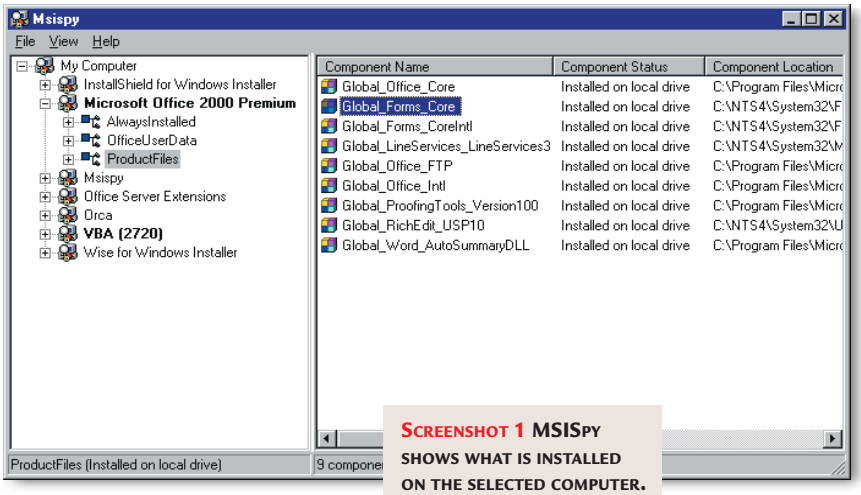
MSISpy knew too much

Uninstall routines are messy at the best of times, but this needn't be the case, says Tim Anderson.

Many Windows problems begin with installation. Software reviewers are continually installing products and solving mysterious setup problems is a huge time-waster. Sometimes the only solution is to shrug and try another machine, or even a clean Windows installation. Part of the problem is that uninstall routines do not always work, and even when they do, they rarely leave the system in exactly the state it was before. Ironically, this last issue is doubly true of Microsoft software.

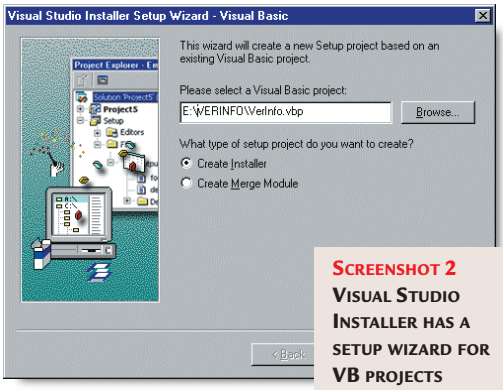
Now imagine how it ought to work. What if Windows kept a database of exactly what software is installed, not only at the application level but right down to individual components and files? You would be able to see at a glance what was installed on any system, without hunting through the Start menu or Explorer folders. Equally, uninstall should work properly since the system knows why each file was installed and for which application.



Screenshot 1 MSISpy shows what is installed on the selected computer. When you select a component in the right pane, the products that use that component appear in bold on the left

To illustrate the point, screenshot 1 shows a utility called MSISpy. MSI stands for Microsoft Installer. It shows all the installed products and lets you drill down into their components and files. A right-click menu gives options to Configure, Reinstall or Uninstall. MSISpy uses the runtime Installer API, making the point that in the new scheme of things installation is not a one-shot process, but combines the functions of installation, maintenance and repair.

Unfortunately, MSISpy can only analyse products that have been installed with the Windows Installer, which is why only a few applications are listed. Many developers and software companies have invested considerable effort in installation routines, and are reluctant to throw them away, as they will have to in order to adopt the new system.



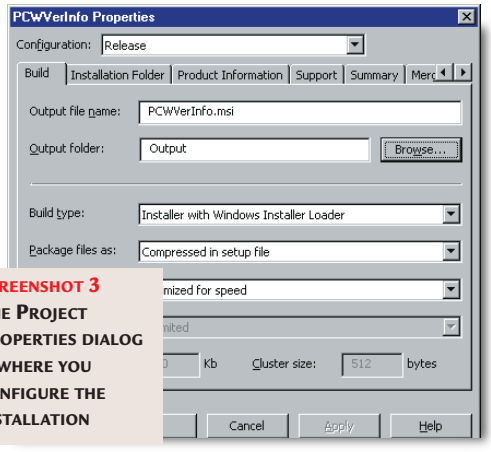
Screenshot 2 Visual Studio Installer has a setup wizard for VB projects

Microsoft is trying to encourage its use by making it a requirement for Windows certification, the replacement for the old logo scheme. Whether or not it becomes standard remains to be seen.

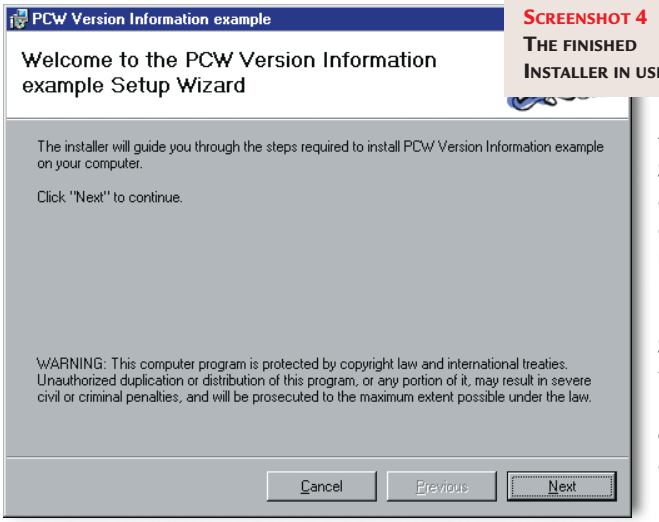
If you want to have a go at this, there's an easy way you can try out the Windows Installer. If you have any Visual Studio product, you can download the Visual Studio Installer for free. Here is how you could make an installation for a simple Visual Basic application, using the version information example from January's column. This is a little utility for viewing the version information in an executable file.

The Windows Installer

Users of Office 2000 will have noticed the appearance of a new gadget called the Windows Installer. This is a setup service that is built into Windows 2000 and which can be added to Windows 95, 98, and NT4 systems. At first glance it is just another installation utility, providing an alternative to third-party systems like InstallShield and Wise. This is a serious misunderstanding. The Windows Installer is a new approach to application management that provides exactly the centralised management described above, plus numerous other features.



Screenshot 3 The Project Properties dialog is where you configure the installation



SCREENSHOT 4 THE FINISHED INSTALLER IN USE

1 Install and run the Visual Studio Installer, which appears in the Visual Studio group in the Start menu. This takes you into Microsoft Development Environment, as used by Visual InterDev and Visual J++. Choose Visual Studio Installer Projects and then Visual Basic Installer from the New Project dialog.

2 The wizard asks you to select a Visual Basic project (screenshot 2). Find the project file, check Create Installer, and then click Next.

3 The wizard has two translation options. You can either create an installer for any locale, or localised version of Windows, or else restrict it to the current locale, in which case you can use the extended character set. The former is the more attractive option in most cases, so check this one and click Finish.

4 The wizard now creates the installer project. You can now choose Build to create the installer file. Before you do, though, you probably want to change a couple of settings. The dialog for this is hidden away. Select the name of your installation in the Project Explorer, then right-click and choose Properties (screenshot 3). Under Build type, select Installer with Windows Installer Loader. This means that the Windows Installer service itself will be installed, if it is not already present, and is essential if you want to target versions of Windows other than Windows 2000. You can also amend the target location, product name, support details (who to phone for help), and other information.

Another tip is that by default the wizard places the application shortcut at the very top of the Start menu. To amend this, double-click the File system element in the Project Explorer. In the File System windows, find the entry for User's Start Menu, create subfolders as required and move the shortcut into the right location. You probably want to rename the shortcut to a friendly name.

5 Now choose Build and find the result, which will be in the Output sub-folder of the Installer project folder. You can test the install by running SETUP.EXE in the normal way (screenshot 4).

The resulting installation, which only took a few minutes, has some interesting features. For example, if you go to the Add/Remove Programs utility in Control Panel, you will find your application listed. Run it, and the Installer opens with options to Repair or Remove. If you created an installation with several optional components, these would be listed with options to add or remove them. Should the user uninstall the application, the Installer will be intelligent about removing the application files, but not the Visual Basic runtime files. If you now run MSISpy, which you can download along with the Visual Studio Installer, you will find the new application listed.

Clever but complex

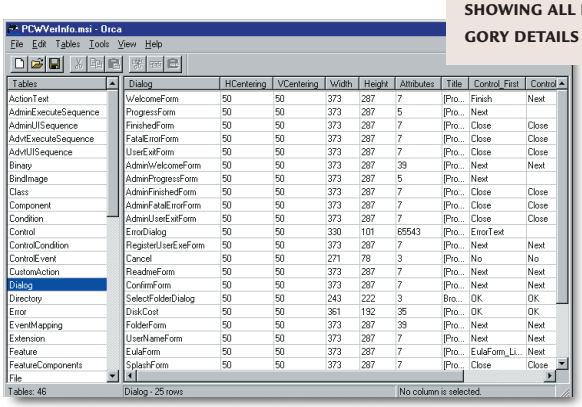
The Visual Studio Installer is simple to use, once you get your head around the way it works, which is very different to the script-driven solutions used by previous setup utilities. The fundamental point is that the whole process is data-driven. When you edit an Installer project, you are editing a database.

To prove it, download the utility mysteriously called Orca from Microsoft's site, and open the .MSI file you have just created if you followed the earlier example (screenshot 5). It is no more or less than a collection of database tables.

There are several advantages. For example, the Installer can exploit database features such as transactions, so that failed installations can be rolled back successfully and uninstall has a chance of working properly. The Installer database remains on the target machine, enabling utilities such as MSISpy to work.

Merge modules are designed to be merged into other installations

A great feature of the Installer is called merge modules. These are themselves installation packages, but are designed to be merged into other installations. If you look closely at a Visual Basic installation in Visual Studio Installer, you will see several merge modules listed, with .MSM extensions. MSVBVM60.MSM is the module for the core VB runtime files. If all VB applications use this module, the VB runtime can be managed intelligently by the system.



SCREENSHOT 5 AN INSTALLER DATABASE SHOWING ALL ITS GORY DETAILS

All installations and merge modules have a product and an upgrade code. Both of these identify the product, but the difference is that while the product code should change with each significant new version, the upgrade code does not. When you release a new version of your

application, the Installer looks for a previous version with the same upgrade code. If it is present, the Installer offers to remove the old version before installing the new one. Note that there is an option to mark files and registry entries as not to be deleted on uninstall. While this undermines the idea of complete uninstallation, it is essential if you want to preserve user settings on upgrade.

If you dig deep into the Windows Installer, you will find it well thought-out but complex. Nobody wants to contemplate building installer databases by hand, and the Visual Studio freebie has limitations, such as the inability to edit dialogs in the user interface. Another issue is that while the built-in features of the Installer are excellent, there are times when you need a script, to incorporate special logic or processing. It is possible to incorporate scripts, using the Windows Scripting Host, or to call external functions, but not with Visual Studio Installer. The easy solution is to get hold of the new Windows Installer tools from Wise or Installshield, that offer more sophisticated editing facilities. You can also investigate advanced Installer features such as 'install on demand' and patching, both of which are built into the system.

Why should you use the Installer?

Microsoft has made surprisingly little noise about the Installer, yet it has the potential to make life easier for Windows users. It is not just a matter of install and uninstall, but it makes the whole system more resilient and reliable. Developers should make the effort to use it.

What's next in Visual Studio

The next version of Visual Studio is not likely to arrive until the second half of 2000, but Microsoft is already talking about some of the new features. I recently spoke to Dave Mendlen, product planner for Visual Basic and Visual InterDev, who presented two Visual Studio 7 technologies, web forms and web services.

Visual Studio 6.0 is already replete with web features. While the underlying Active Server Page (ASP) technology is impressive, there are some strange aspects to how it is used in Visual Studio 6.0. Visual InterDev 6.0 is the primary web development tool, but it tends to encourage authoring ASP scripts, VBScript or Javascript running on the server but stored in ASP pages. At the same time, Microsoft's declared strategy involves minimising ASP script and placing application logic and processing into COM components written in Visual C++ or Visual Basic and again running on the server.

The new Visual Studio features web forms, that use ASP in the right way

This anomaly is being addressed. The new Visual Studio features web forms, that use ASP in the right way. The scenario is that you can drag a button or other control to a web form, set its properties, and double-click to open the code editor for the events it fires. The code in this case is not VBScript, but Visual Basic or Visual C++, since you are actually editing a COM component. From the browser's perspective, the end result will be standard HTML 3.2 pages for good compatibility.

Web services are another thing entirely. The idea is to get the ease of use of COM components but with messages passing over the Internet as XML, rather than through Windows networks. The key to this is the ability to build components in the Visual Studio languages that have the ability to define their interface in XML and also to send and receive XML packets. With this done, it becomes possible for web applications running anywhere else on the web to have programmatic access to the component.

The underlying technology is not exclusive to COM, and this would also be a good way to integrate with Java or Corba-based systems. It does mean that Windows developers who are building COM components will have an easy route to creating real distributed applications, where 'distributed' means components running anywhere on the web.

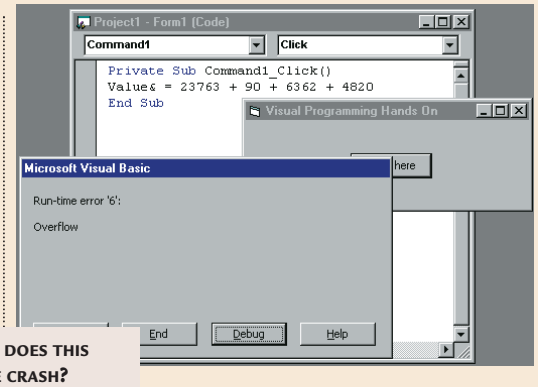
Code for crashing

Nelson Jerram is having a pretty serious problem with a piece of code he has written.

He emailed to ask: 'If I have the following code in VB:

```
value&=23763+90+↙
6362+4820
```

I get runtime error '6', even though the value variable is long. Why?'



WHY DOES THIS CODE CRASH? ANOTHER VB MYSTERY SOLVED.

This is because VB uses an implicit variable to sum the values on the right-hand side. Because the first value is small enough to fit in an integer, VB uses an integer type, which then duly overflows. The solution is to use the function Clng to force the compiler to use the a long variable instead. In other words:

```
value& = ↙
Clng(23763)↙
+90+6362+4820
```

Not very obvious, but that is the fun of programming!

(*Key: ↙ code string continues*)

PCW CONTACTS

Tim Anderson welcomes your Visual Programming comments and queries. Contact him at visual@pcw.co.uk or via the PCW editorial office.

You can download the Visual Studio Installer from http://msdn.microsoft.com/vstudio/downloads/vsi/default.asp

Download the Installer SDK from http://msdn.microsoft.com/developer/sdk/wininst.asp

For further information, see the newsgroup microsoft.public.platformsdk.msi

Wise is at www.wisesolutions.com, and Installshield at www.installshield.com