers etc
XRef 2.0	Crescent	£95	G, Q	Current	Efficient management of complex application source

Thanks to all the companies that took the time to send us product information.
Apologies to any companies whom we failed to contact or where unable to reply before we went to press.

Supplier	Telephone Number	Key	Supplier	Telephone Number	Key
Bits Per Second	01273 727119	B	Grey Matter	01364 654100	G
Contemporary Software	01273 483979	C	Programmer's Paradise	0161 728 4177	P
dL Technology	0181 559 0049	d	QBS Software	0181 994 4842	Q
Equations Technology	01935 74144	E	System Science	0171 833 1022	S
eurovoice	01473 730000	e	Visual Components	01892 834343	V

A supplement to EXE Magazine, October 1995
Visual PROGRAMMING
EXE

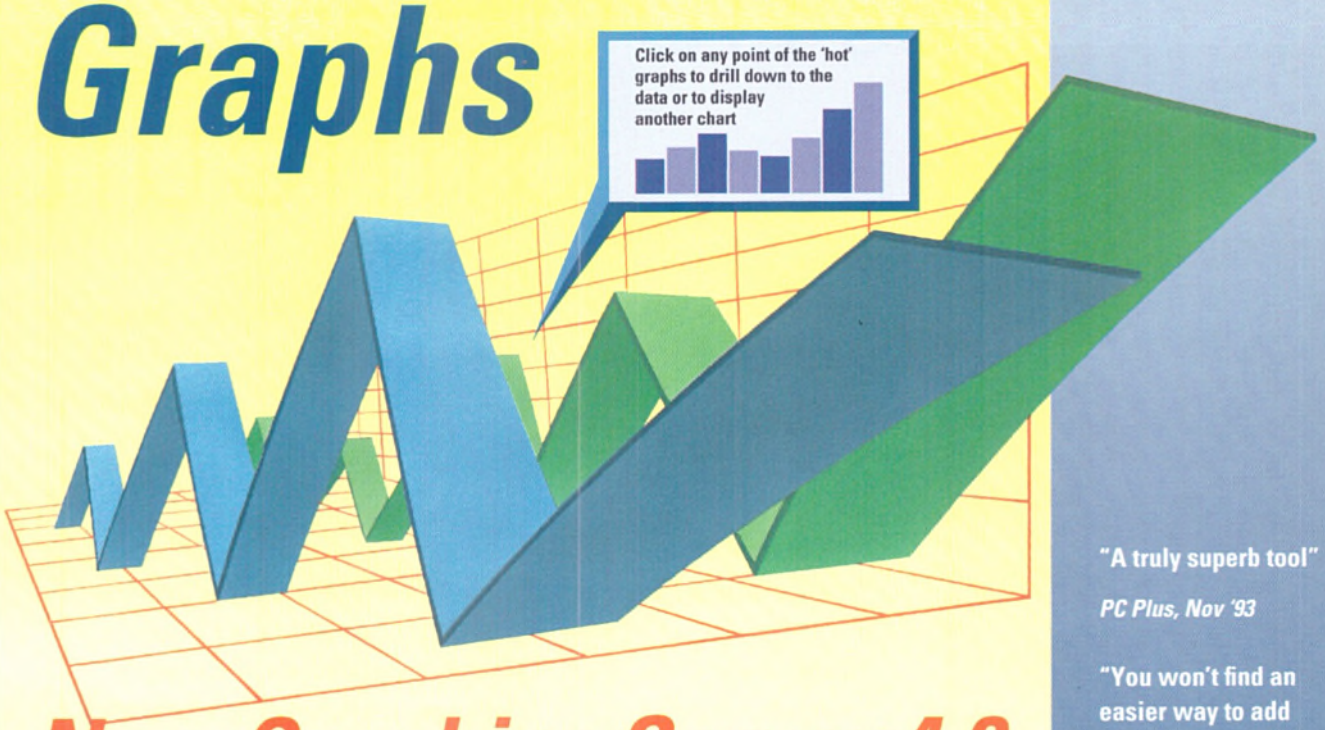
Editor: David Mery **Staff Writer:** Roland Perera **Advertising:** Steve Miles
Production: Mark English, Kate Adams **Research:** Jacqui Ramrayka

Centaur Communications Ltd, 50 Poland Street, London W1V 4AX UK.
Tel: 0171 287 5000 Fax: 0171 437 1350 e-mail: editorial@dotexe.demon.co.uk. / stevenm@dotexe.demon.co.uk

Language

Visual C++
C/C++
SQLWindows
Visual Basic
Turbo Pascal
Powerbuilder
FoxPro
Windows4GL
Access

Programmable Graphs



New Graphics Server 4.0

Thousands of developers use the original GRAPH.VBX, licensed from us by Microsoft back in 1992.

Thousands more took advantage of the enhanced graphs and charts in the 1993 release of Graphics Server.

Now Graphics Server 4.0 is here.

NEW ***Interactive Toolbar***

By accessing the 'property pages', this new feature of the VBX can be used by you at design-time and by your users at run-time to change the properties of the graph or chart - and you can configure the toolbar.

You can also control every aspect of a graph or chart by accessing the 180+ low-level functions in the DLL.

NEW ***Rotatable True3D***

Now your users can project graphs in True3D with control over rotation, elevation and eye position.

NEW ***Real-time graphing***

One or more data sets can be displayed in a scrolling window and updated several times a second.

NEW ***More Graphs and Stats***

Contour plots with colour-grading, combination graphs, open-high-low-close, box-whisker and more.

AND STILL

All the interfaces you need (VBX, DLL, FLL, C++ graph class etc), 16 and 32 bit versions for Windows NT and '95 - there's even an OLE Custom Control (OCX). And unlike the GRAPH.VBX in VB Pro, Graphics Server's VBX gives you 'hot' graphs, full axis control, printing to the DC and null values for missing data, plus all the new features like True3D and the Toolbar.

STAY OUT IN FRONT !

Call us now, and find out how Graphics Server 4.0 can keep you and your users out in front.
(And remember, you can distribute the run-times royalty-free).

"A truly superb tool"

PC Plus, Nov '93

"You won't find an easier way to add charting to your Windows applications"

PC Magazine, Sept '93

BITS PER SECOND

14 Regent Hill
Brighton
England
BN1 3ED

Telephone
01273 727119

Fax
01273 731925

now fast and friendly tackles huge and difficult

Enterprise-wide 3-tiered applications

Network partitioned solutions

High performance remote data access

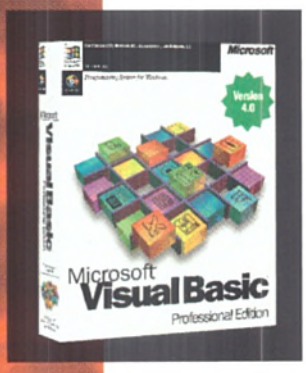
Large scale team development

32-bit with built-in 16-bit support

Open, reusable component creation

Enhanced IDE

New data-bound contrlos



Introducing VB 4.0... looks reassuring like version 3.0, but now enables teams to develop multi-tier, scaleable, 16- and 32-bit client/server solutions

Visual Basic Professional Edition
Load and recompile 16-bit Visual Basic 3.0 applications to re-use design effort for 32-bit Windows 95 and Windows NT. Build OLE enabled applications with support for OLE controls and OLE Automation, and create custom OLE components for enhanced reusability. Use the new Jet database engine and Data Access Objects (DAO) to rapidly develop database applications.

Visual Basic Enterprise Edition
Create scaleable client-server applications for Windows 95, Windows NT and Windows 3.1, and use the new remote data control for Microsoft SQL Server and Oracle 7.0. Utilise remote OLE automation for three-tier application development. Includes Microsoft SourceSafe for safe configuration management.

TO ORDER CALL:

SYSTEM SCIENCE
0171-833-1022
System Science, 1-6 Bradley's Close, White Lion St. London N1 9PN
Fax: 0171-837-6411

QBS SOFTWARE
0181-994-4842
QBS Software Ltd, 10 Barley Mow Passage, London W4 4PH
Fax: 0181-994-3441