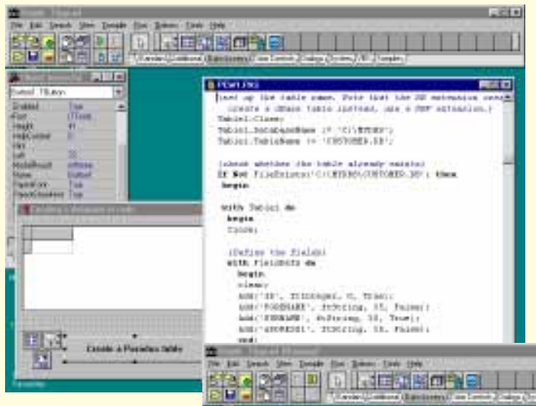
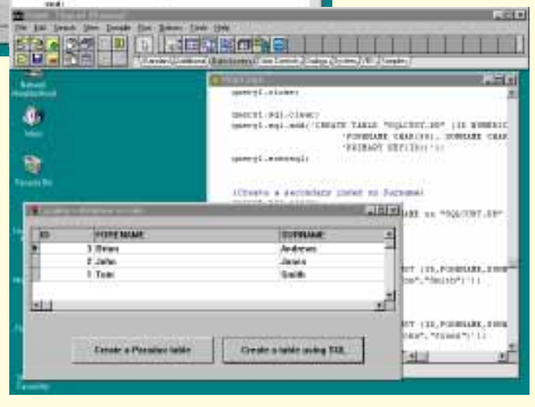
Table manners

You would hardly know it from the manuals, but Delphi lets you create database tables entirely in code. Tim Anderson shows how, checks out a newcomer to Windows visual development, and offers ten handy Visual Basic tips.

Borland has been so keen to emphasise Delphi's ease of use that some of its best features are almost hidden. An example is the creation of new database tables. The documentation encourages you to wheel out the Database Desktop to do this interactively; but what if you want to create a database in code? It is more complex but has many advantages, particularly for applications which will be distributed. Users do not want to define tables, nor should they have to struggle with the Borland Database Engine configuration utility. All these operations can be done more tidily in Delphi code. Unfortunately, the documentation for creating tables on the fly is all but hidden. Here is a demonstration of two ways to do it.





Above *It is quite possible to create tables in Delphi code, without using the Database Desktop. This example creates a Paradox table*

Right *Running the application creates the data tables, fills them, and displays the results in a bound grid*

The first and quickest method is to use a TTable component. The step-by-step method is as follows:

1. Start a new project in Delphi and place TTable, Datasource and DBGrid components on the form. The grid is not strictly needed to create the table, but gives a way of viewing the results. Next, place a button on the form with a caption of "Create table". The code to do this will go in the button's Click procedure.
2. Double-click the button and enter the following code. Note that this procedure assumes the existence of a MYDBS directory on your C drive. *(Fig 1.)*

This code defines the fields for the new table. There are four parameters to the FieldDefs.Add method. The first names the field. The second defines the field type, and the possible values can be found in Delphi's online help under TFieldType Type. The types available will vary according to the database format used.

The third parameter is the size of the field, and is only meaningful where the size is not already determined by the field type. The last parameter states whether or not it is a required field. *(Fig 2.)*

Indexes are defined using the IndexDefs.Add method. The first two parameters define the index name and field respectively. The third parameter is a set of type TIndexOptions. Again, not all the options apply to all database formats. For example, in dBase the primary key is meaningless. *(Fig 3.)*

3. Now run the code. The BDE creates and fills the table. Performance is fine, as long as the hardware is sufficient to run the BDE in the first place — realistically you need an 8Mb 486 machine or better.

Another option is to use SQL. The latest versions of Borland's database engine can use SQL on local tables as well as on server databases. In Delphi (against intuition) you use a Query component to execute SQL instructions, even when there is no result set to return. Here is how it works.

1. Start a new project, and add Query, Datasource and DBGrid components. Again, the latter two are only needed to display the data. Finally, add a button and open its Click method.
2. Next write code as *Fig 4*.

The Clear, Add, ExecSql sequence is the fundamental technique for executing SQL instructions on a database. In this example the query.DatabaseName property is set to a directory name, which tells the BDE that it is working on a local, desktop database. The other problem is finding the correct syntax for the SQL commands. Not all SQL commands are supported for local databases, and the main restrictions are summarised in Appendix C of Delphi's Database Application Developer's Guide. Note that double quotes are used within the SQL string, to avoid conflict with Delphi's single-quote string delimiter.

The SQL shown creates a Paradox table with a primary key. The primary key must be defined in this statement; you cannot add it afterwards. *(Fig 5.)*

So which is the best technique? There does seem to be some performance penalty for using local SQL, but the advantage is that the same code will run if you later convert your application to talk to a SQL backend server. Ideally you should try both the methods described to discover which gives the best performance in your particular system.

Fig 1

```
{set up the table name. Note that the DB extension creates a Paradox table.
To
  create a dBase table instead, use a DBF extension.}
Table1.Close;
Table1.DatabaseName := 'C:\MYDBS';
Table1.TableName := 'CUSTOMER.DB';

{check whether the table already exists}
If Not FileExists('C:\MYDBS\CUSTOMER.DB') then
 begin

 with Table1 do
  begin

 Close;
  {Define the fields}
  with FieldDefs do
    begin
    clear;
    Add('ID', ftInteger, 0, True);
    Add('FORENAME', ftString, 35, False);
    Add('SURNAME', ftString, 35, True);
    Add('ADDRESS1', ftString, 35, False);
    end;
```

Fig 2

```
{now define indexes}
  with IndexDefs do
    begin
    clear;
    Add('ID','ID',[ixPrimary, ixUnique]);
    Add('SURNAME','SURNAME',[]);
    end;
```

Fig 3

```
  {actually create the table and open it}
  CreateTable;
  Open;

  {Append some records}
  Table1.AppendRecord([1, 'Brian', 'Smith','1 The Street']);
  Table1.AppendRecord([2, 'Joe','Brown','34 The Square']);
  Table1.AppendRecord([3, 'Martin','Wilson','43 The Close']);
  Close;
  end;

{Display a confirming message}
Application.MessageBox('Table created', 'Personal Computer World', mb_OK);
end;

{Display the table in the grid}
Table1.IndexName := 'SURNAME';
Table1.open;

datasource1.dataset := Table1;
dbGrid1.Datasource := datasource1;
dbGrid1.Refresh;
```

Into the Fourth Dimension?

Mac users have enjoyed an elegant graphical interface for years. Together with Apple's famous Human Interface Guidelines you might imagine this head start gives long-standing Mac developers an advantage over their Windows counterparts. In reality, the greater clout of the Windows marketplace more than compensates, and Mac development tools are poor in comparison.

The database area is particularly weak on the Mac, since companies almost always use PCs for serious database work. Although FileMaker Pro has won many hearts as a flexible end-user database, developers have been left to choose between 4th Dimension (4D), Omnis 7 or the Windows-like FoxPro; 4D is the biggest fish in this relatively small pond. ACI is now releasing a Windows version of 4D, which means that all the leading Mac databases are now cross-platform tools.

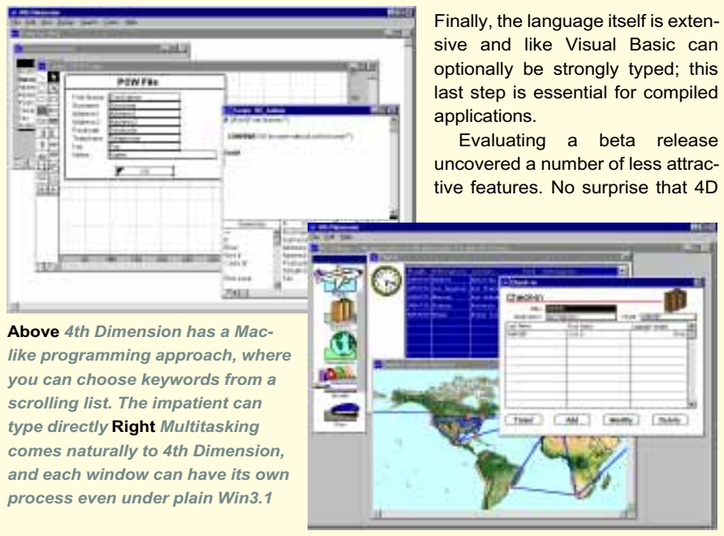
Taking the good points first, 4D joins Clarion, Visual Objects and Delphi in providing a compiler that creates native machine-code. You are still saddled with a large runtime database engine, but the speed improvements are impressive. The compiler can produce fat binaries that will run either as 68000 or Power PC applications on the Mac, or even (it is claimed) wrap Windows and Mac binaries into a single executable. Second, there is support for multiple processes, implemented by 4D itself on the Mac and on Windows 3.x, and by the operating system on Windows 95 and NT, which support multithreading. Third, 4D is usable on the desktop or as a database server, and it has built-in referential integrity, a visual relationship builder, and password-based security. The server product has a version control system for team development.

Fig 4

```
Query1.Close;
Query1.DatabaseName := 'C:\MYDBS';

{The extension determines the data format}
If Not FileExists('C:\MYDBS\SQLCUST.DB') then
  begin
  query1.close;

  query1.sql.clear;
  query1.sql.add('CREATE TABLE "SQLCUST.DB" (ID NUMERIC(10,2),'+
                 'FORENAME CHAR(35), SURNAME CHAR(35),'+
                 'PRIMARY KEY(ID))');
  query1.execsql;
```



Above *4th Dimension has a Mac-like programming approach, where you can choose keywords from a scrolling list. The impatient can type directly* **Right** *Multitasking comes naturally to 4th Dimension, and each window can have its own process even under plain Win3.1*

Finally, the language itself is extensive and like Visual Basic can optionally be strongly typed; this last step is essential for compiled applications.

Evaluating a beta release uncovered a number of less attractive features. No surprise that 4D looks Mac-like even under Windows, and does not approach either the polish or the flexibility of competing products like Access, Delphi or Visual Basic. For example, a 4D button can run a script when clicked and that is its limit. A VB button supports eleven events, including GotFocus, DragOver, MouseDown, MouseMove and MouseUp.

4D has a minimal range of screen objects. It does not support VBX or OCX controls, although it has its own native components called external objects, created in C or Pascal. The language has a crippling limitation: you cannot directly call functions in most DLLs or any Windows API functions, although you can compile special DLLs for use only by 4D. The language is fundamentally procedural, with no object orientation beyond the superficiality of a graphical interface builder. Finally, 4D has a complex and expensive range of add-ons which you have to buy to get full functionality. These include 4D Compiler, 4D Backup, 4D Server, 4D Chart, 4D Open for data access from other applications, and 4D remote for modem access.

As ever, the next version promises to be greatly improved. In the meantime, the main attractions of 4D are its compiled performance and its cross-platform features. If you have to support a mixed Mac and PC environment, 4D looks a reasonable but expensive option. Unfortunately, its numerous quirks and limitations make it unlikely to find favour outside that market.

Fig 5

```
{Create a secondary index on Surname}
  query1.sql.clear;
  query1.sql.add('CREATE INDEX SURNAME on "SQLCUST.DB"
(SURNAME)');
  query1.execsql;

 {Add some data using SQL INSERT}
 query1.sql.clear;
  query1.sql.add('INSERT INTO SQLCUST
(ID,FORENAME,SURNAME) ' +
                 'VALUES(1,"Tom","Smith")');
  query1.execsql;

  query1.sql.clear;
  query1.sql.add('INSERT INTO SQLCUST
(ID,FORENAME,SURNAME) ' +
                 'VALUES(2,"John","Jones")');
  query1.execsql;

  query1.sql.clear;
  query1.sql.add('INSERT INTO SQLCUST
(ID,FORENAME,SURNAME) ' +
                      'VALUES(3,"Brian","Andrews")');
  query1.execsql;

  end;

query1.close;
query1.sql.clear;

{Define the query}
query1.sql.add('SELECT * from SQLCUST ORDER BY
SURNAME');
query1.open;

{Display the data}
datasource1.dataset := Query1;
dbGrid1.Datasource := datasource1;
dbGrid1.Refresh;
```

Ten Visual Basic Tips

We shall regularly publish tips for Visual Basic, Delphi, VBA, Visual dBase and FoxPro and other popular languages. If you have a tip others may find useful, please post or email it to Tim Anderson at the address below. You will also find David McCarter's Visual Basic Tips and Tricks help file on this month's cover CD, containing hundreds of tips and examples for VB developers.

1. To make a window appear Always on Top, use the SetWindowPos API call with the HWND_TOPMOST flag. To remove the setting, use the same call but with HWND_NOTOPMOST. For example:

```
'Declares (needed if you do not
include WIN31API.TXT)
Declare Sub SetWindowPos Lib "User"
(ByVal hWnd%, ByVal
hWndInsertAfter%, ByVal X%, ByVal
Y%, ByVal CX%, ByVal cy%, ByVal
wFlags%)

' SetWindowPos Flags
Global Const SWP_NOSIZE = &H1
Global Const SWP_NOMOVE = &H2

' SetWindowPos() hwndInsertAfter
values
Global Const HWND_TOPMOST = -1
Global Const HWND_NOTOPMOST = -2
```

```
' In your program include the follow-
ing code to make Form1 always on top
Call SetWindowPos(Form1.hWnd,
HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE
+ SWP_NOSIZE)
```

2. Always save forms as text for reduced risk of corrupting your project, and compatibility with VB add-ons like the Setup Wizard. To do this by default, choose Options - Environment, select Default Save As Format, and choose the Text option.

3. When using VB to access external databases, attach them to an Access MDB for best performance. This ensures that JET holds the table structure in a memory cache.

4. Check your code for unused constants and declares, unused VBX controls, or even forms that are no longer used by your project. All these bloat to your application.

5. Don't use VBXs where native VB code will do. It is worth a little extra coding to get added performance.

6. Avoid hard-coding paths into your VB application. Sooner or later the path will be wrong and the code will break. Use App.Path to get the directory in which the application resides, and the API call GetWindowsDirectory to find where Windows is installed.

7. Check App.PrevInstance to find if your application is already running. For example:

```
Sub Main()
If App.PrevInstance then
Exit Sub
End If
```

8. You can make text boxes automatically select text when they get the focus. Use the following code:

```
Sub Text1_GotFocus ()
        Text1.SelStart = 0
        Text1.SelLength = 65000
End Sub
```

Since 65000 is near the maximum length for a text box value, VB will automatically select the whole text.

9. Use PICCLIP.VBX to store toolbar images or other graphics that need to be displayed quickly. It is very much faster than loading images from disk.

10. Never use a picture box where an image control will do. The image box is a lightweight control which uses far less system resources.

PCW Contacts

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

4th Dimension is from ACI UK **01625 536178**. Prices not yet available.