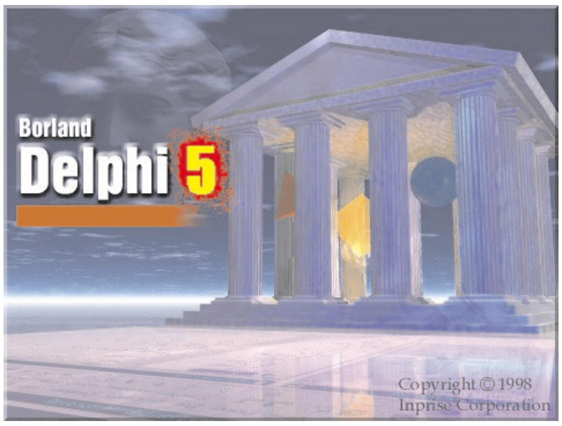
Borland Delphi 5

Reviewed by Brian Long

No doubt many Delphi developers will be hoping that this new version release from Inprise will be leaps and bounds ahead of Delphi 4 in the stability stakes. Well, the developers at Inprise are certainly very aware of the mistakes that were made with version 4 and have been striving very hard to ensure the same problems do not beset this new release.

As far as they are concerned, Delphi 4 went too far too soon, and so it was not possible to iron out all the stability issues by the time the product was released. With version 5 they are said to be focusing on stability rather than packing limitless new features into the box. That said, the set of features includes a number of things that make me very happy. The fact that Delphi 5 was originally scheduled for a June release and has been given an extra month or so in development and testing gives me hope.



Of course, as with other products of the same ilk, Delphi 5 is a superset of its predecessor. Version 5 has all that was supplied with Delphi 4 and more besides. For information about what was on offer in Delphi 4, check the review that appeared in *Developers Review* Issue 5, August 1998 (which you can also find on our website in the *Reviews Online* section). This review will focus on what is new. The information presented here is based upon pre-release software and so there may be some changes by the time the product gets into the box.

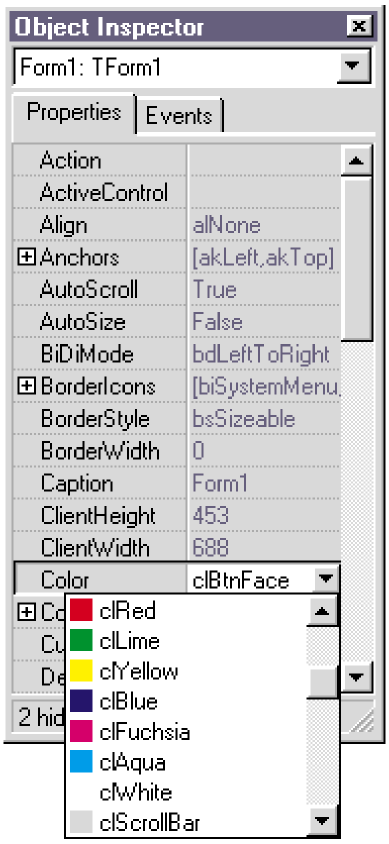
One point to get straight before we start is that the Java Byte Code compilation support that was demonstrated at the Inprise Developer's Conference in 1998 is not in Delphi 5, so if you were getting all excited, preparing to read all about the Delphi/Java marriage, well, sorry. What can I say? Maybe in the next version? On the other hand, maybe they decided it wasn't feasible, or maybe wasn't a good idea.

So what strikes you as you load up the new version? The IDE has had another facial, and in certain respects looks a little more like Visual Basic .

Object Inspector

The primary visual difference is with the Object Inspector. If you need to expand an expandable property, one click on the white-looking + will do it, rather than having to double click the property name. Also, more importantly, properties that have an obvious visual implication now show what they are. For example the `Cursor` property and the `Color` property (see Figure1).

Figure 1: Properties with visual implications are now clearer.



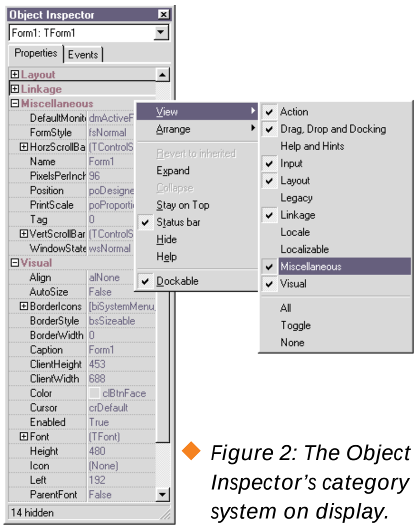
Action	
ActiveControl	
Align	alNone
⊞Anchors	[akLeft,akTop]
AutoScroll	True
AutoSize	False
BiDiMode	bdLeftToRight
⊞BorderIcons	[biSystemMenu,
BorderStyle	bsSizeable
BorderWidth	0
Caption	Form1
ClientHeight	453
ClientWidth	688
Color	clBtnFace

Another welcome arrival is the concept of property and event *categories*. Delphi 4's component set meant that, for most components, there were a lot of properties and events on the Object Inspector, many of which the average developer would never touch. Now, properties and events are categorised and you can elect to hide or display properties from whichever categories you choose. You can also ask the Object Inspector to list the categories out as expandable top-level entries. This means that you can omit all the irrelevant properties that camouflage the important ones that you tend to work with. If any properties are hidden, the Object Inspector's status bar tells you how many, as shown at the bottom of Figure 2. Of course, you can disable category display if you wish, as was done for Figure 1.

All this means a richer development environment for those creating components with customised property editors, as they can now ensure the properties are displayed in whatever fashion makes most sense. Of course, it also means the Object Inspector is a little less overwhelming, and a little friendlier.

Forms And The Form Designer

To help version control systems deal with forms in a more simplistic manner, the form designer now stores



DefaultMonit	dmActiveF
FormStyle	fsNormal
HorzScrollBa	(TControlS
Name	Form1
PixelsPerInch	96
Position	poDesigne
PrintScale	poProporti
Tag	0
VertScrollBar	(TControlS
WindowState	wsNormal
Align	alNone
AutoSize	False
BorderIcons	[biSystemMenu,
BorderStyle	bsSizeable
BorderWidth	0
Caption	Form1
ClientHeight	453
ClientWidth	688
Color	clBtnFace
Cursor	crDefault
Enabled	True
Font	(TFont)
Height	480
Icon	(None)
Left	192
ParentFont	False

Figure 2: The Object Inspector's category system on display.

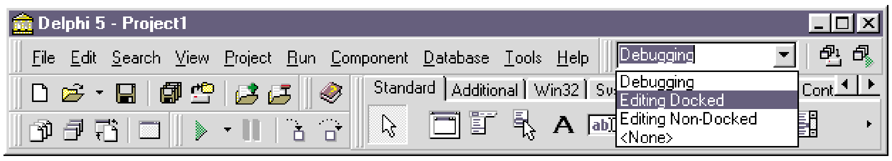
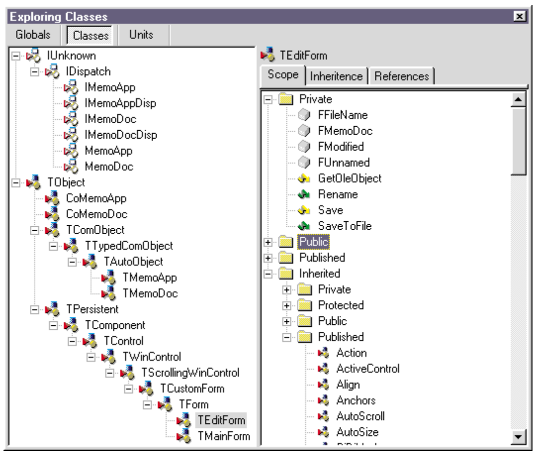


Figure 3: You can choose customised global desktop layouts.

forms in text format. In other words, .DFM files are textual by default. To store a form in binary, right click on it and uncheck the Text DFM item. To store forms as binary by default, the Preferences page of the Environment Options... dialog has a New forms as text checkbox, which can be unchecked. The VCL's support for translating between binary .DFM files and their textual equivalent has been extended to support this.

Another form-related change is another option on the same dialog page. Auto create forms is checked by default. This matches the behaviour of earlier versions of Delphi, where each new form is auto-created by default. You can uncheck this option to cause any new forms to not be auto-created, meaning you have to take charge of creating the relevant form instance when the program requires access to a form. If a form should be auto-created, but this option is not set, you can still use the Forms page of the Project Options dialog to toggle this.

Figure 4: Using the Project Browser.



Mention should be made of one other change in the IDE that will prove very popular and, frankly, is well overdue. At last, dialogs that require directories to be entered give you the option of browsing to the relevant directory. No longer will you have to type in colossal path names and simply hope you got them right *[or cut and paste them from a DOS box command line prompt! Ed]*.

Desktops

A great new feature is the concept of *global desktops* (as opposed to project desktops). Once you have organised the IDE windows the way you want them, you can store them as a new desktop through an entry on the View menu. The information stored in a desktop includes which windows are visible, where they are, and who they may be docked with. Also, the selected page of the Object Inspector, and available property categories are recorded, along with the entire state of all the speedbars.

To load up a specific desktop layout, go to the View menu again, or better still, use the combobox on the new Desktops speedbar on the main Delphi window (see Figure 3). When you need to do some debugging, you can switch from your normal desktop to the debugging desktop (which could well have all the standard debugging windows docked into the maximised editor, for example).

In fact, out of all the desktops you set up (which are stored as .DST files in the BIN directory) you can specify which one is your debugging desktop. Then, as soon as you start debugging a project, the debugging desktop is loaded.

Project Manager

The Project Manager has had a few changes. When working with a project group, you can switch between projects using the drop down selector at the top of the Project Manager. You can also now copy and paste files between projects. Additionally, files can be dragged into projects from Windows Explorer.

Resource script files (.RC files) can be added to a project and will be compiled to .RES files from within the IDE, rather than requiring external compilation as before.

Project Browser

The old Object Browser, (practically untouched since Delphi 1) has had a facelift. Figure 4 shows it displaying one of the sample projects. For starters, if the project has not been compiled, the Project Browser (as it is now called) will compile it when it is invoked. The Object Browser simply refused to make itself available. You can choose whether the browser will work just on symbols from within your project, or across the whole VCL. You can also customise the way the items within a given class or are displayed in the tree view on the right (or indeed whether the tree view details section is displayed at all).

Double clicking on a symbol in this new browser opens a new window, with tabs available as appropriate. If you double click a class, you get scope (what's inside it) and references (which source lines refer to it) pages, whereas double clicking a class member gives just a references page.

Figure 4 shows that you can still browse through global symbols and units as well as classes. One nice addition to the unit browser is information on which units the selected one is used by.

Code Editor

Delphi 4 transferred the debugger options from Tools | Environment Options... to the new Debugger Options... dialog under the Tools menu. Delphi 5 does the same thing with the large number of editor options. Tools | Editor Options... produces a five page dialog (also produced by right clicking in the editor and choosing Properties). Four of these pages will be familiar to Delphi 3 and 4 users, but the available editor keystroke mappings have now been given their own page.

A notable addition to the list of keystroke mappings is Visual Studio emulation (recently added to Borland C++Builder 4). Also in the list is New IDE Classic. This is a custom keystroke mapping, implemented in an editor enhancement module. The Open Tools API has been extended to include editor enhancement modules, which either can provide a complete keystroke mapping, or can customise individual keystrokes (independent of active keystroke mapping).

A sample enhancement module project is supplied, and indeed is pre-installed into Delphi 5. This demo does two things. Firstly, it adds a new keystroke mapping called New IDE Classic. It also customises Ctrl+B to invoke a buffer list (in any keystroke mapping).

The Language And VCL

As a novelty, the underlying Delphi language has not really been added to or changed at all this release. The only thing worth mentioning is that the System unit contains a few LOCK machine instructions to ensure that reference-counted strings and dynamic arrays are now thread-safe when being read from and when used as parameters.

To match the release version, the Visual Component Library now reaches version 5 (the compiler itself is version 13, and so we have a VER130 conditional define available). New things to look out for include the following.

The well known problem of the main form not animating when minimised or restored has been fixed. Because of the internal make-up of a VCL application, Delphi 2, 3 and 4 simply hid all forms when the main form was minimised. The actual task bar icon was the minimised version of the Application object's window. This all meant that, unlike most Windows applications, you did not see the standard animation of the main form minimising itself to the task bar. This long term irritation has now, thankfully, been fixed.

Events of the Application object can be set up automatically using the new TApplicationEvents component. This surfaces all the events of the Application object and you can therefore make the Object Inspector responsible for implementing the basic event handler skeletons. Each form can have an ApplicationEvents component placed on it, and when the relevant event of Application is triggered, the corresponding event of each ApplicationEvents executes in turn.

You can ensure one particular ApplicationEvents executes its event before any other by calling its Activate method. In fact, it is expected that the OnActivate event handler for a form will call that form's ApplicationEventsActivate method, to make it the first one. To stop any more ApplicationEvents components executing their event handlers, the component has a CancelDispatch method.

The TAppletApplication class has been introduced to allow you to make a Control Panel project (.CPL file) with as many Control Panel applets (TAppletModules) inside as you like. You can access these classes from the File | New... dialog, and so can add many extra items into the Control Panel from either one or more .CPL files.

There are facilities for doing custom drawing in a TToolBar, and reordering the headers in a THeaderControl. The TListView class has had some work done to it. It now supports work areas, images for sub-items when set to report style, and InfoTip (or help hint) support on all items in the list view.

There are more standard actions to add into your action lists, including more edit-related actions (delete, undo, select all) along with some help actions (contents, help on help and topic search).

When writing web applications, web action items have a Producer property so you can have the content of response messages automatically updated when the action item executes.

Inside the new unit Contnrs unit, you will find a number of useful container classes. TObjectList, TComponentList and TClassList are based upon the normal TList, with a few extra methods and properties. Basically, these are type-safe wrappers around TList to help you easily store

No really, we're confident the users are going to love it.

object references (`TObjectList`), component references (`TComponentList`) and class references (`TClassList`). `TObjectList` can be told to own the objects place in it, and `TComponentList` can pick up on components being destroyed and update itself.

The `TStack` class is also based upon a `TList`, with `TObjectStack` inheriting from it, and then `TQueue` is a descendant of `TStack`, with `TObjectQueue` inheriting from it.

One minor routine that has been added is called `FreeAndNil`. This takes an object reference variable as its sole parameter, destroys the object and resets the reference to `nil`. So what historically has involved two statements can now be reduced to one procedure call.

Frames

There have been techniques available for years that allow you to design a form and then embed that form within another, maybe as a page in a page control, for example. These days, this involves constructing the form using the `CreateParented` constructor, which means having to do things programmatically.

For an alternative and potentially easier way to re-use forms within other forms, we can now design things called frames. A frame could be considered a reusable section of a form, designed in isolation. You make one with `File | New Frame`, or by choosing a `Frame` from the `File | New...` dialog. This allows you to design a `TFrame` descendant much as you would design a form.

When you want to use one of your frames, you pick it from the component palette like any other real component. The `Frames` 'component' can be found on the `Standard` page of the component palette. Unlike component

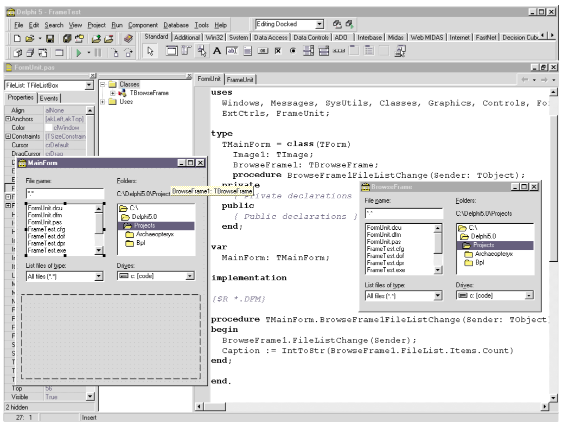


Figure 5: Frames can be embedded into forms: easy compound components.

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Fo
  ExtCtrls, FrameUnit;

type
  TMainForm = class(TForm)
    Image1: TImage;
    BrowseFrame1: TBrowseFrame;
    procedure BrowseFrame1FileListChange(Sender: TObject);
  private
    { Private declarations
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.BrowseFrame1FileListChange(Sender: TObject
begin
  BrowseFrame1.FileListChange(Sender);
  Caption := IntToStr(BrowseFrame1.FileList.Items.Count)
end;

end.
```

templates (a recorded collection of related objects that are duplicated when used), frames are not individually listed on their own palette page. Instead, when you pick the `Frames` component, you get a list of the available frames in this project. As soon as you choose one, the frame is embedded in your form. You can then manipulate the frame itself, or the objects within the frame.

A frame represents a convenient way to create proper compound components, components made from many other components. Just place the constituent components on a frame, set their properties, implement their event handlers, and away you go. When the frame is placed on a form, each individual constituent component can be customised, moved etc, but not deleted (you get a rejection error message if you try).

When you place a frame on a form, you are creating an instance of the underlying frame class type. This means that any changes to the original frame are immediately reflected on the form. In addition, just as with form inheritance, any property changes made to the frame instance will override the properties of the original. You can extend the functionality of the frame's components by making event handlers for them on the form. These event handlers default to initially calling the corresponding method from the frame class. You can either add extra code, or replace this call with your own code.

It's difficult to show this concept without sitting in front of you and going through an example, but Figure 5 shows a directory browsing frame having been developed and subsequently used on an image browsing form. The file list on the frame has an event handler hooked up to its `OnChange` event. The file list (from the frame) on the form also has an `OnChange` event handler set up. You should be able to see the call to the original handler in the editor.

To-Do List

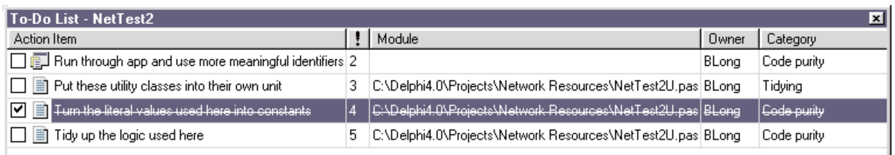
Delphi now supports project to-do lists. To-do list items can be project-wide, or can relate to an individual source file. You can view the to-do list by choosing `View | To-Do List` and it pops up a window which, like most of the other IDE windows, can be docked into the editor (see Figure 6).

New items for the to-do list can be added directly into the window, and are stored in a .TODO file with the same name as your project. However, you can also add entries by right clicking and choosing `Add To-Do Item` (`Shift+Ctrl+T`). This gives a dialog, which ultimately inserts a special comment into your source file. The comment specifies whether the item is still to do, or done. It also indicates a priority between 1 and 5, the owner, category and text of the item, for example:

```
{ TODO 1 -oBLong -cStop-ship :
  Fix the Access Violation
  problem in this routine }
```

This indicates a high-priority to-do item entered by `BLong` in the `Stop-ship` category, relating to an

Figure 6: A Delphi 4 project loaded into Delphi 5 with some things still to do.

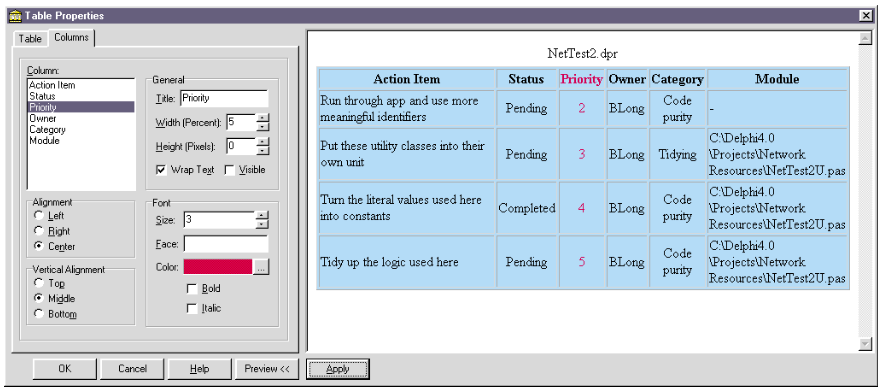

To-Do List - NetTest2

Action Item	!	Module	Owner	Category
☐ Run through app and use more meaningful identifiers	2		BLong	Code purity
☐ Put these utility classes into their own unit	3	C:\Delphi4.0\Projects\Network Resources\NetTest2U.pas	BLong	Tidying
☑ ~~Turn the literal values used here into constants~~	4	~~C:\Delphi4.0\Projects\Network Resources\NetTest2U.pas~~	~~BLong~~	~~Code purity~~
☐ Tidy up the logic used here	5	C:\Delphi4.0\Projects\Network Resources\NetTest2U.pas	BLong	Code purity

Access Violation. As soon as you type one of these to-do comments into your source, it makes its way straight into the to-do list. When a to-do item has been completed, the window draws a line through it to highlight this fact. In addition, if you have a to-do item in a file that is part of the current project, but which is not open, it will be drawn grey in the list.

Double clicking a source code to-do item in this window takes you to the comment in the relevant source file. The list distinguishes between project-based to-do items and source-based to-do items by using different bitmaps in the list window (again, see Figure 6). The To-do list can be sorted by any column, by clicking on the column header, or using the right click menu.

You can copy the To-do list onto the clipboard in either standard text format, or as an HTML table, whose various attributes you can customise (Figure 7).



Title: Priority
Width (Percent): 5
Height (Pixels): 0
[✓ Wrap Text] Visible
Alignment: [✓ Center]
Vertical Alignment: [✓ Middle]
Size: 3
Face:
Color:
Bold
Italic

NetTest2.dpr

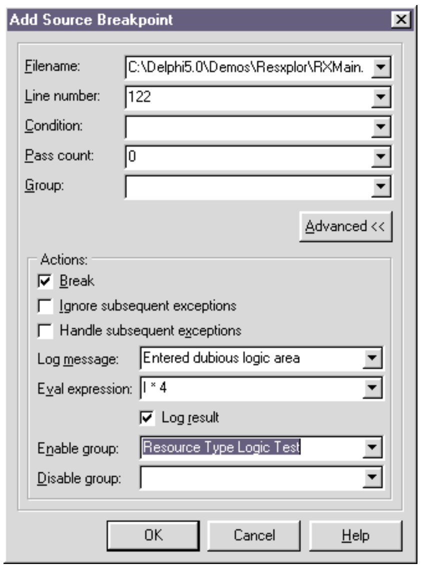
Action Item	Status	Priority	Owner	Category	Module
Run through app and use more meaningful identifiers	Pending	2	BLong	Code purity	-
Put these utility classes into their own unit	Pending	3	BLong	Tidying	C:\Delphi4.0 \Projects\Network Resources\NetTest2U.pas
Turn the literal values used here into constants	Completed	4	BLong	Code purity	C:\Delphi4.0 \Projects\Network Resources\NetTest2U.pas
Tidy up the logic used here	Pending	5	BLong	Code purity	C:\Delphi4.0 \Projects\Network Resources\NetTest2U.pas

Figure 7: A to-do list exported to HTML.

Debugging

To accompany the CPU window view of the program's machine code, stack, memory and CPU registers, you now have the option of an FPU window. This shows the state of the floating-point unit and displays either floating-point information, or MMX information.

Figure 8: Breakpoints don't have to 'break' the program.



Filename: C:\Delphi5.0\Demos\Resxplor\RXMain.
Line number: 122
Condition:
Pass count: 0
Group:
[✓ Break]
Ignore subsequent exceptions
Handle subsequent exceptions
Log message: Entered dubious logic area
Eval expression: I * 4
[✓ Log result]
Enable group: Resource Type Logic Test
Disable group:

The IDE supports some drag and drop operations during a debugging session. You can drag any expression from the editor into a watch list. This adds a watch on that expression. You can also drag an expression to an inspector, to conveniently inspect it. Finally, you can drag an expression to the dump or stack pane of the CPU window, which causes them to locate to the address of the expression.

The Run menu has an extra item with respect to program stepping, called `Run Until Return` (`Shift+F8`) which will execute a subroutine and stop when it returns to its caller. This can be very useful when you step into a long routine by accident, having intended to step over it.

Another entry on the Run menu is `Attach to Process...`, which was available in Delphi 4 through an undocumented registry entry. This allows you to attach to an externally running application and then proceed with debugging it.

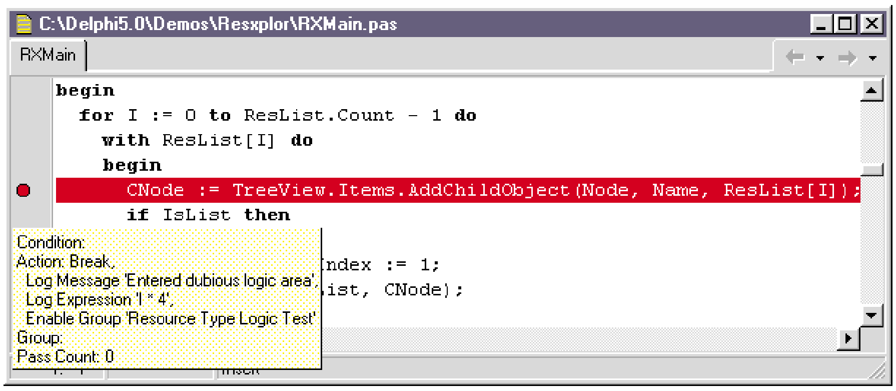
The `Evaluate/Modify` dialog now has extra convenience speedbuttons that allow you to take the current expression being evaluated and either make a watch expression from it, or load it into an inspector. Another watch-related convenience is the ability to ask the watch window to launch a debug inspector for the highlighted watch expression.

When using the modules window to see the modules loaded into the current process address space, if a module is selected, all its entry points can now be listed either by name or by address.

The breakpoint properties dialog (Figure 8) has more options now, thanks to a new Advanced area. You can tell the breakpoint not to break, but to simply log itself in the event log. Additionally you can have an expression evaluated every time the breakpoint is hit which might be to use handy if you have subroutines that produce side-effects, and you can elect to log the value of an expression as well.

Breakpoints can now also be allocated a group, and the action of one

Figure 9: Breakpoint details visible in a tooltip.



```
begin
  for I := 0 to ResList.Count - 1 do
    with ResList[I] do
    begin
      CNode := TreeView.Items.AddChildObject(Node, Name, ResList[I]);
      if IsList then
```

```
ndex := 1;
ist, CNode);
```

Condition:
Action: Break,
Log Message 'Entered dubious logic area',
Log Expression 'I * 4',
Enable Group 'Resource Type Logic Test'
Group:
Pass Count: 0

breakpoint can be to enable or disable a whole group of other breakpoints.

Breakpoint properties are now available by right-clicking the red circle that represents the breakpoint in the editor's left-hand gutter region. Also, pausing your mouse over the circle will show you the main properties in a tooltip (Figure 9).

In the debugger options dialog, there are new checkboxes to allow you to debug any processes launched by your application, to allow side effects in the evaluation of expressions in the watch window, and to disable the automatic use of multiple evaluators. When debugging multi-process applications, which may be written in different languages (say C++ and Delphi), Delphi will, by default, use the appropriate expression evaluator for the language used to build the program currently under the eyes of the debugger.

There are additional entries in the list of exceptions that you can choose Delphi to ignore. These include exceptions coming from Microsoft's ADO libraries, internal VisiBroker exceptions along with CORBA system and user exceptions.

Whilst debugging, particularly with multi-process debugging, you can right click on a process in the thread window and set temporary debugging options for that individual process with a copy of the debugger options dialog.

COM/ActiveX

From the `ActiveX` tab of `File | New...` you can now make an Active Server object. This is an Automation object that can be accessed from an Active Server Page (ASP) on an IIS Web server and which can access the various interfaces representing the user request, response, server and so on. When you ask for an Active Server object, the wizard offers you the option of having a simple ASP test script generated.

When asking Delphi 4 to make a new (non-Automation) COM object with type library support in an application, the supplied Wizard would happily oblige. However, the generated type library OLE Automation flag would be unchecked by default. Now this may sound sensible, since you are not asking for an Automation object, but wasn't. Basically the idea is that if that flag is not set, it is down to the COM programmer to write all the marshaling code to get data in and out of the COM server, not a pleasant task at all.

Delphi 4 users found that client applications would report being unable to talk to the COM object's interfaces, getting the error: *Interface not supported*. Not really knowing what the problem was, they would give up, or use a more heavyweight Automation object instead. Delphi 5's COM object wizard helps out by asking if you want this option checked, and defaults to doing so. It also now defaults to offering type library support, which you can uncheck if you want.

Another change is that when developing COM/Automation objects and the type library editor wants to refresh the underlying implementation source, you get an updates dialog showing you what is about to be done. This gives you the opportunity to veto some of the proposed changes (see Figure 10).

When importing type libraries in Delphi 5, you now have the opportunity of installing COM servers as components onto the component palette. The component will surface any Automation events of the server onto the Events page of the Object Inspector, spectacularly simplifying the job of responding to server events. All the available properties of the server will be available at runtime, though they are not listed on the Object Inspector.

This componentising of Automation servers is a good progression of the limited event support added in Delphi 4. A whole bunch of Microsoft Office servers are pre-installed on the `Servers` page of the component palette making Office Automation very easy.

Visual Basic can manufacture Active Servers and Active Controls with sparse vtables, ie vtables with gaps in. These cause problems for Delphi 3 and 4, but Delphi 5 can correctly deal with the situation. If a sparse vtable is detected, the type library importer will add dummy entries in the interface definition in the import unit.

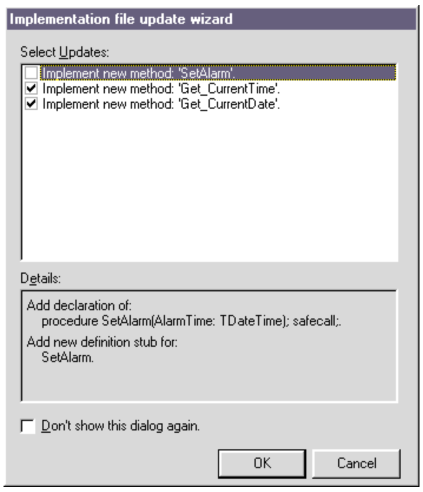
The are some minor tweaks too. The COM object factory class (`TComObjectFactory`) has been amended to correctly set the little-known global `CoInitFlags` variable for correct COM free threading support, without help from the programmer. COM servers also cater sensibly when trying to register themselves in the Windows registry, but without sufficient access rights. Finally, the `HResult` COM error type has now changed back from an unsigned integer to a signed integer, as it was in Delphi 3.

CORBA

CORBA developers awaiting the inclusion of an IDL to Pascal translation tool will be initially disappointed by its apparent lack of existence. So still, with the product supplied as is, you can only access non-Delphi CORBA servers using slow late-bound Dynamic Interface Invocation after registering the IDL file in an Interface Repository, or by writing the Pascal stub class by hand.

However, if you need to access non-Delphi servers all is not lost, as the Delphi 5 feature set is actually said to include IDL2PAS. How do these two, apparently contradictory, statements pan out? Well, current information

Figure 10: Type Library Editor refresh can be customised.


Implementation file update wizard

Select Updates:

- [ ] Implement new method: 'SetAlarm'.
- [✓] Implement new method: 'Get_CurrentTime'.
- [✓] Implement new method: 'Get_CurrentDate'.

Details:

Add declaration of:
procedure SetAlarm(AlarmTime: TDateTime); safecall;.
Add new definition stub for:
SetAlarm.

[ ] Don't show this dialog again.

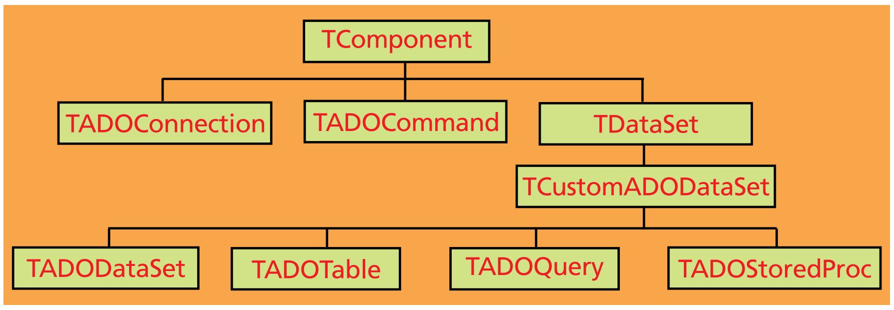


Figure 11: The ADO support component hierarchy.

suggests that IDL2PAS (or IDL2Delphi, or whatever it gets called) will be a downloadable part of the product, available from the Inprise website.

This decision seems based on the fact that Inprise are anticipating making various improvements and revisions over the coming months and want to make sure they have a good avenue of making the updated versions always available. I would imagine also that it is taking rather more work than anticipated, and they have no desire to hold up release of the main product, simply because one small part of it is not quite ready.

The primary in-the-box change with CORBA support is that Delphi 5 ships with VisiBroker 3.32 (as shipped with C++Builder 4). However, network message traffic is lessened by the client no longer pinging the server to maintain a connection.

Internet

The `Internet` page on Delphi 4's component palette had 26 components on it. Delphi 5 shrinks this to just 8. The NetMasters FastNet suite of components now has its own `FastNet` palette page. However, their `THTML` component has been removed in favour of `TWebBrowser`, which is a wrapper around the Internet Explorer ActiveX control (found on the Internet page).

For more information on internet support, see the InternetExpress section later.

Database Support

ADO exceptions were mentioned just above. To quell all those dissident developers who find the BDE to be too much of an archaic monolith, the Delphi developers have bowed to pressure and introduced some native ADO dataset components. These components communicate directly with Microsoft's ADO (ActiveX Data Objects) libraries, bypassing the BDE completely. These ADO libraries can be installed from the Delphi CD in case you do not already have them on your machine.

The hierarchy of new ADO components can be seen in Figure 11, and the dataset components will connect quite happily to data source components and so populate normal Delphi data-aware controls.

The `TADOTable`, `TADOQuery` and `TADOStoredProc` components are plug-in ADO-ready versions of the BDE `TTable`, `TQuery` and `TStoredProc` components. The `TADODataSet` is a more generalised object that uses an ADO RecordSet. It can be used in conjunction with the `TADOCommand` for specialised SQL execution. The `TADOConnection` represents a connection to an ADO database (rather like a BDE `TDataBase` component).

Additionally, since ADO knows about certain specialised field types, the VCL has the new field components shown in Table 1.

These new dataset components have not been designed for effortless migration of old BDE dataset code. For example, to re-execute a query with a `TQuery` object, you would call the query's `Close` and `Open` methods. With `TADOQuery`, you simply call the `Requery` method.

With regard to the more traditional BDE dataset components, we have a few changes here as well. The `TDatabase` component has an `Execute` method to allow an SQL expression to be executed without using a separate `TQuery` component. This method has a number of optional parameters defined. These allow you to pass in SQL parameters, cache the query for efficient re-execution in the current transaction, and obtain a cursor to the result set (so you can give it to an existing `TTable`). The method returns the number of records affected by the execution of the supplied SQL.

A client dataset has a published `Constraints` property, for record-level constraint enforcement. Also, it will enforce field level constraints set up by `TField` or `TCheckConstraint` objects.

Data Module Designer

Since its introduction in Delphi 2, the data module designer has looked very dull and simple, like a form designer with a white background. A lot of attention has been paid to it for this new version. It has been turned into a visual design tool that makes it easy to create, document and maintain data modules. The data module designer has three areas.

The components view is the same as the normal data module designer in previous versions.

The tree view shows the hierarchical view of the data module's components, such as that between a table and

Table 1: New ADO field classes.

Field Class	ADO Field Description
`TWideStringField`	Fields holding strings of 16-bit Unicode characters
`TGuidField`	Fields storing **G**lobally **U**nique **ID**entifiers
`TVariantField`	For Variant fields
`TInterfaceField`	Fields holding interface references
`TIDispatchField`	Fields holding Automation object references

its fields, or between a session and a database. Components can be dragged to change their interrelationships, for example a datasource could be dragged from one table to another. You can also add items from the component palette directly into the tree view.

The Data Diagram view is basically a documentation tool that shows relationships between components on the data module. To set up a relationship you can drag items from the tree view, and then use special icons to describe the relationships. If you set up, say, a master/detail relationship on the Data Diagram view, then all the pertinent component properties will be set. You can set lookup relationships and examine properties of linked components. Also, you can add in comment blocks and define allude relationships (a real world or commentary linkage between two things), which has no effect on any properties. All this extra information is saved in a `.DTI` file.

Figure 12 shows a Data Module designer displaying a few master/detail relationships.

InterBase Express

In addition to teeChart, QuickReport, FastNet and the InterBase EventAlerter component, Inprise have now licensed and updated Free IB Components. These direct access InterBase components (in other words, like the ADO components, they do not use the BDE) are now referred to as InterBase Express (IBX). They can be found on their very own InterBase page of the component palette and are described as working with InterBase 5.5 and higher.

MIDAS

The MIDAS technology allows middle-tier application servers to communicate with thin clients around the network, thereby allowing the development of distributed applications. The connection between the client and server applications can be through many protocols.

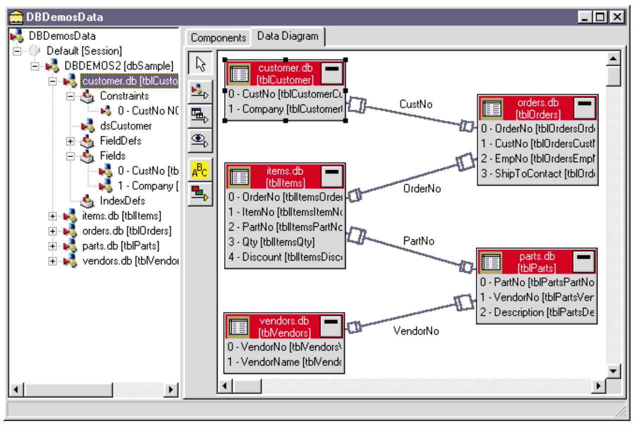
In addition to the existing connection objects that support COM, DCOM, OLEnterprise, CORBA and sockets, Delphi 5 introduces the `TWebConnection` component to allow an HTTP connection to be used. This means that it can be used to connect through a firewall, and can also use SSL security. It exposes properties for setting up the connection, including `UserName`, `Password`, `URL`, `Host`, `Port` and `Proxy`. Remote data accessed through an HTTP connection can take advantage of resource pooling.

MIDAS 3, as supplied with Delphi 5, now directly supports stateless remote data modules. This means that you can share remote data modules in MIDAS servers running within MTS (Microsoft Transaction Server) without writing a whole lot of support code to conform to the MTS requirements. Unfortunately this means some architectural changes in the MIDAS code, which ultimately will mean some changes to your old MIDAS applications. For example, the `IProvider` interface has been removed and changed to the `IAppServer` interface. Correspondingly the client dataset `Provider` property has gone. There are a number of other changes that will need to be borne in mind, but on a brighter note there are lots of new `BeforeXXX` and `AfterXXX` events in the client dataset and provider objects. These allow custom data to be sent with calls to the server.

InternetExpress And XML

The InternetExpress page of the component palette contains components that enable you to write web server applications that are MIDAS clients. The components include a `TXMLBroker` and a `TMIDASPageProducer`. An appropriately constructed web server application can request from the MIDAS application server that data packets are encoded as XML instead of `Variant` byte arrays. These XML data packets can be picked up and interpreted by new JavaScript libraries (also supplied with Delphi 5) to write a thin client application that will run in a JavaScript-aware web browser.

You can also use the InternetExpress components to build non-MIDAS web server applications, by including the provider and its dataset into the web server application itself.



◆ *Figure 12: The new data module designer.*

Team Development

Delphi has been able to hook up to version control systems since version 1. The Open Tools API allows DLLs to be written that interface between the Delphi IDE and some version control system. All versions of Delphi have shipped with a DLL to allow you to work with Intersolv's PVCS.

Delphi now ships with a team development tool of its own, called TeamSource. This is not so much a version control system as a workflow management tool. It has been designed to help development teams manage their daily tasks within a shared development environment.

Many version control system tools work on the basis of allowing one developer to edit a file at any time, by locking it and unlocking it, which is the serial source control model. There is also a parallel source control model, where each developer works on a local copy of shared files. When a developer finishes work on a file, their changes must be reconciled with those of other developers. This means comparing and spotting differences between the local version and the master copy of the file and then synchronising the two. After this reconciliation, the local copy reflects accepted changes in the master copy, and the master copy reflects local changes that were applied.

TeamSource goes beyond simple file version control by managing a parallel model of source control. TeamSource actually uses a version control system to store and retrieve shared files. Its predefined support is for PVCS (which you need to buy) and the simplistic archiver, ZLib (which is free in the box). Support for other tools can be introduced by writing custom TeamSource Extensions (a DLL with a `.TSX` file extension).

Delphi 5: A Second Opinion

by Dave Jewell, Technical Editor

To my mind, Delphi 3 represented the last major release of this development system. The huge amount of under-the-hood change required by the introduction of packages meant that Delphi 3 was a radically different animal to its predecessor (remember *CMPLIB32.DCL*? – if you don't, then be glad). Since that time, Delphi 4 and 5 have progressively built on the Delphi 3 foundation, adding large numbers of relatively small 'tweaks' and functionality enhancements.

Of course, this isn't to belittle the recent work that Borland have done, there are many welcome new features in this release, but I'm especially delighted to hear that stability is the key emphasis of Delphi 5! I'm particularly enthusiastic about the new native ADO components for those who prefer to dispense with the BDE, the host of Office Automation components that come pre-installed on the Delphi 5 component palette and the new Frames facility mentioned elsewhere by Brian. Truth to tell, I didn't embrace Delphi 4 with anything like the enthusiasm that Borland might have wished, and I don't believe I was alone. Up until now, Delphi 3 has remained my *de facto* development system, but with the launch of Delphi 5, I can certainly see plenty of compelling reasons for an upgrade.

The new Object Inspector certainly has a cleaner 'look' to it. The categorisation changes are a step in the right direction, but I feel that they don't go far enough and I think it's rather confusing having the same properties appearing in multiple categories. For example a form's, `Left`, `Top`, `Width`, and `Height` properties all appear in the `Localizable`, `Layout` and `Visual` categories. I can see why they should, but I don't like the fact that they do! Personally, I would have preferred a more flexible scheme, allowing the developer to create his or her own categories. I don't believe this would be difficult to achieve.

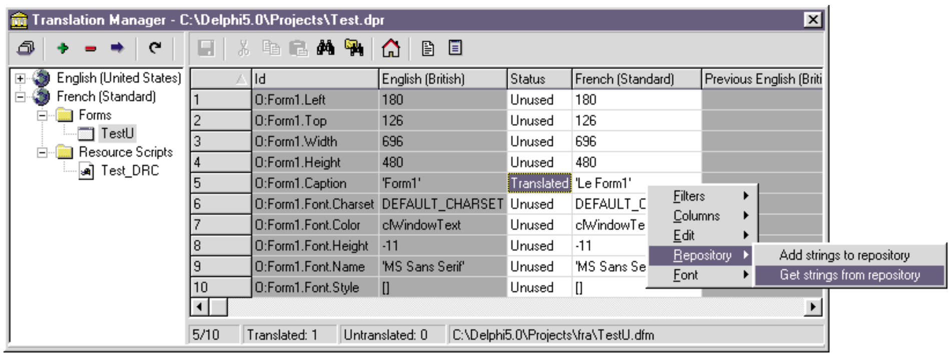
While discussing possible enhancements to Delphi in a recent interchange on CIX, I commented that it would be nice to see more source code to the built-in property editors (this is especially useful for those developers who wish to create enhanced versions of existing property editors). I was therefore delighted to see that Borland have now included the source code to `PicEdit` and `StrEdit` with this release of Delphi. It's a small start, but it's a start!

In writing this *Second Opinion*, the Editor was keen for me to comment on how Delphi 5 stacks up against other popular development systems. Well, it should hardly need stating, but I consider that in terms of ease of use and programmer productivity, Delphi remains head and shoulders above Microsoft's resolutely non-visual Visual C++. Delphi also wins hands down against Visual Basic, and always will do until Microsoft gets serious about adding pointer support to the language, or else creates a decent application framework for VB that wraps the underlying Windows API. If you don't know what I'm talking about here, just try implementing an owner-draw listbox in a VB application and you will soon see how quickly Visual Basic runs out of steam. No cheating with third-party DLLs or OCX controls is allowed!

Of course, there is one big chink in Delphi's armour, and it's something that I've mentioned in the past. Just as Microsoft seem determined to keep Visual C++ as non-visual as possible, Borland likewise could do a lot more in the area of ActiveX control creation. Rather than simply providing a conversion process for existing VCL components, I'd like to see something much more like the Visual Basic approach. I'd like to be able to visually lay out a new ActiveX control *from scratch*, perhaps making it up from an aggregation of existing OCX or VCL controls. This is one area in which VB scores heavily over Delphi. If Delphi could combine the RAD-based ActiveX creation of VB with the lightweight ATL technology of Visual C++, then you'd have a matchless development environment.

Finally, unlike Brian, I'm not disappointed regarding the lack of Java byte code compilation in Delphi 5. In fact, I'm deliriously happy at its absence! From my perspective, I believe that the sun is beginning to set on the Java phenomenon. The dust is settling, the hype is being seen for what it was, hype, and an increasing number of developers are waking up to the fact that the only way of getting decent runtime performance for portable applications is to use native code compilers and traditional cross-platform development tools. Quelle surprise! (see my coverage of Qt and XVT in this month's *Platforms Column*). Delphi is, indisputably, the best development system for general purpose Windows application development. Let's keep it that way, without turning it into some Jack-of-all-trades, master-of-none monstrosity. If Borland have got any sense (and I suspect they've got a lot more now than they had six months ago) then they'll be thinking very seriously about a Linux port of Delphi... which actually seems to be *just* what they *are* doing right now!

TeamSource has been used, in various guises and revisions, by the Borland development teams for over four years now. It has been used in teams from just a couple of developers as well as in teams of up to fifty developers, and so has stood the test of time and proven its scalability.


Translation Manager - C:\Delphi5.0\Projects\Test.dpr

	Id	English (British)	Status	French (Standard)	Previous English (Briti
1	0:Form1.Left	180	Unused	180	
2	0:Form1.Top	126	Unused	126	
3	0:Form1.Width	696	Unused	696	
4	0:Form1.Height	480	Unused	480	
5	0:Form1.Caption	'Form1'	Translated	'Le Form1'	
6	0:Form1.Font.Charset	DEFAULT_CHARSET	Unused	DEFAULT_C	
7	0:Form1.Font.Color	clWindowText	Unused	clWindowTe	
8	0:Form1.Font.Height	-11	Unused	-11	
9	0:Form1.Font.Name	'MS Sans Serif'	Unused	'MS Sans Se	
10	0:Form1.Font.Style	[]	Unused	[]	

5/10 | Translated: 1 | Untranslated: 0 | C:\Delphi5.0\Projects\fra\TestU.dfm

Figure 13: The Translation Manager.

Localisation

Delphi 4's Resource DLL Wizard now has some friends to help you with developing localised applications. The Integrated Translation Environment (or ITE) is a set of three tools integrated with the IDE to help manage development of an application localised for several locales. These tools are the aforementioned Resource DLL Wizard, the Translation Manager and the Translation Repository.

The Resource DLL Wizard has been improved to support multiple projects at a time, and choose multiple locales to have resource DLLs created.

Having made some resource DLLs, the Translation Manager shows all the forms, string resources and any other pertinent files from each resource DLL. You can translate the individual resources from within the Translation Manager, and it shows which items have yet to be translated. It can also add or remove resource DLLs by invoking the wizard as needed.

A neat feature of the Translation Manager is the Active Language, which lets you run localised versions of your application under the debugger.

The Translation Repository can be found on the `Tools` menu. This is a centralised storage location for translations that can be shared between different projects and be accessed by different developers. The translations in the repository can be accessed through the Translation Manager. Figure 13 shows the Translation Manager retrieving strings from the repository. The ITE can be tailored to a certain extent using a new page in the environment options dialog.

Additional Stuff

Since the reorganisation of Inprise *(What's that? Déjà vu? I've had that feeling before, you know...)* into Inprise and borland.com, and then into Inprise and Borland, they seem to have shifted their focus slightly away from Enterprise users and back to the humble developers again.

Take, for example, the inclusion of the source for the Decision Cube components, which allow you to do drill-down, pivot-able data analysis forms. Granted, a number of people have issues with these components, but they can be powerful allies in the fight to make an application management-friendly.

Another new addition to the CD-ROM is the old Borland Resource Workshop, much missed since the demise of Borland Pascal 7. This old tool allows you to easily create and manage string tables, and other standard Windows resources.

The development environment now supports a number of documented command-line switches, some of which were supported by Delphi 4 in an undocumented capacity. You can entice the IDE to keep a constantly updated log of its memory allocations on its caption bar (`-HM`), and also to report any heap corruption errors there as well (`-HV`). The normal splash screen can be disabled (`-NS`) and also the default project can be prohibited from loading (`-NP`). Automated makes (`-M`) and builds (`-B`) are supported as are a number of debugging options.

Product Versions

As usual, the product comes in three main versions, although like C++Builder 4, the names have changed around a bit. We still have the entry level Standard version, and the mid-range Professional version. What used to be called the Client/Server Suite is now Delphi 5 Enterprise and the very expensive high-end version that *used to* be called the Enterprise version (to be released later in the year) is now Delphi 5 for Application Server.

At the time of writing, pricing information is unavailable.

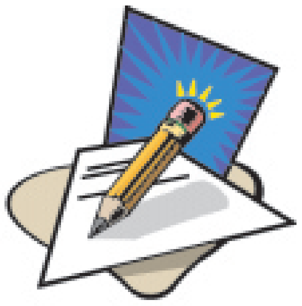
Conclusion

I think Delphi 5 is a cracker of a product, with lots of new, fun things in it. But then I enjoy using most versions of Delphi. So if that makes me a little biased, perhaps you should check out the feature set described in this review and then try the product out for yourself.

To be honest, customers buying the Enterprise Version will be getting quite a lot for their money. If you have the money for that version and you are focusing on localised applications, or database-oriented internet applications, then you will probably be the happiest customers. Purchasers of the Standard and Professional versions may not have quite so much to shout about. Details of what is to be included in each flavour are not fixed as I write, but there are rumours that the Standard version may have database support completely removed. Also, the Professional version may not ship with the ADO support, although it may be available for separate purchase.

Brian Long is an independent consultant and trainer. you can reach him at brian@blong.com

Subscribing To Developers Review!



When creating world-beating applications to tight deadlines it's essential to have the right tools to hand.

To ensure you get the most up-to-date information on what software development tools to use, subscribe to *Developers Review* now. You'll receive 6 issues a year containing in-depth reviews from leading authors, product surveys, 'first-look' reviews of new products and new releases, book reviews, news and much more.

To start receiving your regular copy of *Developers Review* simply fill out the subscription form below and send it off by post or fax as indicated. Telephone subscription orders are welcomed on +44 (0)181 249 0354. For more information visit our website at www.itecuk.com

Developers Review Subscription Request

BOS

Please complete this form and post or fax it to:

Developers Review, 9a London Road, BROMLEY, Kent BR1 1BY, United Kingdom

Fax subscriptions to: 0181 249 0376 (UK) +44 181 249 0376 (International)

Payment MUST be included, a receipt will be sent to you. Sorry, NO purchase orders!

Name (Mr/Ms) ____________________

Position ____________________ Company ____________________

Address ____________________

____

____________________ City/Town ____________________

County/State ____________________ Postcode/Zipcode ____________________

Country ____________________ Email ____________________

Telephone ____________________ Fax ____________________

Subscription (please tick):

❑ United Kingdom £36.00 ❑ Europe £40.00 ❑ Rest of the world £50.00

Note: varying costs reflect postage.

Payment Method (please tick):

❑ Please debit my ❑ *Visa* ❑ *Mastercard* ❑ *American Express* account by £____________

Card number: ____________________ Expiry date: _____ / _____

Cardholder name: ____________________ Signature: ____________________

Note: Your credit card will be debited by the Sterling amount shown above, converted to your local currency by your credit card company

❑ I enclose a **Sterling** cheque or bank draft drawn on a **United Kingdom** bank (ie bank's address on the cheque/draft is in the UK), or a **Sterling Eurocheque**, (**PAYABLE TO ITEC PLEASE**) for £___________
Sorry, we can't accept payment by bank transfer or Giro transfer. Please do not send payment in other currencies