Into the **Visual Age**

Tim Anderson looks at IBM's plans for VisualAge Basic, finds a new visual implementation of Prolog, and introduces new sections for Visual Basic and Delphi.

All programming is visual now. The quick riposte is that most programming tools are resolutely procedural with an array of visual tools to disguise the fact. But there's no doubt that the likes of Visual Basic and Delphi have won the argument about how to program. That leaves a difficult choice for a column like this one. With so many visual tools out there, should it become a pot-pourri of miscellaneous programming news and tips? Or should it revert to being product specific, dedicated to Visual Basic which remains the most popular Windows development tool? A further complication is that third-party components in the form of VBX or OCX/ActiveX controls can be hosted by a variety of different programming tools.

In response, Visual Programming Hands On has been expanded and will be divided into three parts. The first will cover visual programming generally, including components that are useful in a wide range of products. The other two sections will cover Visual Basic and Delphi respectively, so that users of the two most popular general-purpose visual languages will always find something specifically for them. Much of the material in this column is a direct response to your comments and queries, so please keep them coming to me, by email or at the usual *PCW* address.

IBM's new BASIC

At the time of writing, IBM is in open beta with its version of Basic, an addition to the VisualAge product family. The press release refers superciliously to Basic as a "scripting language", but nevertheless IBM's release is a great testimony to VB's influence. The final release date has not been announced, but will be before the end of the year. It is a cross-platform product, with versions initially available for OS/2 and Windows NT. Windows 95 compatibility will follow, and it will be possible to deploy Visual Age for Basic applications on AIX.

With Visual Age for Basic, IBM seems to have several goals in mind. One is to make OS/2 a more appealing platform, by introducing an enormously popular language and making it easy to convert existing Visual Basic applications. It is also part of IBM's attempt to establish its preferred object model, SOM, on the Windows platform. Visual Age for Basic will support SOM, OpenDoc and OCX. Finally, it is a tool for IBM's DB2 database, with integrated access using embedded SQL and the ability to create stored procedures and user-defined functions.

As a DB2 add-on or an OS/2 utility, Visual Age Basic looks likely to succeed, but whether it will challenge Visual Basic itself on the Windows platform looks more doubtful. Judging by the beta, system demands are as high or higher, with 24Mb RAM recommended for development. Like VB 4.0, it is an interpreted language and unlikely to win on performance. It is broadly compatible with Visual Basic, but that compatibility does not extend to data access code. On the plus side, IBM promises a proper implementation of inheritance and, should SOM catch on, Visual Age Basic will be very useful. Look out for a full review in due course.

Visual Prolog

Back in the seventies, a partnership between two Frenchmen, one a computer scientist and one a logician, produced a new language called Prolog (short for Programming in Logic). Unlike procedural languages, which give step-by-step instructions to the computer, Prolog does problem solving by inference and recursion. To give you a flavour, here's a complete program that looks up a telephone number:

```
PREDICATES
nondeterm tel_no(symbol,symbol)

CLAUSES
tel_no("Bill","0123 4567").
tel_no("Jane","0765 4321").

GOAL
tel_no("Jane", Number).

In this case the output is:

Number=0765 4321
1 solution
```

Prolog's particular strength is in artificial intelligence and expert systems. It would be a good choice for an application that assessed insurance risks or for a program to create timetables for schools, trains or airlines. In the late eighties, Prolog was marketed by Borland as Turbo Prolog, following which rights reverted to the Prolog Development Center (PDC). PDC has now come up with Visual Prolog, a graphical development environment for Windows (16 and 32-bit) and in due course for OS/2. Visual Prolog includes layout editors and Code Experts, which allow you to create a graphical interface by using drawing tools and responding to dialogs. It works by means of a set of Prolog extensions called the Visual Programming Interface, a framework for controlling a graphical interface.

PDC argues that Prolog's clarity and efficiency makes it not only a tool for building expert systems, but a challenge to more popular products like Delphi and Visual Basic. It compiles to native executables and performance is impressive. ODBC is supported for database work. A particularly nice feature is the integrated help authoring system, which makes it easy to create and edit online help from within the development environment. Nevertheless, the unfamiliar language combined with lack of support for VBX or OCX components, or OLE in any form, will ensure that Visual Prolog remains a niche product. For projects which lend themselves to a Prolog implementation, though, Visual Prolog is mightily impressive.

Visual Basic

Trouble with menus

VB programmer Ian Moss writes with a menu problem. "*I can add and remove items from indexed menus, no problem. What I want to do is create menus that have submenus. I am adding menu items that are divisions of a basketball league. Each division has teams associated with it. I read the division names from a database, and create the correct number of menu items. I want each division menu to have a sub menu containing the teams in that division.*"

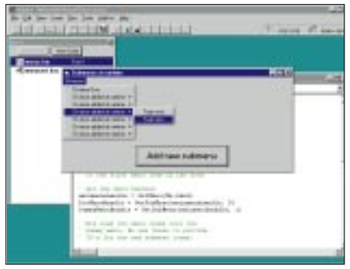
Here is a classic Visual Basic problem. Ian needs to create menus that have submenus, at runtime. VB's menu editor is a doddle to use. Creating menu items at runtime is easy using a control array and the Load command. But creating submenus at runtime is not in the book. It can be done, but only by trickery. It is another reason why serious VB programmers need Daniel Appleman's book, *Visual Basic Progammer's guide to the Windows API* (see review).

Using the Windows API, you can modify and add to the menus in a VB application.

Fig 1 VB Tip: Creating submenus at runtime

For this routine to work, you will need to use the VB menu editor to create a menu with two toplevel items. The first (Divisions) must have at least one sub-item, to make it a pop-up menu in API terms. The second (Dummy) must also have a sub-item, with an index value set, to make it a control array. I've named this mnuDummyArray. Finally, give the second toplevel item an empty string for a caption, and an enabled property of False. The user will never see the dummy menu.

Several API functions are included and these must be declared. The code below is for 32-bit Visual Basic, but with small amendments will work in 16-bit as well. Code to respond to clicks on the new menu items should be placed in the mnuDummyArray_Click event, using the Index parameter to detect which one was chosen.



You can create VB submenus at runtime, with a little help from the Windows API

```
Private Sub Command1 Click()

Dim mainmenuhandle As Long
Dim DivMenuHandle As Long
Dim DummyMenuHandle As Long
Dim NewMenuHandle As Long

Dim lRetVal As Long
Dim spareID As Long
Dim iCount As Integer

‘ this routine appends a new submenu
‘ to the first menu on the form

‘ get the menu handles
mainmenuhandle = GetMenu(Me.hwnd)
DivMenuHandle = GetSubMenu(mainmenuhandle, 0)
DummyMenuHandle = GetSubMenu(mainmenuhandle, 1)

‘ Now load two menu items into the
‘ dummy menu. We use these to provide
‘ ID’s for the new submenu items.

‘ count the existing items in the dummy submenu
iCount = GetMenuItemCount(DummyMenuHandle)

‘ load two new ones
Load Me!mnuDummyArray(iCount)
Me!mnuDummyArray(iCount).Caption = “Team one”

Load Me!mnuDummyArray(iCount + 1)
Me!mnuDummyArray(iCount + 1).Caption = “Team two”

‘ Create the new submenu
NewMenuHandle = CreatePopupMenu()

‘ Add two items to the submenu
spareID = GetMenuItemID(DummyMenuHandle, iCount)
lRetVal = AppendMenu(NewMenuHandle, MF ENABLED Or
MF STRING, spareID, “Team one”)

spareID = GetMenuItemID(DummyMenuHandle, iCount + 1)
lRetVal = AppendMenu(NewMenuHandle, MF ENABLED Or
MF STRING, spareID, “Team two”)

‘ Append the new submenu
lRetVal = AppendMenu(DivMenuHandle, MF ENABLED Or
MF POPUP, NewMenuHandle, “Division added at runtime”)

End Sub
```

p314 ➢

An entry with a submenu is not a normal menu item, but the top level of a pop-up menu, so you use the CreatePopupMenu function to return a handle to a new pop-up menu. Then AppendMenu is used both to add items to the submenu, and finally to add the pop-up menu to the existing menu structure.

The new menu will look pretty, but won't execute any code without further work. The problem is that VB creates menus with a two-stage process. When you design a menu, or change menu properties with VB code, you interact with an internal VB menu object. Visual Basic uses this internal object to generate the correct API calls that make the menu work. If you call the API directly, bypassing the internal VB object, VB doesn't know about the changes you have made.

When you click on a menu item, Windows sends a WM_COMMAND message to the application which includes a menu ID. This ID identifies the menu item, enabling the correct code to be executed. If you add a menu item using the API, VB will not recognise the ID, so the message sent when that item is clicked is ignored.

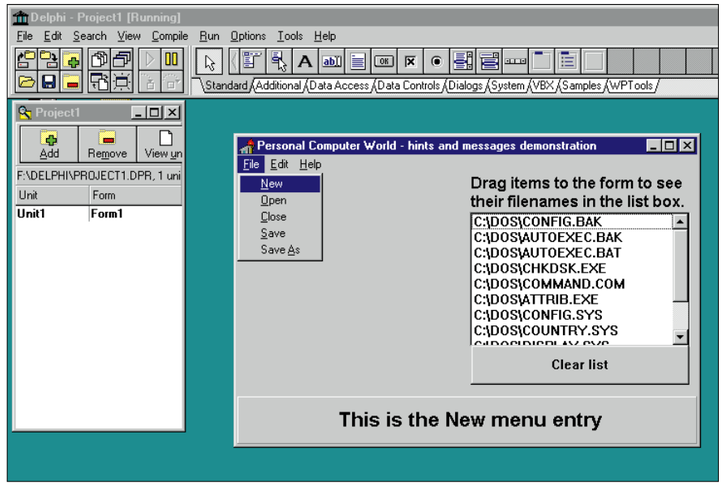
The workaround is to set up a dummy menu, where the toplevel item is disabled and has no caption, and which contains a control array. Note that the visible property must be True, otherwise the following tip will not work. When you need to add a menu item using the API, your code must first add an item to this control array. Then you can steal the ID for the new menu item, using the API call GetMenuItemID, and use it to create the new API menu item. When the user clicks on the menu you have created, VB is tricked into thinking that the item in the dummy menu has been selected, and will execute code in its click event. Fig 1 contains example code.

I must emphasise that this procedure is only necessary if you must create new submenus at runtime. Adding items to an existing menu or submenu is no problem. Another possibility is to create all your submenus at design time, and set their visible property to false, so that your code can reveal them as required. Finally, why not rethink the user interface completely? Ian's example application might be better served by an outline control, displaying divisions and teams in a tree view.

DELPHI

Delphi Gets the Message

One great thing about working with Delphi is how easy it is to trap Windows messages. Just to recap, much of Windows functionality is a result of system messages being sent to individual windows. For example, moving the mouse sends a WM_MOUSEMOVE message to a window.

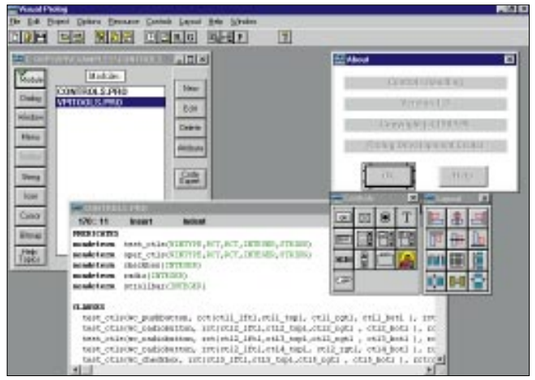


All done with messages: this Delphi application displays hints for the System menu, and accepts drag-and-drop files from Explorer or File Manager

Visual Basic Programmer's Guide to the Win32 API

Noted VB guru Daniel Appleman has issued an update to his popular API guide for Visual Basic users. This is no cursory update. The book has expanded by 500 pages and is more brick-like than ever. Even so, the author apologises for not including every Win32 API function. It is not his fault as the API is now so huge that to include everything would have made the book unmanageable. He correctly observes that once you know how the API ticks, it is not too hard to learn new functions and call them from VB.

Most serious Visual Basic developers will want this book. It accomplishes two things. First, it documents most API functions from a VB perspective, giving the correct declaration and explaining the particular benefits and pitfalls of each one. Second, there is masses on information on how Windows hangs together, including such topics as window handles, messages, co-ordinate systems and memory management. There's no other book like it, and Appleman does the job well.



Visual Prolog combines the Prolog language with a capable set of visual tools

I do have one nagging doubt, and that is why we need this book when Visual Basic should be powerful enough without it. The truth is, the deeper you get into the API from VB, the stronger the case for switching to a more suitable language such as C++ or Delphi. To get the best from VB, you need to be using it mostly within its natural limits, otherwise the benefits of RAD disappear under an avalanche of obscure code. The answer is to use this stuff with discretion, to solve problems that would otherwise leave you stalled. One example is optimisation. For instance, Appleman demonstrates a routine for searching listboxes that is five times faster than pure VB code. For users of your application that could make all the difference.

Sending, trapping, and creating custom messages are excellent techniques for creating powerful and flexible applications.

As an example, here's a couple of tips from Ian Briscoe (thanks for the tips, Ian - a book token is on its way). The first is for displaying hints for the System menu. The system menu does not appear in Delphi's menu editor, but you can still display hints by trapping the WM_MENUSELECT message. In the private section of a form declaration, add the following:

```
procedure SysMenuHint (var Message:
TWMMenuSelect); message
WM_MENUSELECT;
```

Note the message directive at the end of the declaration that tells Delphi this is a message handler. Fig 2 is the code for the procedure.

Ian adds, "Note that the menu selected is not the System menu. We call the inherited menu handler to allow Delphi to add its own hint functionality. We don't set the caption of the panel directly, but go through Delphi's own methods to display the hint, allowing you to still catch the OnHint event to add any additional coding."

The second tip is for trapping a drag-and-drop message from File Manager or Explorer. This is a neat trick that enables users to drag files into your application, for example to open documents in a text editor. First, add ShellAPI to the uses clause of the main form. Then declare the following message handler:

```
Procedure DragDrop (var Message:
TWMDropFiles); message
WM_DROPFILES;
```

Now in the FormCreate procedure add:

```
DragAcceptFiles(Form1.Handle,
True);
```

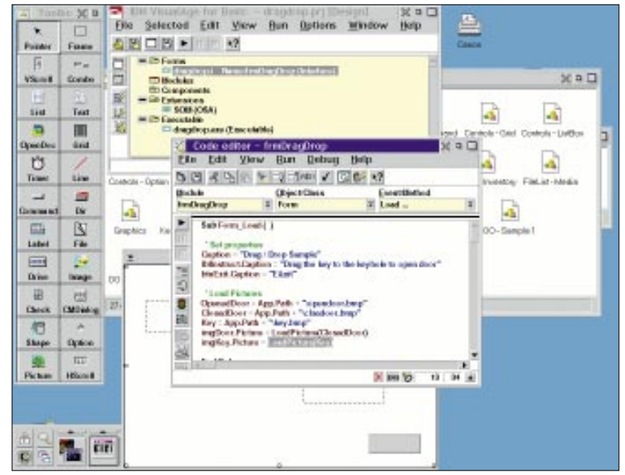
Fig 3 is the code for the DragDrop procedure. This example just displays the filenames in a listbox, but your code can do whatever you want with the files.

•PCW Contacts

Tim Anderson welcomes your Visual Programming comments and tips. He can be contacted at the usual *PCW* address or at **freer@cix.compulink.co.uk**, or **http://www.compulink.co.uk/~tim-anderson/**

Visual Basic Programmers' Guide to the Win32 API is by Daniel Appleman. ISBN 1-56276-287-7. £46.99. Contact Prentice Hall, Tel. 01442 881900

Visual Prolog costs £477 from PDC UK, Tel. 01603 611291



VisualAge Basic brings easy application development to OS/2 at last

Fig 2 Code for message handler

```
procedure TForm1.SysMenuHint (var Message: TWMMenuSelect);
begin
if (Message.MenuFlag and MF_SYSMENU) = MF_SYSMENU then
begin
case Message.IDItem of
0: Application.Hint := '';
SC_CLOSE:
Application.Hint := 'Closes the window and quits the application';
SC_MAXIMIZE:
Application.Hint := 'Expands the windows to fill the screen';
{...etc. Look up the constants in WINAPI.HLP under WM_SYSCOMMAND}
else
Application.Hint := '';
end;
Message.Result := 0;
end
else
inherited;
end;
```

Fig 3 DragDrop code

```
procedure TForm1.DragDrop(var Message: TWMDropFiles);
var
i, numfiles: integer;
lpzFileName: PChar;
begin
numfiles := DragQueryFile(Message.Drop,Word(-1),nil,0);
ListBox1.Items.BeginUpdate;
lpzFileName := strAlloc(101);
for i := 0 to numfiles do
    begin
    strPCopy(lpzFilename,'');
    DragQueryfile (Message.Drop,i,lpzFileName,100);
    ListBox1.Items.Add (StrPas(lpzFileName));
    end;
strDispose(lpzFileName);
ListBox1.Items.EndUpdate;
DragFinish(Message.Drop);
Message.Result := 0;
end;
```