Bugging out

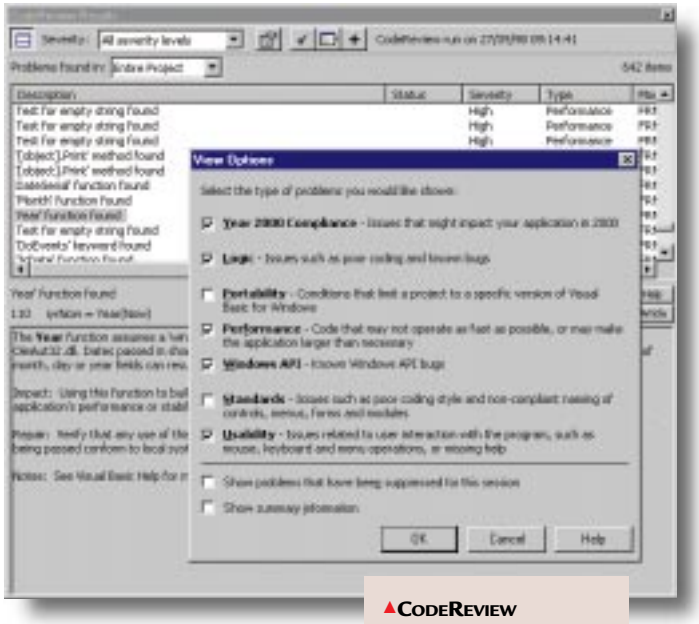
It's easy to catch program bugs and streamline your code in Visual Basic, says Tim Anderson.

Congratulations to NuMega and Compuware, who have shipped DevPartner Studio for Visual Studio 6.0 almost before Microsoft's product hit the shelves. Visual Studio itself has an array of debugging facilities, but DevPartner Studio goes several steps further. A number of development tools are supported, but the main focus is on Visual C++ and Visual Basic.

Smart move

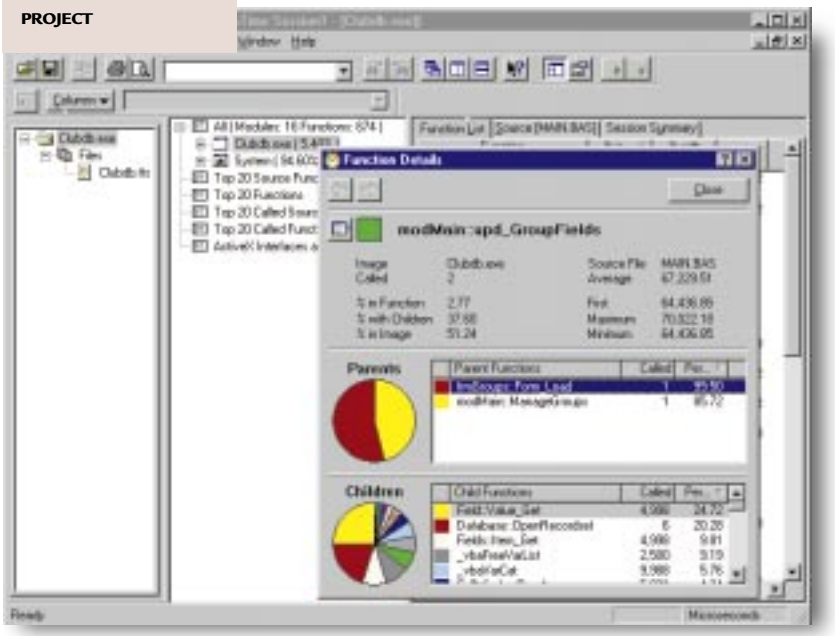
I tried out the VB tools on a project I knew to be of poor quality. A club membership database, it was ported from VB 3.0 to VB 6.0 with minimal changes. First, I tried out SmartCheck. If you run an application with SmartCheck enabled, then it will catch unhandled errors before VB's own error dialogue appears. The SmartCheck error dialogue is more informative than VB's default, particularly with ActiveX and API problems. When you exit the application, you can view a report of all your application's events, with a synchronised view of the source code, making it easier to trace why the error occurred. SmartCheck will also test compliance with different versions of Windows.

Alongside SmartCheck, DevPartner studio includes FailSafe. This automatically adds error handlers throughout a project, the aim being to make the code uncrashable. Whereas an unhandled error in VB will normally terminate the application, Failsafe will pop-up a printable dialogue explaining the error and inviting you to describe what you did to cause it. The problem with FailSafe is that it only catches code that raises an error, whereas many bugs are logic errors that deliver wrong results without actually crashing the application. If you are suffering from crashes though, FailSafe may well help.



▲ **CodeReview ruthlessly examines your code for potential problems**

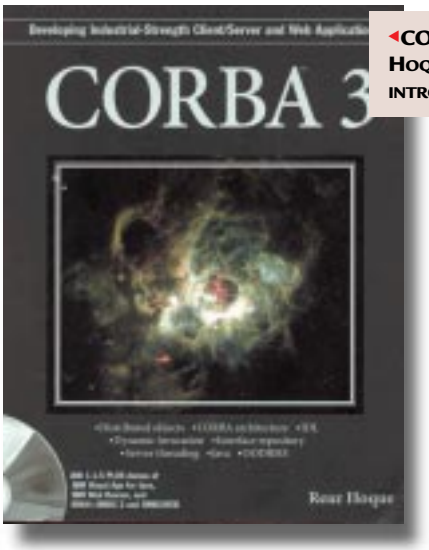
▼ **Analysing a time-consuming function in a Visual Basic project**



Next, I tried Code Review, which analyses your code and compares it against a database of rules. Code Review made 1,500 comments on my modest application. For example, it picked up use of the Global keyword, pointing out that Public should be used instead as Global may not be supported in future versions.

A few of the comments were more illuminating. Where there was a test for an empty string, it noted that testing for Len(strSomeString) <> 0 could be 150% faster. Where the Print method was used, it noted that the API call TextOut was often 450% faster. It found variables declared but not used, controls with no help assigned, objects created but not explicitly freed, and potential Year 2000 problems. The project quality would be significantly improved by addressing all these points.

Then I tried TrueTime. Like FailSafe, this involves building an instrumented version of your application. Run the application, and TrueTime analyses the performance. In a nutshell, TrueTime tells you exactly how long the application



CORBA 3 BY REAZ HOQUE IS A GOOD INTRODUCTION

spent in each function.
Most functions call other functions, so TrueTime lets you drill down to see what really took the time. A TrueTime session report contains a huge amount of information, and you can view results at API level as well as in terms of your Visual Basic code. I found that only 5.4% of the execution time was running VB code; the rest was in system calls. I was also able to see which functions took up most execution time, how many times they were called, and how the time broke down between child functions.
TrueTime is brilliant, and as a bonus will show you how VB works.

Learning Delphi 4

Delphi is a superb product, but very poorly documented. It is as if Inprise likes to tease developers by packing in some great features but not letting on how to use them. For example, try looking up TControlBar in the new Developer's Guide. It isn't mentioned, as far as I can see. There is an online help entry, in the usual terse style without any examples, and that's all. A TControlBar is what you would use to dock several floating toolbars on one site, in the style of Delphi's own interface. If you try to do serious work with dockable controls, you will find that it is not altogether a trivial process and good documentation would be a great help.

Another problem area is CORBA, which is new in Delphi 4 and a key feature. Try to use it, though, and you will find the documentation hopelessly inadequate. To make matters worse, there aren't many good Delphi books around. Recently published is the *Delphi 4 Developer's Guide* by Steve Teixeira and Xavier Pacheco. This is an unusual book, in that some of the chapters are not printed but are provided on the supplied CD. You can easily be caught out, because the missing chapters are listed in the table of contents but all you find printed is a one-page summary. This is strange, but you do still get over 1,100 pages of intermediate-to-advanced Delphi information. There is a lot of useful material here, but the authors do seem to be infected with the Delphi 4 malaise in that several important topics are just not covered at all. If you want to learn about Action lists, docking controls or CORBA applications, you will have to look elsewhere. The book is still well worth having, with valuable material on topics including the Windows API, COM and ActiveX programming, and component programming. But this is not a quick route to learning Delphi 4's new features.

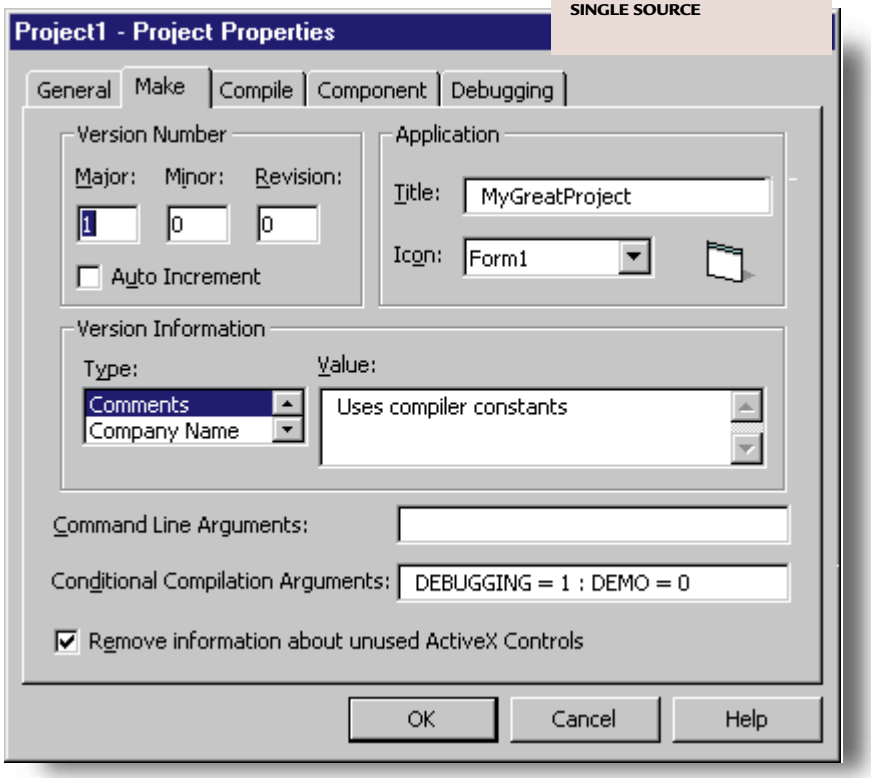
Delphi is a superb product, but very poorly documented

For the time being, Delphi developers who want to work with CORBA will have to turn to more general titles. One recent publication is *CORBA 3*, a developer's guide by Reaz Hoque. Although it is aimed at C++ and Java developers, this title provides an excellent overview of what CORBA is and how it works. Beginning with an architectural overview, it goes on to describe a simple CORBA application in detail. Topics in other chapters include CORBA services, Interface Definition Language, and Object Oriented databases. With a good book like this, plus what little documentation there is supplied with Delphi, you have some chance of making sense of Delphi's CORBA features.

Beginner's corner

One of the more mysterious topics for newcomers to programming is the compiler directive. Most languages have this feature, including Visual Basic since version 4.0. The relevant VB

USE COMPILER CONSTANTS IN VB TO CREATE DIFFERENT EXECUTABLES FROM A SINGLE SOURCE



& answers

Q I have Microsoft Visual Basic 4.0 Pro, and I would like to know how I can create my own metafile from scratch (in code) and save it as a standard .wmf file.

STEPHEN HOLLY

***A** Windows has several API functions for dealing with metafiles. In principle, a metafile is a set of Windows drawing commands saved to disk. The function CreateMetaFile creates a WMF compatible with Windows 3.1, while CreateEnhMetaFile creates an improved metafile that only works on Windows 95 or NT. You can use both from VB, but for success you will need a detailed knowledge of how Windows draws graphics. Daniel Appleman's classic book, Visual Basic Programmer's Guide to the Windows API, includes an example drawing application which loads and saves metafiles, and this would be a great place to start.*

Q I am trying to write a script that runs under NT Server to create a new user account. I would need to call an API, but can find no documentation on Windows APIs pertaining to NT and account administration.

MARK WILKES

***A** The NetWorking API is the one you want, and it is documented in the Platform SDK. You can find it on the MSDN (Microsoft Developer Network) CDs that come with most development products or by subscription. Unfortunately, while the Windows Scripting Host WshNetwork object deals very well with things like mapping network drives or printers, adding a user account is beyond its scope. It will be easier when the Active Directory is available, in Windows NT 5.0, since this has an automation interface that includes methods for creating new users.*

Q I have often had to use a DOS batch file to automate some processes in Windows 95, but there does not seem to be any obvious way of launching Windows or DOS applications from within VB Script. In VBA version 3 and up, you can use the Shell command, but it is not available in VB Script. Am I missing something here?

JOHN LONGBOTTOM

***A** Several readers have written in with similar queries. You need to use COM automation. This code runs the CD Player:*

```
Set WshShell
=Wscript.CreateObject
("Wscript.Shell")
WshShell.Run
("CDPLAYER.EXE")
```

The scripting host exposes a number of objects, providing most of the functionality of the old batch files, for manipulating files and connecting network drives. It is a new approach if you are used to batch files, but useful since many Windows applications are programmable through COM automation. You can create your own automation servers with the full version of VB, and program the scripting host's objects from VB apps.

statement is #IF ... #ELSE ... #END IF. In Delphi it is {$IFDEF}, and in C++, #ifdef. It works in a similar way to the standard IF ... END IF, but with important differences. The conditional expression has to be based on a specially declared compiler constant, and the part that is not executed is not even compiled. In other words, the #IF statement is not represented in the final executable at all.

There are two ways to use compiler directives in Visual Basic. The main technique is to use the box called Conditional Compiler Arguments in the Project Properties dialog. Constants you define here are global. The other method is to put a #Const declaration in your code. These declarations are private to the module they are in — you can override a global directive just for that particular module. Normally, the only values you need give a compiler constant is 0 or 1. VB will not accept non-integer values for global compiler constants, and any non-zero value counts as true.

So what is the point of them? One common use is for debugging. Typically, you want additional checks and tests during development, which you strip out to make code more efficient for deployment. With a compiler directive, all you need do is to place your debugging code within a #IF DEBUGGING block. (In most languages, you would use DEBUG, but in VB that conflicts with the Debug object.) When you come to deploy, all you need do is to remove the #Const DEBUGGING declaration, or set it to 0, and do your final build.

Here are some other handy uses for compiler directives:

- Use a DEMO constant to disable key features so you can put a demo version of your application on the internet.
- Use a BETA constant to add beta-specific code like a time-out after a certain date.
- Use the built-in WIN16, WIN32 and MAC constants to create platform-specific code.
- Perhaps you are considering switching from one grid control to another. You want to run with the new grid, but preserve the code that works with the old one. Define a NEWGRID constant to do just that.

Despite their advantages, some developers refuse to use compiler directives. The truth is, they are inherently dangerous. Effectively, you are testing a different version of the software than the one that gets released. The problem is even worse if you use multiple directives in the same project. Use or refuse? The decision is yours, but personally I would not want to be without them. Just take care now.

PCW CONTACTS

Tim Anderson *welcomes your queries and tips. He can be contacted c/o the PCW editorial office (address, p10) or email* visual@pcw.co.uk.

DevPartner Studio 6.0 costs £838.00 (£984.65 inc VAT) from GreyMatter (01364 654100), or individual apps such as TrueTime or CodeReview can be purchased separately for less. See www.greymatter.co.uk *and/or* www.numega.com.

Delphi 4 Developer's Guide by Steve Teixeira and Xavier Pacheco costs £54.95 inc VAT (Book and CD). ISBN 0-672-312840.

CORBA 3 by Reaz Hoque costs £42.99 inc VAT (Book and CD). ISBN 0-7645-3200-6.