Getting automated

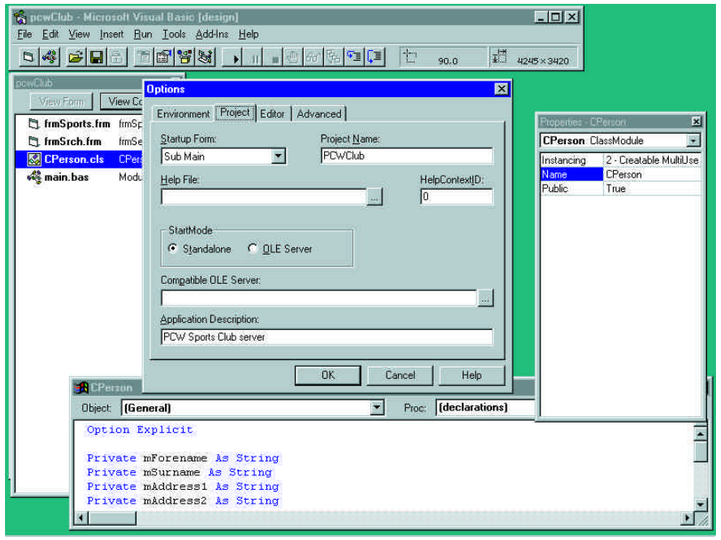
Tim Anderson does clever stuff with automation servers in the final part of the workshop. Plus, polishing the Sports Club database and adding some essential new functions.

Last month's workshop demonstrated how a Visual Basic class can be plucked out of the standalone version of VB and, with care, planted into Microsoft Office as a Visual Basic for Applications class. That is one way of re-using code, but an even better approach is to create objects that can communicate with any number of different applications without the need to recompile. With the rise of PC networks, and now the internet, this kind of software is the way of the future.

Under Windows, the way to achieve this is by using Microsoft's Component Object Model, or COM. This is the technology beneath OLE and ActiveX, and Visual Basic programmers can use it without knowing the detail of how it all works. For example, what if the PCW Sports Club wants to get at membership details not only via Word, but also from other programs like accounts and desktop publishing packages?

A key part of the Sports Club application is the CPerson class module which describes a club member. By making a few changes, that class module can become an automation object which exposes its properties and methods to other applications. The steps are as follows:

1. Open the CPerson module and press F4 to reveal the class properties. Set the Instancing property to CreateTable MultiUse, and the Public property to True.
2. Add a module to the project using Insert - Module. In the module, create a Sub Main.
3. On the Tools menu, in Options - Project, change the project name to PCWClub, and put a few words of description in the Application Description field. Finally, change the Startup Form to Sub Main.



Two key steps in creating an automation server are: first, the properties for the class to be exposed; and second, the project options

The project name must be unique to your automation server. The full class name of objects in your server will be of the form PCWClub.MyClass.

By taking these simple steps, you have created an automation server that lets other applications create and control objects of the CPerson class. All that remains is to register the class in the system registry. If you run the application in VB's development environment, it will be registered temporarily. If you build an .EXE or OLE DLL, it will be registered permanently. Then you can write code like this in Excel:

```
Dim myobj As Object
Set myobj = CreateObject
("PCWClub.CPerson")

  If myobj.FindPerson("Bloggs",
"Fred") Then
MsgBox "Fred lives at: " &
myobj.Address1
End If
```

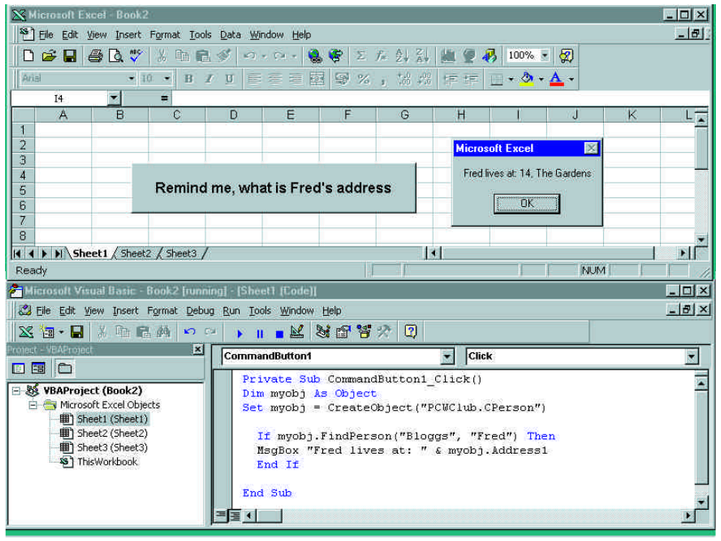
Although applications such as Excel function both as automation servers and standalone, most VB applications will be one or the other. Often, VB automation servers have no user interface, since this is provided by the client application. The workshop example, though, is designed to work in both guises. The trick is to use Sub Main to detect whether the application is running standalone or as an automation server. Here is the code:

```
Sub Main()

If App.StartMode <>
vbSModeAutomation Then
frmSearch.Show
End If

End Sub
```

You may wonder what Sub Main does when running as a server? The answer is, nothing at all. The only thing the server application does is to expose its classes so other applications can create and control objects. For testing, you can simulate this mode by setting the startmode in Project Options to OLE Server. Run the application and then minimise VB. Next, run another instance of VB, open the References dialog and check the PCW Sports Club server. Now you can test the server by creating objects of the CPerson class.

If you have run a compiled VB automation server such as PCWCLUB.EXE, it will be entered permanently in the system registry. Once you have finished testing, it is good practice to remove it. You can do so by running it from a command line with the /UNREGSERVER parameter. DLLs are unregistered using REGSVR32.EXE which is found in the System directory. Run it without parameters to see the switches.

Automation servers are powerful but do present some new programming challenges. The section of the Visual Basic manual called "Creating OLE Servers" is essential reading.



Excel is able to get at Sports Club details by using a VB automation server

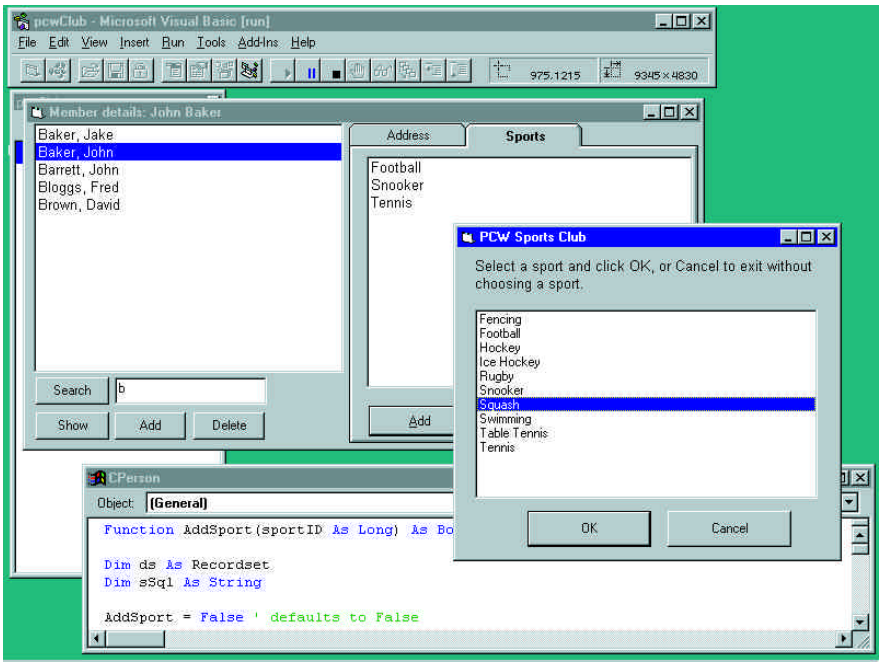
Adding the essentials

The Sports Club database is also used as a standalone application, and the version in last month's workshop is lacking some essential features. First, there is no way to add or delete members; and second, you cannot add or remove sports from the list which applies to each member.

The thinking behind the design of this simple application is that interface code belongs in the main form, while database code belongs in the CPerson class so that it can be used in other applications or as an automation server. The natural approach is to create new public methods for CPerson that give this new functionality. For example, here is code to add a new member:

```
Function CreateNew(sSurname As
String) As Long
' creates a new person in the
database
Dim lId As Long
myRecordSet.AddNew
myRecordSet!surname = sSurname
lId = myRecordSet!ID
myRecordSet.Update
Me.load (lId)
End Function
```

p260



Adding a sport to a member's list of interests is achieved via a simple dialog

The ID field is a counter, which means that the JET database handles the business of ensuring that the new member has a unique number. There is an issue, though, about how to cope with users who change their mind.

What if someone starts to create a new member, and then wants to back out and leave things as they were? One possibility is to call the AddNew method, but not to call Update until the user confirms the action. Unfortunately, bullet-proofing the application so that Update is only called after AddNew or Edit is prone to error. The easier approach is to minimise the time when JET has unsaved changes in its copy buffer. In this application, clicking the Add button creates a new member with the surname "Unnamed". If the user wants to cancel the addition, it is just a matter of clicking Delete.

The DeletePerson method is a little more involved. The problem is that there may be other records, in the SPORTLINK table, which refer to the member being deleted. To maintain data integrity, these records also need to be removed. Database objects have an Execute method which is an ideal solution. Execute takes an SQL command and applies it to the database. For example:

```
sSql = "DELETE * from SPORTLINK
where SPORTLINK.MEMBERID = " &
Str$(lId)
myDB.Execute sSql, dbFailOnError
```

The code at form level also has some work to do. When a member is deleted, the name must be removed from the list currently displayed, and the other fields on the form updated as necessary.

To make sense of adding sports to a member's list of interests, you need to throw a dialog listing the available sports. The dialog has a SportID property. To add a sport, the application takes these steps:

1. Show the Sports dialog modally, which means the user must either choose a sport, or cancel, before continuing.
2. When the OK button is clicked, the Sports dialog sets the SportID property to the currently selected sport.
3. Next, the program calls CPerson's AddSport method, passing the SportID as a parameter. AddSport creates a dynaset-type recordset which looks for records in the SPORTLINK table that match this member with the chosen sport. If the dynaset is empty, AddSport adds the required record. If it is not empty, AddSport reports that the member is already linked to that sport.
4. Finally, the program updates the form with the new list of sports.

Finishing touches

There is plenty more work to do in improving the Sports Club application. One professional touch is to enable and disable buttons according to whether or not they are applicable. For example, when no sports are listed, the Remove sport button should be disabled. Next, you can add keyboard shortcuts for mouse-free typing.

Another important area is error-handling, to prevent the program from crashing and to show the user informative messages when things go wrong. For instance, the database could become corrupted.

Finally, there is the issue of multiple users and what happens when two people try to update a record at the same time.

■ *All the code for this month's workshop is on the cover CD. And see Hands On Visual Programming (p301) for answers to queries concerning this workshop and other Visual Basic problems.*

Beating the OLE jargon

OLE has lots of strange jargon. and here are two examples that can cause confusion. Mastering these issues is important to make good use of the technology.

First, you will see references to in-process and out-of-process servers. In-process servers are DLLs which run in the same address space as the calling application, whereas out-of-process servers run in their own address space. This is a decision you take when building a VB executable. In-process servers have substantially better performance but introduce more programming restrictions.

Second, there is the matter of early or late binding. Binding is the process of locating the properties and methods which the client application calls. If you use variables declared as Object in the client application, then these identifiers are not resolved until runtime. This is called late binding. On the other hand, if you use an OLE-type library to resolve these identifiers at compile time, the code will execute faster. This is early binding. To use early binding in Visual Basic, open the Tools - References dialog and check the type library required. Then, declare variables of the specific class required, rather than the generic Object. This is much faster and also enables you to detect errors in parameters, properties or method names when the application is compiled. Another bonus is that you can use an object browser to inspect the interface of available classes.

Naturally, the best performance combines both techniques — that is, in-process servers called with early binding.

PCW Contacts

Tim Anderson welcomes your comments and queries. Write to the usual *PCW* address, or email freer@cix.co.uk.