Clean-up campaign

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

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

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

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

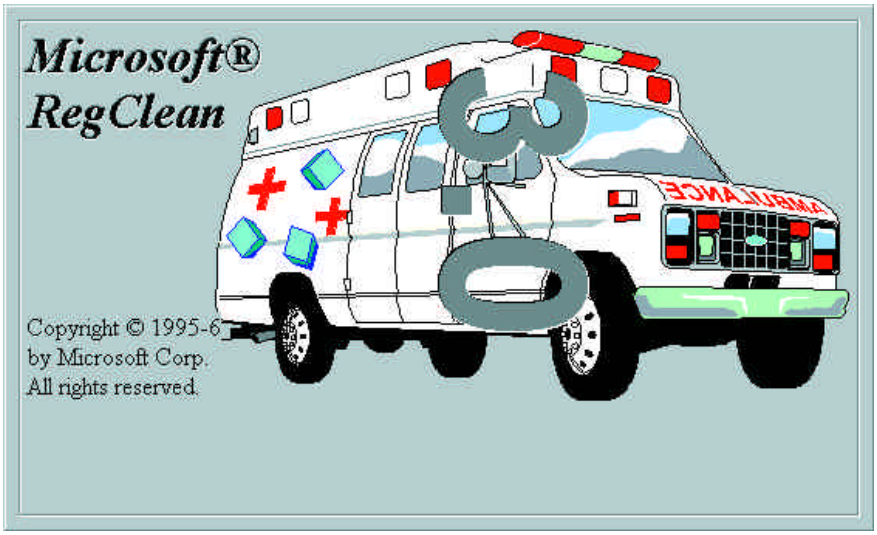
1. Using a registry editor, delete the HKEY_CLASSES_ROOT\LICENSES key.
2. Run Regclean.exe and delete all *.OCX and *.OCA files.
3. Delete OLEPRO32.DLL.
4. Restart Windows and reinstall Visual Basic.

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

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

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

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



Microsoft's RegClean 3.0: proceed at your own risk

p308 ➢

dedicated to that purpose. Do not install it on a system used for real work.

4. Persuade your users to adopt the same policy.

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

Delphi

Borland's Conference CD

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

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

Mixing Delphi and Access

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

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

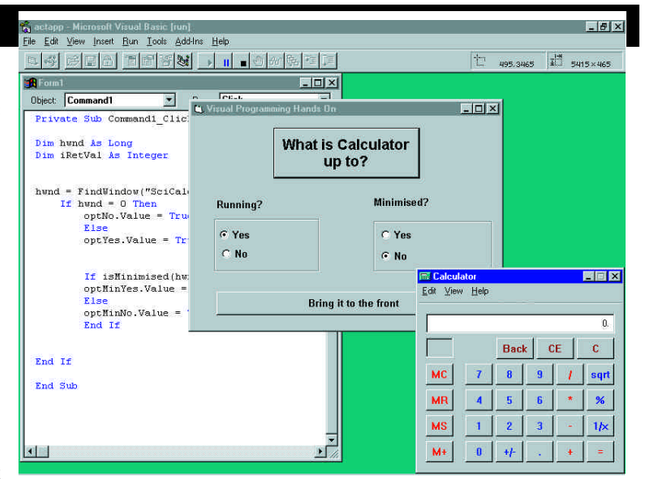
Thanks to the popularity of Microsoft Office Professional and Visual Basic, desktop data is frequently stored in Access MDB files. This creates a problem for other applications which need to get at the data, especially since Microsoft has never documented the structure of an MDB. In any case, the format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

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

Fig 1 Routine written from the DAO COM interface

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

begin

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

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

snMembers.close;
db.close;

end;
```


Powers of detection



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

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

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

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

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

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

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

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

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

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

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

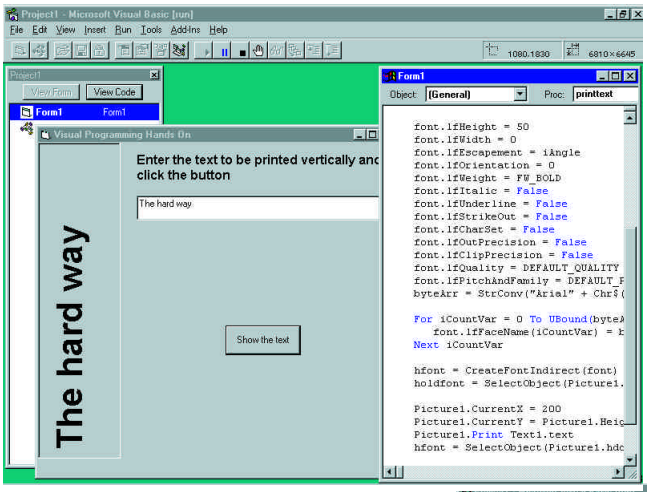
```
Function isMinimised(hwnd) As Boolean

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

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

End Function
```

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

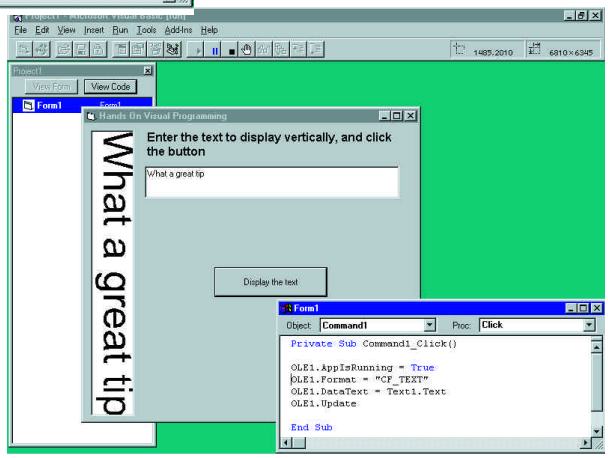
```
hwnd = FindWindow("OpusApp", vbNullString)
  If hwnd <> 0 Then
    If isMinimised(hwnd) Then
    iRetVal = ShowWindow(hwnd, SW_RESTORE)
    Else
    BringWindowToTop (hwnd)
    End if
  End if
```

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

p311 ➢



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



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

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

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

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

Visual Basic

Going vertical

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

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

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

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

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

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

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

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

Cover CD

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

PCW Contacts

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

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