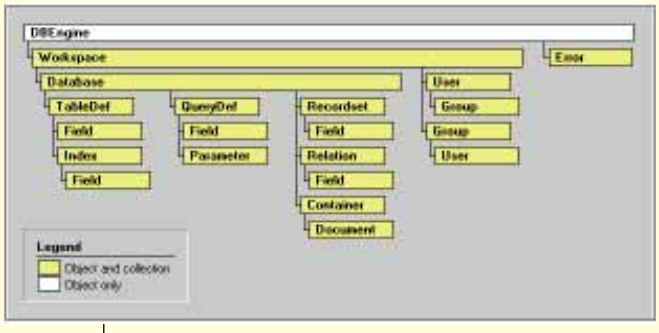
# Dealing with **data**

**Tim Anderson** **explores data access in Excel 7.0, why Designer Widgets was released full of bugs, and how to access text properties in VBXs within Delphi. Plus, seven tips for visual programming.**

Version 5.0 of Excel introduced Visual Basic for Applications, the common macro language that is intended to unify the programming of Microsoft applications. VBA's migration to the main Office components is slow. Access 7.0 now includes it, but unbelievably Word 7.0 remains stuck with WordBasic. The WordBasic language is not so bad, although there are annoying differences; but more seriously Word is not an OLE automation client. It works as an OLE automation server but does so crudely, compared with Excel or Access. The only object made available is Word.Basic, through which you can call WordBasic routines.




*The hierarchy of data access object classes; essential information for database programming in Visual Basic for Applications*

While we wait for Word to catch up, there's plenty to do with VBA in Excel. In particular, the new Data Access Objects (DAO) open up the JET database engine for OLE automation, giving Excel excellent database features and, without the use of the old ODBC add-in, XLODBC.XLA.

There is a natural synergy between spreadsheets and databases. Spreadsheets are well suited to analysing and modelling temporary data, while databases are ideal for robust, permanent storage. Using DAO you can easily write programs that transfer data between Excel and database tables. For example, an Excel application could obtain a customer's order history, feeding the details into a spreadsheet for charting or sales projections. Here's how you might go about it.

The first step is to insert a macro module into an Excel 7.0 workbook. Then, from the Tools menu, choose References and check DAO 3.0 Object Library. This enables Excel to use all the DAO classes and constants. The procedure to retrieve order details might look like *Fig 1*.

This simple example returns a query result as a snapshot and enters the results into a worksheet. The data access code works exactly as it would in Visual Basic 4.0. But DAO will do more than just execute queries: using the data definition language you can create databases with security, encryption and referential integrity. The workspaces collection lets you set up simultaneous data access sessions, with support for transactions. Data can be written as well as read, either by opening an editable recordset (a table or dynaset) or by executing SQL statements. VBA and DAO, combined with a good knowledge of SQL, gives Excel full-blown data management features. All we need now is VBA in Word to open up the same possibilities there.

## Designer Widgets

The migration of VBX add-ons to OCX components continues with the arrival of 32-bit Designer Widgets from Sheridan. Version 1.0 of this product is widely used to add dockable toolbars and tabbed dialogues to VB applications. Evaluating this new version, which comes in triplicate (VBX, 16-bit OCX, 32-bit OCX), proved

## Seven tips for VB and Delphi

**Visual Basic**

**1.** JET loves SQL. If you can use SQL Select or an Execute method, rather than navigational database code, performance is almost always better.

**2.** There are experienced VB programmers who do not realise that the code window can be split: place the mouse just above the thin grey line at the top of the window, and drag down to obtain two separate scrolling sections.

**3.** Avoid variants by declaring all variables. Set Require Variable Declaration to Yes in Environment Options to ensure that Option Explicit appears at the top of all modules.

**Delphi**

**4**. Windows 3.x uses co-operative multitasking, which means your application must regularly yield control back to Windows to avoid locking the user out of other tasks. In Delphi you can use Application.ProcessMessages to give processing time to other applications.

**5**. Re-use your code by developing your own utility functions for frequent tasks. Place these in one Pascal unit, and add it to the Uses clause of any other units that call these functions.

**6**. Use Inc and Dec to generate tightly optimised code, instead of statements like

```
iVar := iVar + 1;
```

**7**. Delphi has no direct equivalent to VB's useful App.Path. You can obtain the same information (the location of the application) by parsing Application.Exename. This returns the application name with its full path. Search the string for the last occurrence of "\" to find the path alone.

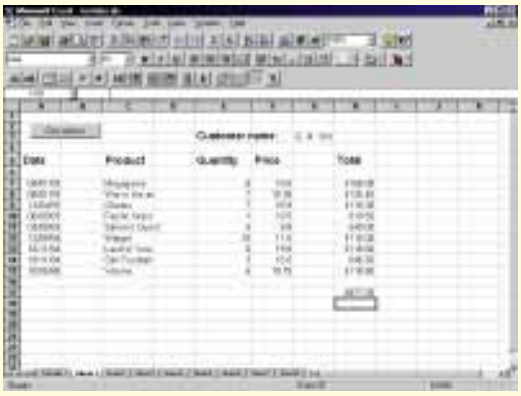
frustrating. This was the release version, not a beta.

The Readme informed me that the product did not work quite as documented. Two major problems were that the tab and notebook VBXs did not work data-bound, and that the FormFX control did not work under Windows 95. Then I tried some of the controls using a late beta of VB 4.0, 32-bit version. Using the tab control first, I noticed that the property page took an age to appear. When it did, I found that altering one of the properties caused an obscure error message, followed by a complete shutdown of Visual Basic 4.0. Presuming this to be a problem with the VB 4.0 beta, I reverted to VB 3.0 and tried the VBX controls. Performance was improved; but I soon managed to crash VB 3.0 as well.

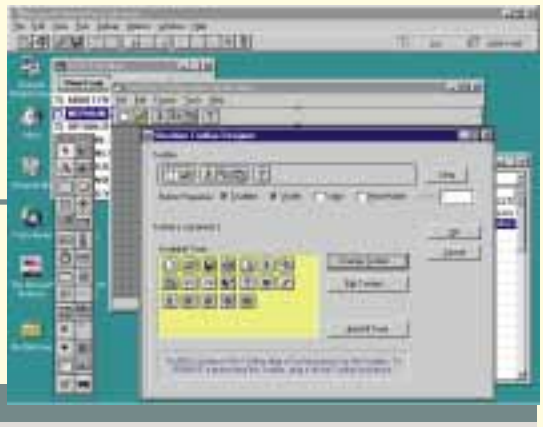
Guessing something was up, I logged on to Sheridan's forum on Compuserve. There I found a host of angry messages from attempted users of Designer Widgets 2.0, a 1.2Mb patch file in the library, and a further string of messages claiming that the patched version was little better. The concerns were mainly to do with stability (as I had discovered), but also incompatibility with projects built using Designer Widgets 1.0.

To be a supplier of third-party tools is a difficult and dangerous game. Just at this moment it is more difficult than ever. Windows 3.x is dying, and Windows users are moving to some combination of Windows 95, NT, or OS/2. On top of that, the main VBX container environment, Visual Basic itself, has just been released in several new versions. Until recently, long delays in Windows 95 and VB 4.0 have made it hard to release new products.

Finally, companies like Sheridan are in the ridiculous position of having to support three component standards — VBX, 16-bit OCX



**Above** *Using Excel 7.0 and data access objects, data can be extracted from database tables as easily as in Visual Basic itself*



**Right** *Designer Widgets 2.0: good when it works. The toolbar designer is a neat way to assemble your own dockable toolbar*

**Fig 1 Procedure to retrieve order details**

```
Sub GetOrders()
Dim db As Database
Dim snOrders As Recordset
Dim sSql As String

Dim sCustNo As String
Dim iOrderCount As Integer
Dim iCountVar As Integer

Clear sheet
Worksheets("sheet1").Cells.ClearContents

'Get a customer reference number
sCustNo = InputBox("Enter customer number")

'Construct SQL query
sSql = "Select * from customer, orders, products "
sSql = sSql & "where customer.cust_no = orders.cust_no
"
sSql = sSql & "and orders.prod_no = products.prod_no "
sSql = sSql & "and customer.cust_no = '" & sCustNo & "'
"
sSql = sSql & "order by orders.date_rcd"

' Open the database
Set db =
DBEngine.OpenDatabase("C:\TESTDATA\MAINDB.MDB")

'Create a snapshot based on the required customer num-
ber
Set sn = db.OpenRecordset(sSql, dbOpenSnapshot)

' Check for no results, find record count
If Not sn.EOF Then
sn.MoveLast
iOrderCount = sn.RecordCount
sn.MoveFirst
End If

' Enter customer name into worksheet
If iOrderCount <> 0 Then
MsgBox "There are " & Str$(iOrderCount) & " orders for
this customer"
Worksheets("sheet1").Cells(3, 7).Value = "'" & sn!Cust-
name
Else
MsgBox "There are no orders for this customer"
End If

' Enter order details into worksheet
For iCountVar = 0 To iOrderCount - 1
With Worksheets("sheet1")
    .Cells(7 + iCountVar, 1).Value =
DateValue(sn!Date_rcd)
    .Cells(7 + iCountVar, 3).Value = "'" & sn!Product
    .Cells(7 + iCountVar, 5).Value = Val(sn!Quantity)
     .Cells(7 + iCountVar, 6).Value = Val(sn!price)
    .Cells(7 + iCountVar, 8).Value = .Cells(7 + iCount-
Var, 5).Value  * .Cells(7 + iCountVar, 6).Value
End With

sn.MoveNext
Next

' Close database
db.Close
```

and 32-bit OCX, and these are meant to work in a variety of different and not completely compatible environments. Sheridan's president Bob Wolf had the grace to post an apology on Compuserve, and pointed out that, "The permutations between versions (VBX, 16- and 32-bit OCX), host environments (VB 3.0, VB 4.0, Access, etc.) and operating systems (Win 3.x, WFWG, NT, Windows 95) turn out to be 71 separate test environments." Sheridan therefore deserves some sympathy, although it remains extraordinary that a product as unsteady on its feet as Designer Widgets 2.0 should acquire shrinkwrap status.

And what about the product? Presuming Sheridan soon delivers a solid release, and improves performance, it has good potential. There are many small improvements: the index tab control can be data-bound to give an instant card-file database; and a new notebook tab control allows creation of a comforting spiral-bound interface. Its appeal is a little reduced for users of VB 4.0 and Windows 95, since toolbar and tabbed dialogue controls come as standard, but the Designer Widgets 2.0 versions offer many extra features. Just make sure you get one that works.

## Delphi, VBXs and strings

If you have used VBX controls in Delphi, you may have come across a problem when trying to use string properties. When you set up Delphi to use a VBX, using the Install Components dialogue, it creates a Pascal wrapper unit in your Windows System directory. Among other things this wrapper converts VB property types into Delphi types. In the case of VB strings, properties are converted to Pascal strings limited to 256 characters in length. For example, the TX Text control, which is supplied by Borland in the Delphi RAD pack, has several string properties such as Text and SelText. The Pro version of the same control has an RTFSelText property, needed for programmatic access to formatted text. Delphi exposes these properties, but helpfully truncates them to fit into a Pascal string, rendering them useless.

There are solutions. First, note that all Delphi controls support two methods, SetTextBuf and GetTextBuf. These methods work with pchars, solving the 256-character limit. Internally they work with the API messages WM_SETTEXT and WM_GETTEXT. These are straightforward to use, although before calling GetTextBuf you must first determine the size of buffer required with GetTextLen. *(Fig 2.)*

This works fine if it is simply a text property which you need to obtain. But what text you set or obtain depends entirely on how the control has implemented the WM_SETTEXT and WM_GETTEXT messages. If there is more than one text property, as with the TX Text control, this solution is not enough. There is another way. A number of functions in the BIVBX unit give lower-level access to VBX properties. Two of these are declared in BIVBX.INT, as follows:

```
function VBXGetPropByName(Ctl: HCTL;
 lpszName: PChar; lpValue: Pointer):
 Err;
function VBXSetPropByName(Ctl: HCTL;
 lpszName: PChar; lVal: Longint):
 Err;
```

These functions enable you to get and set properties using pointers. You will need to put BIVBX in the Uses clause of any unit which calls them. For example, to retrieve the SelText property of a TX Text control, use the code in *Fig 3*.

Be warned that calling functions in BIVBX is uncharted territory, not documented by Borland. You should also note that the code will need amending according to the type of property you are accessing.

**Fig 2 Determining buffer size**

```
procedure TForm1.Button1Click(Sender: TObject);
var
  lpzTextString: PChar;
  lSize: Longint;
begin
  lSize := TextControl1.GetTextLen;
 {allocate memory allowing for null character}
  GetMem(lpzTextString, lSize + 1);
  TextControl1.GetTextBuf(lpzTextString,lSize);
 {...do what you want with lpzTextString}
  FreeMem(lpzTextString, lSize);
end;
```

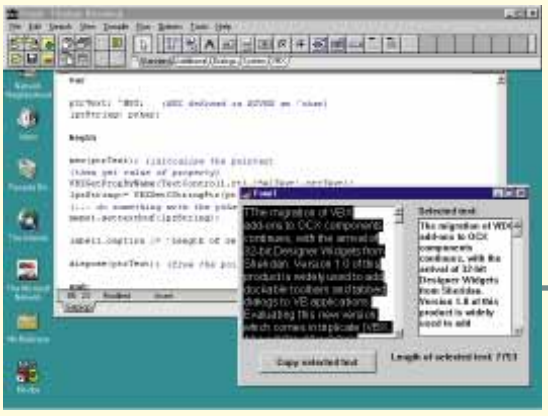
**Fig 3 SelText property in TX Text control**

```
procedure TForm1.Button1Click(Sender: TObject);

var
ptrText: ^HSZ;   {defined in BIVBX as ^char}
lpzString: pchar;

begin
new(ptrText); {initialise the pointer}
{then get value of property by name}
VBXGetPropByName(TextControl1.ctl,'SelText',ptrText);
lpzString:= VBXGetCStringPtr(ptrText^); {convert to pchar}
{... do something with the pchar}
dispose(ptrText); {free the pointer}

end;
```



*Obtaining the value of the seltext property in a VBX is not straightforward, but it can be done*

### PCW Contacts

**Tim Anderson** welcomes your Visual Programming comments and tips. He can be contacted via *PCW* at the usual address, or **freer@cix.compulink.co.uk**

**Designer Widgets** costs £99 from Contemporary Software on **01727 811999**.