All in a **day's** work

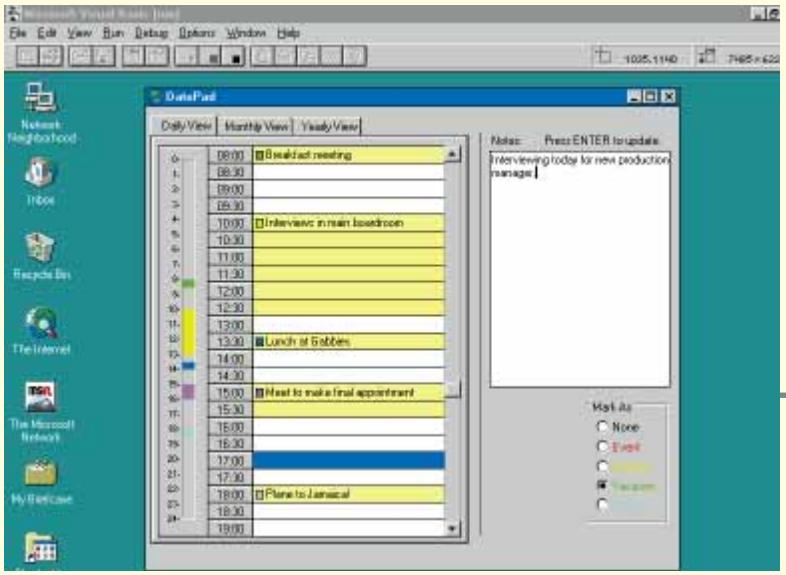
The OCX is designed as a universal Windows component. Tim Anderson tries it in a calendar application, explores DLLs in Delphi and glances at Visual dBase.

Fancy constructing your own PIM (personal information manager)? Sheridan's new Calendar Widgets makes the job easy. Calendar controls that show a month at a time are two a penny, but Calendar Widgets also offers yearly and daily view controls. The daily view control is what makes PIM construction so easy since it provides an editable grid of appointment slots complete with a time selection bar, and a facility to define overlapping tasks. The monthly and yearly controls can be bound to a table in an Access .MDB. There is also a date combo that drops down a graphical calendar, enabling users to enter dates by point-and-click. It's just the thing for applications like project management, scheduling, or presenting date-related information.

Calendar Widgets is useful, and interesting for the new techniques it demonstrates. Sheridan has hedged its bets by supplying three control types: version 3.0 VBXs for compatibility with VB 3.0 (but not Delphi), 16-bit OCXs for the likes of Access 2.0, and 32-bit OCXs for VB 4.0 or other OCX containers such as Visual FoxPro. Since Calendar Widgets plopped onto my desk before the final version of either FoxPro or VB 4.0, Sheridan has shown a touching faith in the stability of the OCX standard. All was well with our late beta versions.

The main benefit of OLE controls is compatibility. The VBX standard is a way of extending Visual Basic, and to make VBXs work with other environments such as Delphi or Visual C++ requires emulating VB. By contrast, OCXs use OLE 2.0 communication. There is a price to pay for this application-independence. Some aspects of performance may be improved, especially in 32-bit OCXs, but other things will be slower: for example, there is a noticeable pause when inserting a Calendar Widget OCX into Access or FoxPro, and again to display its property page, compared to the snappy performance of the VBX equivalent in VB 3.0. It is like working with embedded OLE objects in documents: you sense that Windows is running very fast under the surface to present the illusion of seamless integration. But once the OLE object is up and running, performance should be as good or better than a VBX. Processors are getting quicker, and the concept of universal components has great appeal, so I guess it's worth it.

Another area of doubt is over compatibility. You would have thought that OCX support would be all or nothing, but in fact OCX containers vary in the level of compatibility they offer. For example, Visual FoxPro does not support the ISimpleFrame interface, which means that you cannot embed one control in another. In practice, while all OCX controls should work in FoxPro, and in other OCX container environ-



Build your own PIM with Calendar Widgets. This VB 3.0 application demonstrates the DayView control

Calendar confusion

It is disheartening when the sample code supplied with a product fails to run. This was the case with Datepad, a simple PIM example in Calendar Widgets. The reason was a problem with international date formats. The code that failed was like this:

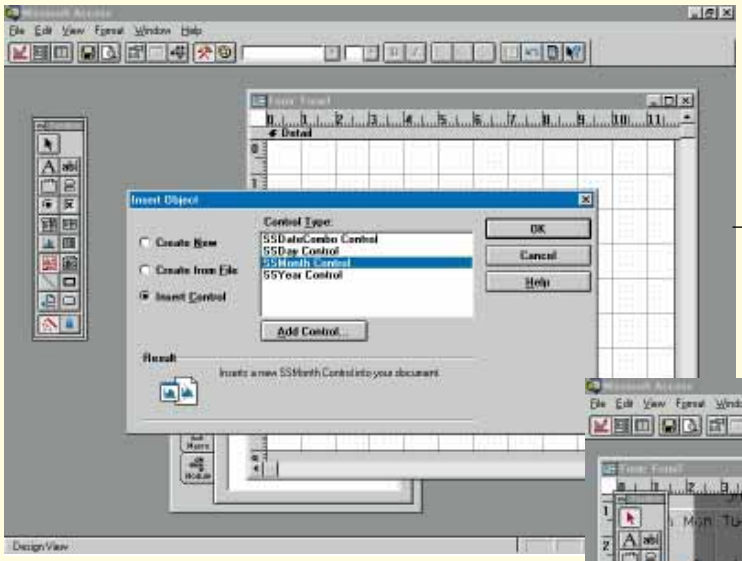
```
sMySql = "select * from appoint
   where date = #" & sSelDate & "#"
Data2.RecordSource = sMySql
Data2.Refresh
```

where sSelDate is the date returned from a calendar control, converted to a string. Because my PC is set up for UK dates, the calendar control was correctly returning a date in dd/mm/yy format. But JET's SQL expects a date in mm/dd/yy format, whatever the international setting in Windows. So the application failed with an error: "Syntax error in date in query expression."

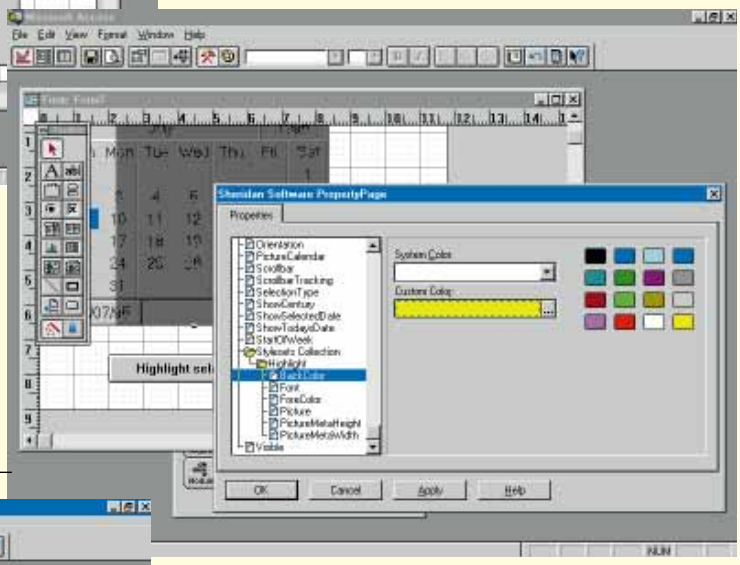
VB and Access store dates internally as a double-precision number. You can take advantage of this to overcome the above problem. The following code works on both sides of the Atlantic:

```
Dim dDate as Double
dDate = DateValue(sSelDate)
sMySql = "select * from appoint
   where date = " & str$(dDate)
Data2.RecordSource = sMySql
Data2.Refresh
```

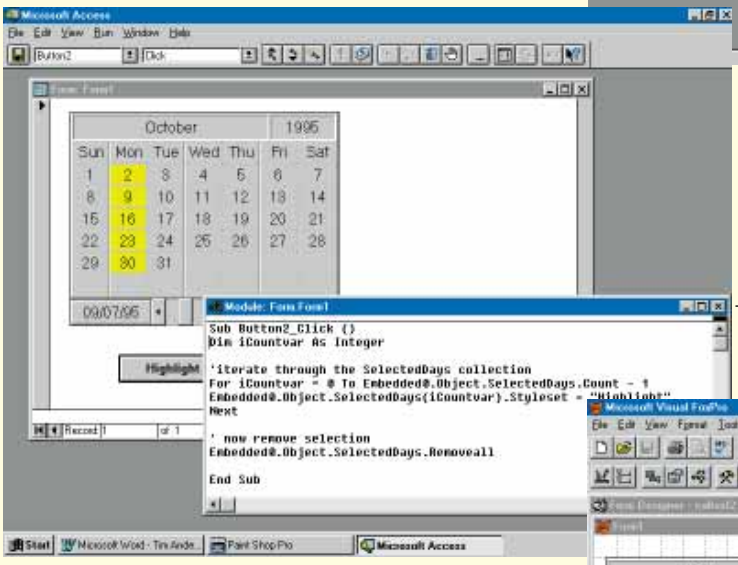
Using OCXs in other applications



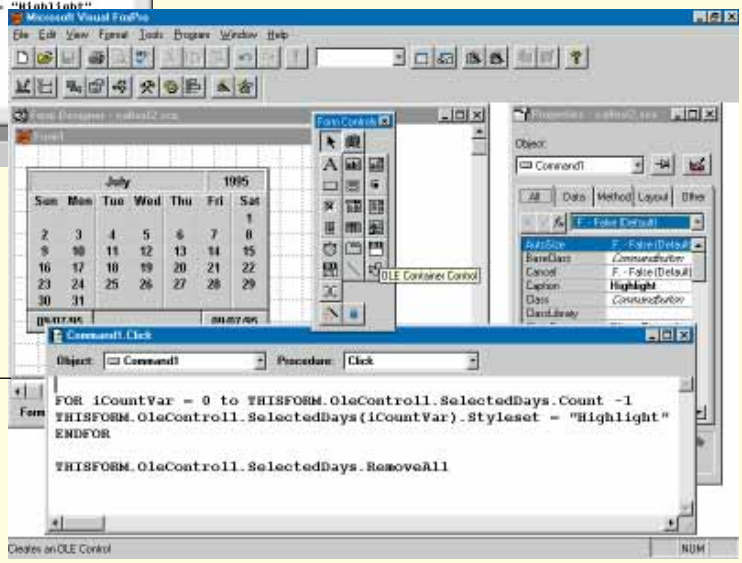
To use an OCX in Access 2.0, first place an Object Frame on a form. From the dialogue which opens, select Insert Control. Then you can choose from all the OCXs registered on the system. Note that Access 2.0 can only use 16-bit OCXs, which are likely to be less common than the 32-bit variety



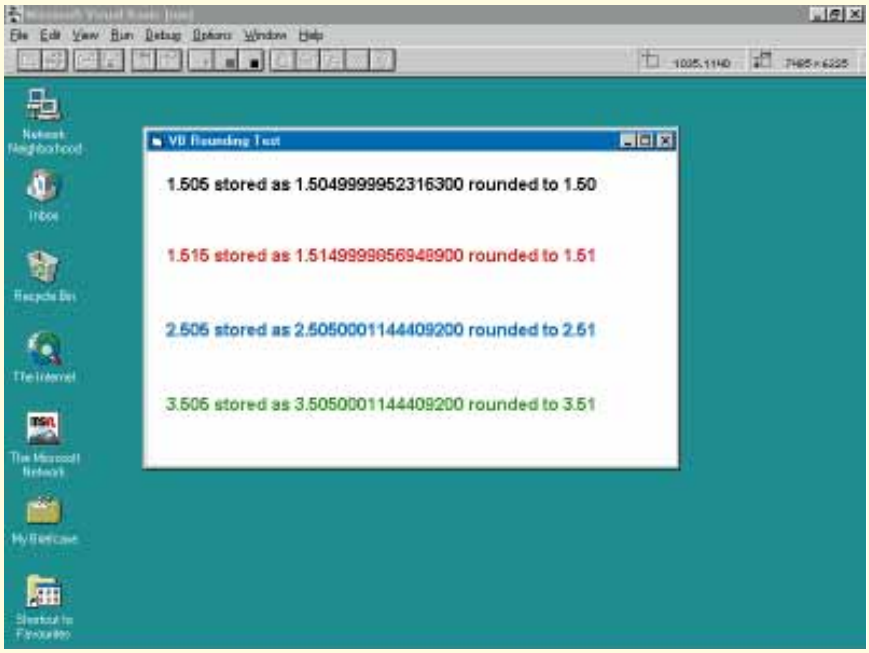
Next, the control can be customised by setting properties at design-time. The Property Sheet is a dialogue created by the OCX control itself, normally accessed by right-clicking on the control. That is one reason why OCXs are larger than VBXs: they contain their own user interface. This example adds a user-defined styleset to the control. A styleset in Calendar Widgets defines the appearance of a date cell

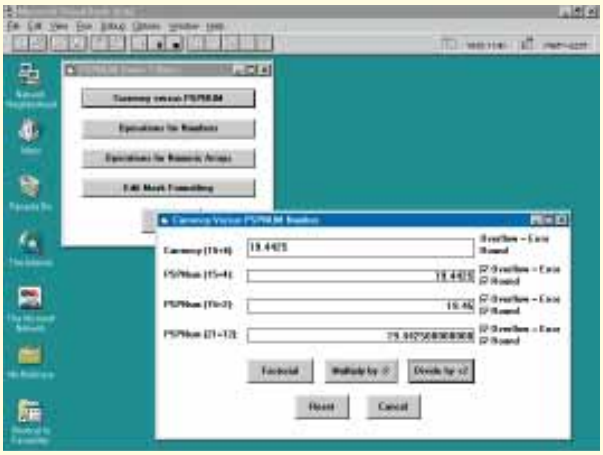


The form at runtime, showing some highlighted date cells. The code which does the work refers to the OCX control, laboriously, as Embedded0.Object. Access 95 supports the With ... End With construct to tidy up code like this



Using a Calendar Widget in Visual FoxPro requires painting an OleControl frame on a form, then inserting the control from a list of available OCXs. Right-click brings up the property page; and xBase code addresses the control in much the same way as Basic





Above *John Plumbley's demonstration that VB has rounding problems too*

Left *Prospero's PSPNUM is a VBX to handle fixed-length precision numbers and arrays*

ments, they do not all offer the same functionality. As another example, OCXs in FoxPro have no data-aware capabilities. A final problem is that Visual C++ in its current version (2.1) has no integrated support for OCX controls. Expect this to change soon. Borland also intends to support OCXs in future versions of Delphi, dBase and C++. In the meantime, don't assume that an OCX control is really plug-and-play and check before purchase that it will do what you need. At worst, a particular OCX could be next to useless; at best, the vendor will have developed with each of the main supporting environments in mind.

Not just Delphi

Following the discussion of floating point rounding in the August issue, thanks to John Plumbley for the following email:

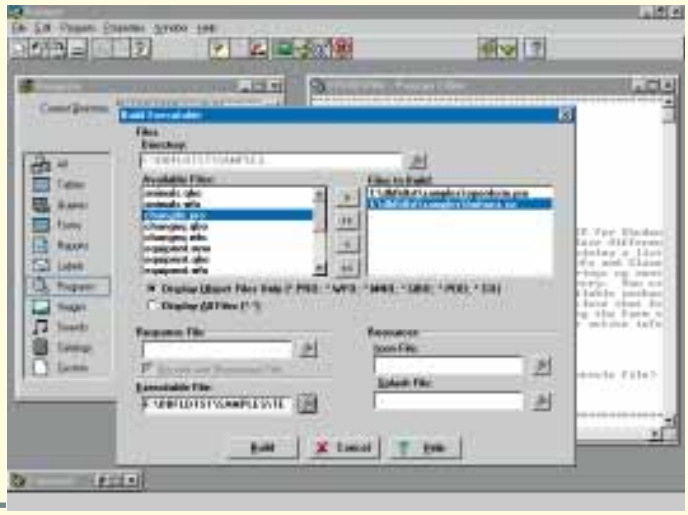
"This does not seem to be banker's rounding — Borland (or the company that does the tech support for them) must have been wrong when it said it was.

It is caused by the way computers store fractional floating point numbers and is a well known problem for the unwary.

To see what's going on use FmtStr with %.15f and you will see that:

```
1.505 is stored as 1.504999999999200
(which %.2f rounds down to 1.50);
1.515 is stored as 1.514999999999418
(which %.2f rounds down to 1.51);
2.505 is stored as 2.505000000001019
(which %.2f rounds up to 2.51);
3.505 is stored as 3.505000000001019
(which %.2f rounds up to 3.51).
```

A similar thing happens in C, and it also



From this dialogue you can build an executable in Visual dBase, although in dBase "compiled" is just another word for interpreted

happens in VB, not only in Delphi.

In my opinion this behaviour is not a problem, as anyone doing this kind of work should not be using fractional real numbers. They should scale all numbers to avoid fractions (i.e. work internally in pennies, or tenths of a penny, instead of pounds)."

John includes listings in Delphi, VB and C to prove his point. So if you are rounding matters in your application — you have been warned. I've also received a VBX intended to overcome exactly this kind of problem from a small company called Prospero. PSPNUM.VBX is a masked edit control which allows you to work with fixed-length precision numbers; a more flexible version of VB's Currency data type. It's a specialist tool which could be valuable for accounting and business applications.

dBase gets Visual

The word "visual" is beloved of marketing departments worldwide, judging by the way it pops up in new product versions. The latest casualty is dBase, now at version 5.5 and called Visual dBase. The xBase language is about as non-visual as you can get, but both Microsoft and Borland have added object-orientated language extensions which, combined with visual design tools, lend some credibility to the "Visual" tag.

I looked at a late beta of the product. Despite the change of name, Visual dBase is not a major upgrade, but adds significant enhancements to the original dBase for Windows. It is not a 32-bit product, although it does recognise Windows 95 with limited support for long filenames. Since neither the IDAPI database engine nor the Crystal Reports report writer support long filenames, dBase developers should stick with 8.3 names until the release of 32-bit versions of these products. OLE automation is now supported, but not OCX controls. There are few language changes but two are particularly welcome: class definitions can now include protected properties (without which true encapsulation is impossible), and at last SQL queries can be executed on local tables without the need for any special SQL mode of operation.

Borland has long been rumoured to be working on a dBase compiler that would create true executables in the style of Delphi or C. Sadly, the project seems to have been abandoned, as the compiler now released for Visual dBase is no more than a runtime version. It's nicely integrated into the product and comes with an Install Builder for creating distribution disks. Overall, Borland has plugged many of the gaps between dBase and Microsoft's Visual FoxPro, arriving on the market at about the same time. Borland's product has a snappier, less cluttered interface and slightly more modest system requirements.

Despite the new features this is an interim release, although likely to be the last 16-bit version, and the dBase to watch for will be the 32-bit version for Windows 95 and Windows NT. It's strange how it always seems to be the next version that promises to deliver the goods.

Calling DLLs in Delphi

Alan Bontoft sent the following query:

"I am a recent convert to the world of Delphi (from VB of course) and am trying to write software which involves the use of third-party DLLs.

"When used with VB, one simply has to declare the procedure/function in a .BAS file and it can then be accessed by any of the forms in the application. I came to use a DLL called INPOUT.DLL with Delphi, and after much searching in both the online help and Delphi manuals I came up with the following declaration:

```
procedure Out(Port, Value: integer);
far; external 'D:\DLLS\INPOUT';
```

"This seems to work fine, but only if placed in the implementation part of a unit. As soon as it is placed in the interface part, which is where a declaration has to be in order for other units to access it, an error occurs. Does this mean a DLL procedure has to be declared in every unit in which it is used?"

You *can* declare a .DLL in the interface part of a unit. For example:

```
unit CallDll;
interface
procedure Out(Port, Value: integer);
function DiskSpaceFree(DiskID: byte):
  longint;

implementation
procedure Out; external
  'D:\DLLS\INPOUT'
function DiskSpaceFree; external
  'SETUPKIT';
end.
```

I've included a function (from the SETUPKIT.DLL distributed with VB) alongside your procedure. Remember, that even though you declare the function in the interface section, to make it public you still need to make an entry in the implementation section to show that its implementation is to be found externally. Another point is that procedures in the interface section are implicitly far; you don't need the "far" keyword.

The most common method is to stuff DLL declarations into a unit on their own. Then, any units in your application which need to call these functions can simply add the unit name to the "uses" clause. That is how Delphi handles the Windows core functions, which are defined in units called WINTYPES and WINPROCS.

PCW Contacts

Tim Anderson welcomes your feedback, tips and suggestions. Contact him at the usual *PCW* address or as **freer@cix.compulink.co.uk**, or **100023.3154@compuserve.com**

PSPNUM: Prospero Software Products **01624 681090**
Calendar Widgets: Contemporary Software **01727 811999**
Visual dBase: Borland **01734 320022**