What's New In Delphi 6?

Brian Long reviews the latest version of Borland's finest

The wait is over and Delphi 6 is here, so let's start a journey through the new features and capabilities that have been added.

Our Delphi 6 examination is a double-act: I'm going to look at what's new, and Dave Jewell follows in the next article with a look at the cross-platform development aspects of Delphi 6.

There are three flavours of Delphi 6: more or less as before, but the names are slightly different. The most expensive is the Enterprise Edition, then comes the Professional Edition and cheapest is the Personal Edition.

If you read Borland's press release about Delphi 6, you will see that it focuses on a few high-profile areas which, as usual, have been subject to the marketing folks' whims and been given snappy names (pun unintended): BizSnap, WebSnap and DataSnap. The idea of these names is to suggest how easy it is to assemble and complete projects using these new technologies, but more on these shortly.

Whilst earlier versions of Delphi had the emphasis on productivity and performance, Delphi 6 focuses its attention on scalability. Some of the key areas in Delphi 6 are targeting e-Business applications, enterprise solutions and cross-platform development. That's not to say the lone developer who writes database applications has nothing to look forward to here. As well as the high profile features, there is a whole raft of improvements and additions.

Note that this review is based on a pre-release version (although it was very close to the final build) so some things are susceptible to change in the shipping version.

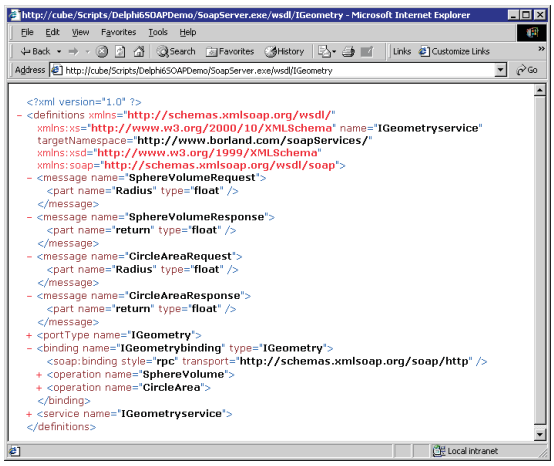
Now, without further ado, let's start looking at the new features, firstly turning our attention to those that Borland seems to be pushing the hardest (and are found only in the Enterprise Edition).

BizSnap

The tagline for this feature set is: *BizSnap Delivers Full Business-to-Business Web Service Integration*. What this means, firstly, is that Delphi now offers the ability to create industry standard SOAP/XML web services that can be used by any application that knows how to talk to web services. What it also means is that you can create client applications that make use of web services. This allows Delphi applications to integrate with web services enabled platforms such as Microsoft's .NET and BizTalk, Sun's ONE and Oracle's .NOW.

Creating the web service is easy enough. You implement the interface that is to be made available, but inherit from `IInvokable`. `IInvokable` is much the same as `IUnknown` but has runtime type information (RTTI) generated for itself and anything based upon it.

Once you have registered the interface and the implementing class with the invocation registry, it becomes an invokable interface. You use a `THTTPSoapDispatcher` from the new `WebServices` page of the Component Palette to pick up HTTP SOAP messages, which are then dispatched to a `THTTPSoapPascalInvoker` component, which uses the invokable interface's RTTI to directly call its methods. A `TWSDLHTMLPublish` component also allows WSDL documents, describing your available web services, to be made available to non-Delphi clients. Figure 1 shows what it generates for a simple invokable interface called `IGeometry`, defined with two methods: `CircleArea` and `SphereVolume`.



➤ *Figure 1: A sample WSDL document from a TWSDLHTMLPublish component describing a Delphi-written web service.*

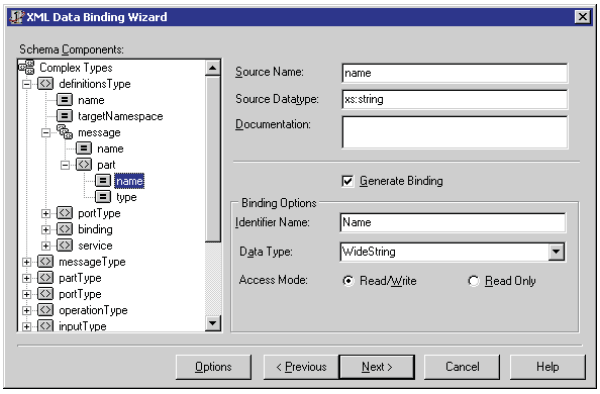
On the client side, you can import a WSDL document using the new web services Import Wizard if the web service is not written in Delphi. If it is a Delphi server, you can simply share the interface unit with the server. A `THTTPRIO` component can be used to generate statically-linked calls to the web service's remote interfaced object (RIO).

Another part of BizSnap is the support for XML document manipulation. The `TXMLDocument` component represents an XML document that can be read from a file, freshly created, or made from a manufactured string. You can read the document, edit it, and save any changes you make. You can also use it to access the objects generated by the new XML Data Binding Wizard (see Figure 2) which understands XML data, DTDs and schema files such as XSD and XDR files). `TXMLDocument` supports using external XML DOM Level 2 parsers to analyse the XML document (such as those from Microsoft or IBM), meaning you aren't locked into a single vendor.

On the data access side, there are a number of XML transform components to help your application use XML documents in lieu of database servers. They convert to

Acronym Overload?

Check Brian's Acronym Glossary in the Delphi6 directory on this month's companion disk.



➤ *Figure 2: The XML Data Binding Wizard.*

and fro between client dataset data packets and XML documents, and rely on you defining the transformation between the two. An XML mapping utility is provided on the `Tools` menu to set up these transformations.

According to Borland, this no-nonsense web service and XML support should allow you to readily move your business into the next wave of B2B e-commerce.

WebSnap

Whilst the old WebBroker support for building web server applications is still there (and in fact has been enhanced to support Apache Web Server for Windows), it is now accompanied by WebSnap (sometimes seen referred to as SiteExpress). WebSnap augments WebBroker with new components, wizards and views, that make it easier to build web applications that contain complex, data-driven, web pages. WebSnap's support for multiple modules and for server-side scripts also makes team development easier. Server-side scripting is available in JavaScript, VBScript or any other ActiveScript language you care to use.

Thanks to the WebBroker support, we already have dispatchers that handle requests for page content, HTML form submissions and dynamic image requests. WebSnap introduces new components called *adapters* which provide a means to define a scriptable interface to your application's business rules (for example, `TDataSetAdapter` is used to make dataset components scriptable).

WebSnap also gives us more producer components to quickly build complex, data-driven forms and tables, or to use XSL to generate a page. The session component allows your application to keep track of end-users and the user list component makes it easy to provide access to user names, passwords and access rights.

The WebSnap web application wizard quickly creates the building blocks of an application with the components that relate to your requirements. There are also wizards available for rapidly creating WebSnap data modules and page modules. You can think of WebSnap as taking over where InternetExpress left off. InternetExpress used scripts to provide more flexible web pages that permitted edited data and so on (albeit using MIDAS at the same time, which WebSnap doesn't).

WebSnap provides much more in the way of high-level facilities for building functionality into a web application. That said, because there are lots of new components (seventeen of them) on the `WebSnap` Palette page, you may find the learning curve quite steep (though I'm assured that it's also short). You can see a simple WebSnap data editing form in Figure 3.

In the same vein, Delphi 6 also comes with a new web application debug server for WebBroker and WebSnap applications. This expects to be run against web server applications set up as COM servers (you choose web app debugger executable as opposed to CGI, WinCGI etc). However, once you have debugged your code, you can readily shift all the web logic into a real web application (just as you can move ISAPI WebBroker code into a CGI application).

The debug server lets you monitor HTTP requests, responses and response time and precludes the need to install a web server on your development machine. It emulates the messages that would typically be sent by a real web server, thereby allowing you to easily debug through your application with the normal IDE debugging facilities.

This new debugging tool is supplied in both the Professional and Enterprise Editions.

DataSnap

The old MIDAS acronym that lived through Delphi 3, 4 and 5 has now been superseded with the term DataSnap (although the runtime DataSnap DLL is still called midas.dll). DataSnap is the natural progression of MIDAS, which allows client applications to talk to application servers and exchange data from any supported RDBMS, in either proprietary binary format or in XML. The communication between client and server can be made using COM/DCOM, CORBA (IIOP), sockets (TCP/IP) or HTTP. SOAP (HTTP and XML) is now another option with the new `TSoapConnection` component.

When building application servers, as well as Remote Data Modules (for DCOM and socket access), Transactional Data Modules (historically called MTS data modules, which now work with COM+ as well) and CORBA Data Modules (form CORBA connections) you can also use a Soap Server Data Module. This would

➤ *Figure 3: A simple WebSnap database record editing form.*

make a web service that happened to be a DataSnap application server.

DataSnap also includes a number of additional components that act as connection brokers. `TClientDataSet` has a new `ConnectionBroker` property that can connect to one of these components and the idea is to add an extra layer of indirection in the specification of a connection. For example, this makes it simple to use one connection at design-time and another at runtime, just by changing the `Connection` property of the connection broker.

`TLocalConnection` acts like a connection component for local providers (in the same application as the client dataset). It allows you to make use of the `IAppServer` interface, simplifying the process of scaling up to using a remote provider in an application server at some later date. `TSharedConnection` allows a client application to connect to an application server that is partitioned into multiple remote data modules using a single connection. Finally, `TConnectionBroker` centralises the connection to an application server of a set of client datasets that all use the same connection.

CORBA

If you are into CORBA you will probably be aware of the updated support for it made available for Delphi 5 users some time in 2000. This consisted of an IDL2PAS compiler being provided, as well as some updated Delphi units to support it, allowing you to move away from the Delphi 4 model of CORBA support, which was inherently tied also to its COM support.

Well, Delphi 6 has all this supplied as standard. You can choose to install VisiBroker 3.3 or 4 during the installation and, depending on which one you choose, appropriate Delphi support is installed. Both versions come with the event and naming services.

There are new wizards for generating CORBA client and server applications that use IDL2PAS to do their job. The CORBA support appears to now be very usable, with the inclusion of IDL2PAS, and CORBA developers should no longer have to use the IDE's COM-oriented Type Library Editor.

One feature that will be of interest to enterprise developers is that Delphi applications can now interface with Enterprise Java Beans (EJBs), with the appropriate intermediate software. Borland AppServer 4.5.1 and later supports Simplified IDL (SIDL), which makes this kind of link possible.

IDE

Having gone through most of the new features that are solely available in the Enterprise Edition, now let's pay some attention to things that are available in other editions as well. The new Delphi 6 IDE can be seen in Figure 4. Even a quick glance at it should reveal some of the changes that have been made.

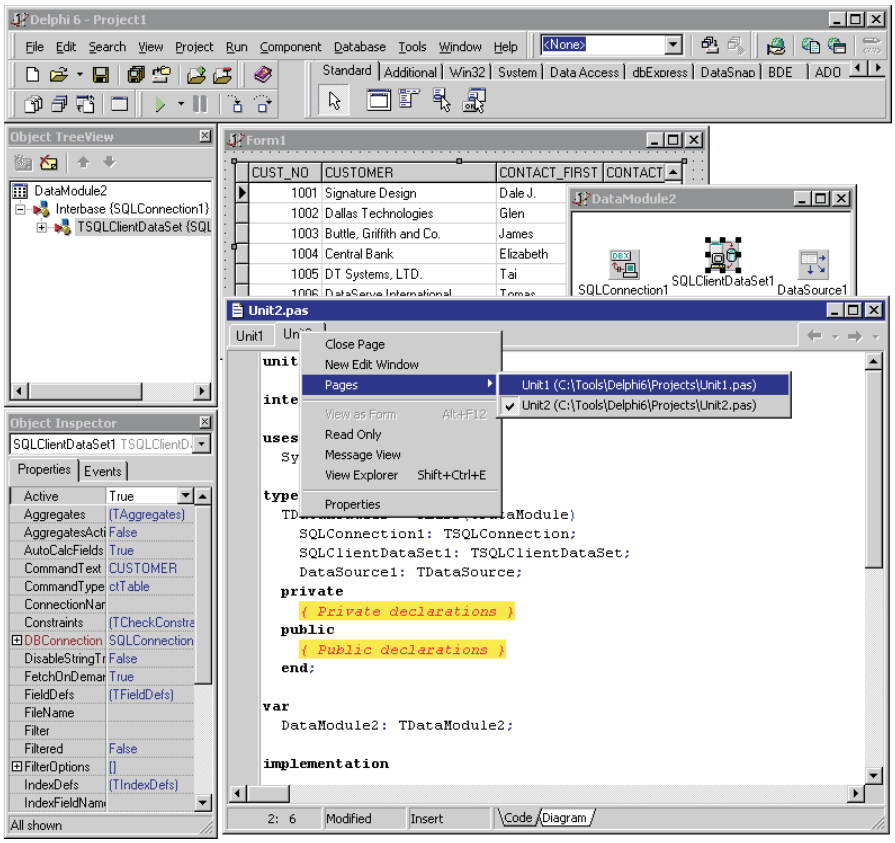
Firstly, the Data Module Designer now looks like it did in Delphi 4 (a simple container for non-visual controls). Delphi 5 had enhanced it to include a diagram page and a hierarchy pane. Delphi 6 has stripped those back away from the Data Module Designer and made them usable by any designer.

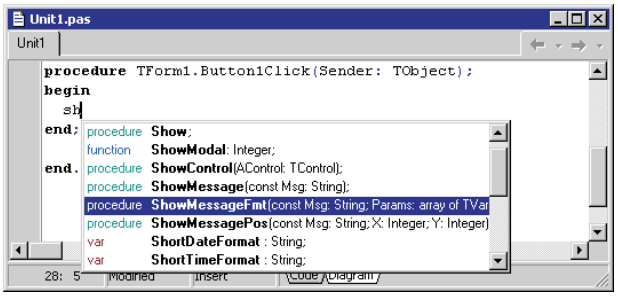
You can see the new Object TreeView in the screenshot. This is a general purpose hierarchy that shows the logical relationships between all components on data modules, forms, frames and web modules. By default it sits above the Object Inspector, but can always be brought to the foreground with `Shift+Alt+F11`.

Also visible in the code editor are some tabs right down the bottom (rather like those used in Delphi 1). These tabs switch between different views on the current file where applicable. The `Code` tab obviously shows source code and the `Diagram` tab (not in the Personal Edition) allows you to set up and document relationships between various components that have been dragged from the Object TreeView.

When developing WebSnap applications, more tabs appear that make it easy to see the result of server-side script without running the application. The `Preview` tab shows the page in an embedded browser. The `HTML Result` tab shows the generated HTML. The `XSL Tree` and `XML Tree` tabs make it

➤ *Figure 4: The Delphi 6 IDE.*





➤ *Figure 5: Code Completion in Delphi 6.*

easier when working with XML and XSL. If you open HTML or XML files in the IDE, you again get extra tabs to give alternate views on them. These extra tabs (particularly the WebSnap ones) are sometimes referred to as *surface designers*.

The editor now has a key mapping for Visual Basic keystroke emulation. It also has a new sample key mapping (in a supplied demo package) that emulates a subset of the Emacs keystrokes.

Another change for the editor is an update to Code Completion (see Figure 5). The Code Completion window (which can be invoked with `Ctrl+Space` when it doesn't pop up automatically) is now resizable. It also strips out any methods that are used in property read/write declarations, so that you are encouraged to use the property itself. As you type more characters, the window dynamically removes inappropriate entries, effectively honing your choice as you type. It works in unit interface sections and uses colours to highlight procedures, functions, etc, making abstract routines display as red.

Another change in the editor is that the individual source file tabs at the top can now be re-ordered by dragging them around. Also, right-clicking on them produces a new context menu which allows you to switch to any other open editor file (visible in Figure 4). The list of files in this menu can be optionally sorted alphabetically.

Talking of popup menus, the one on the Component Palette now gives an alphabetically sorted sub-menu of its pages, allowing you to easily switch to a page that is currently not visible on screen. This is very handy, as there is still no multi-line option and, with 25 pages available, you can only see about half of them without scrolling at 1024x768.

The main window's menu bar has a new `Window` menu to allow you to switch between IDE windows without resorting to the Window List dialog (`Alt+0`). There's also a helper shortcut of `Alt+End` which cycles through all the IDE windows in turn.

The form designer now reacts correctly to the Windows context menu key (the one with the picture of a popup menu on it). Moreover, the form designer's tooltip is much more helpful by default. When you pause your mouse over a non-visual component it tells you the name and type, as in Delphi 5. For controls not inherited from `TWinControl` it also gives the top left position, height and width of the control. `TWinControl`-based components also report the value of their `TabStop` and `TabOrder` properties.

All the options for the form designer are now available in a separate `Designer` page of the environment options dialog, leaving more space on the `Preferences` page, which now has a welcome addition to it. There is a new switch that allows the auto-docking feature of the IDE to be disabled. Normally, as you drag windows around the IDE, they tend to try and dock in any other window that's nearby (unless you hold down `Ctrl`). With the new option set, docking will not occur *unless* you hold down `Ctrl`.

The Object Inspector hasn't been left out of the makeover. Its instance list (the combobox at the top) has been enhanced to display the name and type of all components in the list, and also displays a tooltip with the same information which helps with long component names. When a component property refers to another component (an inline component), the inline component can be expanded and have its properties directly accessed.

This can continue down a number of levels as shown in Figure 6. There, a `TDBGrid` component has a reference to a data source in its `DataSource` property, which has been expanded to reveal the data source component's properties. Similarly, the data source's `DataSet` property has been expanded to show the properties available which include `DBConnection`. This property also refers to another component and, again, that component's properties can be edited.

The same principle applies on the `Events` page of the Object Inspector. Note that the colours, along with various other attributes of the Object Inspector, can now be customised on the `Object Inspector` page of the environment options dialog.

Some other useful IDE enhancements include the support for environment variables in directory settings, and a dedicated page in the environment options dialog for viewing and overriding current environment variable values. The historic `$Delphi` pseudo-variable, which could be used in path specifications, can now have its default value overridden here.

The final noteworthy IDE change is to the `File` menu. As Figure 7 shows, `File | New` is now a submenu that includes a number of common items, as well as a means of getting to the new items dialog.

Compiler

There have been various changes made to the compiler since Delphi 5, some of which were made for the benefit of Kylix, but which have now propagated back into the Delphi product. For example, values in enumerated types can now be explicitly assigned ordinal values. The default behaviour is that the first value in an enumerated type has an ordinal value of 0, and each successive one has an ordinal value one higher.

C and C++ consider enumerated type values to be integers and allow you to override the default compiler values, and now Delphi can do this as well (although Delphi still considers an integer

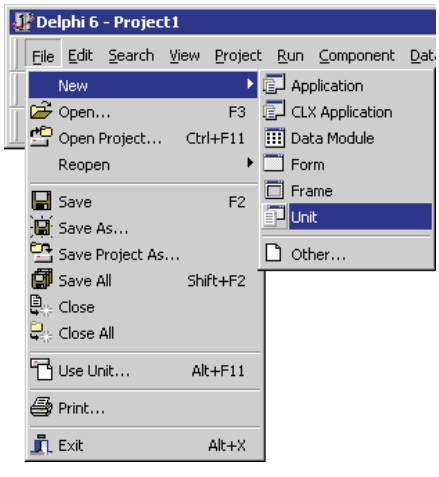
and an enumerated type value rather different, due to its stronger typing). Listing 1 shows a sample enumerated type, with two explicit ordinal values being assigned. Because of the language rules, the commented version shown is an equivalent type definition. This feature is very useful when you need to interact with C or C++ libraries that use enumerated types defined in this way.

The next change is that writable typed constants are now disabled by default (the equivalent of `{$J-}` or `{$WRITEABLECONST OFF}`). Typed constants allow you to set up structured constants with values, but ever since the days of Turbo Pascal, they haven't actually been constant. Delphi 2 added a compiler switch and compiler directive that would prohibit any attempt to assign to typed constants, but it defaulted to the old behaviour (where typed constants act like C static variables).

The default now is to act as if typed constants are actually constant. The suggestion is to use initialised non-local variables to achieve the same effect. I gather the RTL/VCL was scoured for examples of using writable typed constants, as their use is now frowned on in Borland R&D.

To help with source code that may be cross-platform, or perhaps just used in several versions of Delphi, there are several new 'hint' directives. The `library` directive flags a dependency on a particular library or component framework,

➤ *Figure 7: The reorganised File menu.*



such as VCL or CLX. The `deprecated` directive indicates that an item is obsolete or supported only for backward compatibility. Finally, the `platform` directive indicates that an item is specific to a platform (Windows, Linux, etc).

These directives can be applied to any declaration (including that of a unit), as shown in Listing 2. Simply inserting these directives has no effect until you refer to the affected identifiers, which results in compiler warnings saying the referenced symbol is specific to a library or platform, or deprecated.

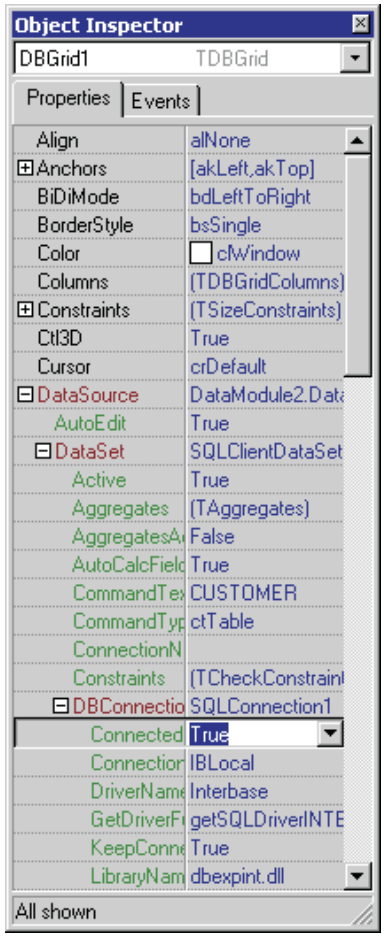
These new warnings can be disabled by turning off all warnings, or toggled on or off individually with the new `$WARN` compiler directive. For example, `{$WARN SYMBOL_LIBRARY OFF}` disables warnings about symbols that are library-specific.

Another new directive allows you to link to C/C++ routines that take variable numbers of arguments (like `printf` and `sprintf` do). You append the `varargs` directive at the end of your import declaration and simply declare all the fixed parameters, as in Listing 3.

If you are writing code that may be compiled on both Linux and Windows, the new `local` directive may help. This has no effect on Windows, but routines compiled into libraries (Linux shared modules) but which are not exported are made more efficient by applying the `local` directive to them.

Talking of compiling for Linux and Windows, you should also know of the new `MSWINDOWS` conditional symbol. Delphi 1 defined `WINDOWS` for Win16, Delphi 2 through 6 defines `WIN32` for Win32, and when Win64 comes along, that will cause another symbol to be introduced. As of now, the new `MSWINDOWS` symbol simply means a Microsoft Windows platform. As a side note, for code that may need to be written differently for different versions of the underlying compiler, the conditional symbol `VER140` has been defined (and is also defined in Kylix 1).

Custom hints, warnings and errors can now be generated from your source with the new `$MESSAGE` compiler directive. Listing 4 shows



➤ *Figure 6: Expanded inline component references.*

the four levels of message you can emit and, as you can see, an error can either be non-fatal to the compilation or fatal.

When building DLLs, you can fully customise how the file name will be built up using the `$LIBVERSION`, `$LIBSUFFIX` and `$LIBPREFIX` compiler directives (or the equivalent options in the project options dialog). For example, all the runtime packages that are supplied with Delphi have had the version numbers removed from the source files. This leaves compiled package files with names like vcl.dcp. Thanks to a `$LIBVERSION` directive, the actual binary package is called vcl60.bpl, but your programs that link to it need not know that. These directives were added partially to solve the irritation of updating your runtime package list each time you upgrade to a new version of Delphi.

For those developers who are writing specialist applications that require custom bits set in their

```
type
  TValues = (vRed, vOrange, vYellow, vGreen = 32,
    vBlue, vIndigo, vViolet = 64);
  //TValues = (vRed = 0, vOrange = 1, vYellow = 2,
  //  vGreen = 32, vBlue = 33, vIndigo = 34, vViolet = 64);
```

➤ *Listing 1: An enumerated type with explicitly assigned ordinality.*

```
const
  VCLVersion = 6 library;
var
  Win32Version: Single platform;
function GetAutomatedSectionEntries(Obj: TObject): Pointer; deprecated;
...
```

➤ *Listing 2: Using the new hint directives.*

```
function printf(Format: PChar): Integer; cdecl; varargs;
```

➤ *Listing 3: Importing a C routine that takes a variable number of arguments.*

```
{$MESSAGE HINT 'Remember to eat'}
{$MESSAGE WARN 'Say no to rugs'}
{$MESSAGE ERROR 'D''oh!'}
{$MESSAGE FATAL 'Knucklehead'}
```

➤ *Listing 4: Custom compiler messages.*

```
program Project1;
uses
  Windows, Forms,
  Unit1 in 'Unit1.pas' {Form1};
{$SetPEFlags IMAGE_FILE_AGGRESIVE_WS_TRIM or IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP}
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

➤ *Listing 5: Setting PE file header bits.*

executable header, there are two new directives that make the process much more straightforward. `$SetPEFlags` can be used to set flag bits in the PE file header `Characteristics` field whilst `$SetPEOpt Flags` sets bits in the optional header `DLLCharacteristics` field. You should use these directives in your project file and they can take an integer expression, possibly made by `OR`-ing constants from the `Windows` unit (see Listing 5).

Perhaps the most useful improvement in the area of directives is with the introduction of the `$IF` and `$IFEND` directives with constant expression evaluation. You can build any expression as long as it only relies on literal values and constants that have already been defined. You can test to see if a symbol has been defined (as `$IFDEF` does) with the new `Defined` intrinsic function, and you can check whether an ObjectPascal constant has been declared with the `Declared` function. Listing 6 shows a simple example of it in use.

If you want to use these new directives in code that may be compiled by older versions of Delphi, you can wrap it up in `$IFDEF` directives that check whether `CONDITIONALEXPRESSIONS` is defined, which it is in Delphi 6 and later.

The old built-in assembler, BASM, has been laid to rest. It was written in a non-portable fashion and needed to be replaced by something that would work in Kylix. The Kylix and Delphi 6 inline assembler is now called CHASM and has been written with portability in mind and has instruction support for MMX, Enhanced MMX, SIMD and Intel SSE for the Pentium Pro, Pentium III, and Pentium 4 CPUs; and AMD Enhanced 3D for AMD K7 CPUs. However, I should mention that, if you are into this sort of thing, you should note the CPU window has not yet been enhanced to understand the additional registers that can be referenced by inline assembler code.

COM

COM support is not in the Personal Edition of Delphi 6, but in the other flavours we now have support for COM+ in Windows 2000, including neutral threading model support and the ability to specify COM+ attributes in the Type Library Editor. Delphi 5 offered MTS support in the Enterprise Edition only, but Delphi 6 Professional Edition offers the new Transactional object support for both MTS and COM+.

When making a new COM object with the COM Object Wizard, just as in C++Builder 5 you can elect to have your object implement any of the interfaces registered on your system (see Figure 8). The wizard will make sure your object has stubs for all the methods in the selected interface.

A new wizard has been added for building COM+ event objects and a new `TCOMAdminCatalog` component appears on its very own `COM+` page of the Component Palette to allow an application to act as an Automation controller for the COM+ administration tool.

Database

As in Delphi 5, database support is also present only in the Professional and Enterprise Editions. Not much has happened with the BDE (now at version 5.1.1), although all the BDE-related components have been moved onto their own `BDE` Palette page. So each of the three individual data access mechanisms from Delphi 5 now has a dedicated Palette page: `BDE`, `ADO` (for the dbGo components, or ADOExpress as they used to be called) and `InterBase` (for the IBX, or InterBase Express components). However, Delphi 6

introduces a fourth data access technology: dbExpress.

dbExpress is a lightweight, cross-platform set of database drivers that provide fast access to SQL database servers, and is provided with both Delphi 6 and Kylix. The `dbExpress` Palette page has the basic `TDataSet` descendants that interact with the driver that you'd expect (`TSQLTable`, `TSQLQuery`, `TSQLStoredProc` and `TSQLDataSet`).

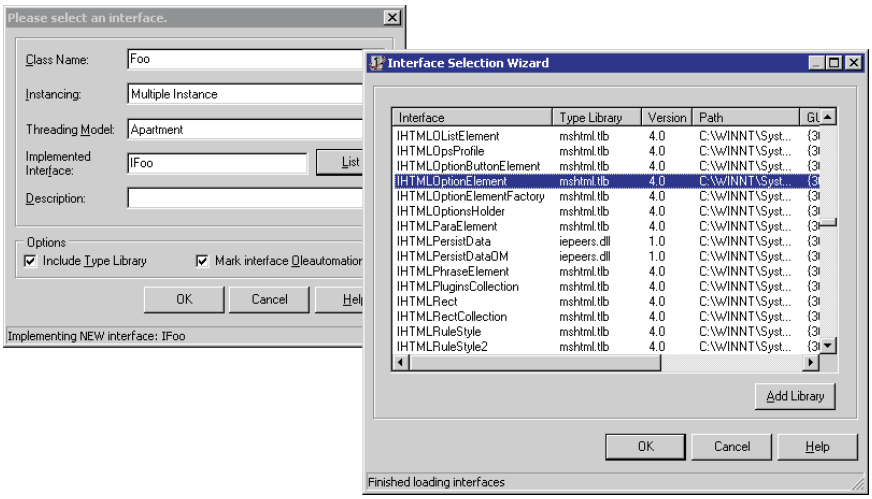
Deployment involves distributing the single driver DLL and a text connection configuration file, or compiling everything into your executable for simplicity.

dbExpress supports multiple transactions across a single connection and uses unidirectional datasets with no inherent client buffering, for efficiency. However, you can connect a dbExpress dataset to a client dataset to get buffering and bi-directional movement if needed. In fact a `TSQLClientDataSet` is ready and waiting for you to use. This component wraps up a `TSQLDataSet`, `TDataSetProvider` and `TClientDataSet` so you needn't worry about connecting these components together: it is just about visible in the data module in Figure 4.

Other changes in the database offering include an update to `TUpdateSQL`, which can now be used against multiple datasets (and they can be *any* type of dataset now, not just BDE datasets). Also, `TClientDataSet` has a new `XMLData` property which gives access to its data packet in XML format (the original `Data` property gives the data packet in an internal binary format).

Whilst on the subject of `TClientDataSet`, I should point out that Borland is pushing the trademarked term *MyBase* as a personal XML database engine. MyBase is just a made up name to describe the ability of any client dataset to work in briefcase applications. Also, as client datasets can store their data (which can be imported from any dataset) either in proprietary binary format or in XML, it encourages its use as a 'database-engine-in-a-component' for lightweight single-file applications, supporting popular data types, including BLObs.

➤ *Figure 8: The enhanced COM object wizard.*




```
const
  MyConst = 10;
{$IF Defined(WIN32) and Declared(UsingVCL)}
  //code that manipulates the VCL
{$IFEND}
{$IF Defined(WIN32) and (MyConst > 9)}
  //more code
{$IFEND}
```

➤ *Listing 6: Using the new conditional compilation syntax.*

Actions

So what's new in the realm of actions? Quite a lot, as it turns out. All actions have additional properties, that allow you to specify a secondary shortcut and a group index, amongst other things.

Also, there are three dozen new standard actions to choose from, including some which are dedicated to modifying various attributes in a rich edit control and some dedicated to manipulating list boxes (for example clear selection, copy selection, move selection). They also include some related to the internet (such as browse URL or send mail) and some related to common `File` menu operations (such as exit and open).

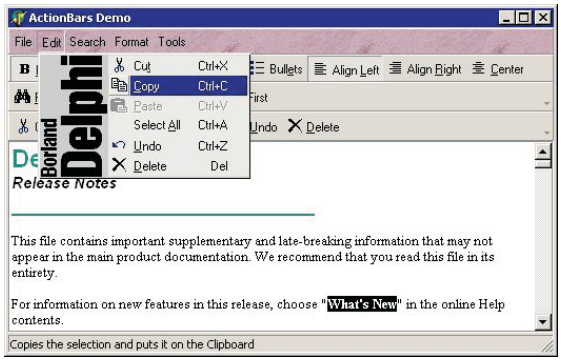
The main change here is the introduction of components that allow you to visually lay out actions on dedicated versions of toolbars and menu bars using the new action manager component. Infrequently used menu items set up in this manner can be hidden in the same way as in the Windows 2000 `Start` menu. You are given control over the background pattern, bitmap or colour of the toolbars, menu bar and menus, and can even make banners that go up the side of a menu (again, like the Windows `Start` menu). Figure 9 shows a demo application with a bitmap tiled across the menu bar, and another bitmap used as a left hand banner in the `Edit` menu.

Custom Variants

The support code for `Variant` variables has been moved into the `Variants` unit and the code has been made portable to other platforms (meaning `Variants` are available in Kylix). Additionally, whilst making this change, they added in support to allow a `Variant` to take custom data types. As an example, the `VarCmplx` unit implements a custom `Variant` for complex numbers. Also, in the Enterprise Edition, the `SqlTimSt` unit implements a custom `Variant` for `TSQLTimeStamp` values, the type used by dbExpress database drivers to represent date/time information.

RTL

As well as the introduction of the new `Variants` support unit, there are a number of other additions to the RTL. A lot of work has been put into making a large number of conversion routines for various



➤ *Figure 9: Action bands in action.*

measurements. The `ConvUtils` unit contains the conversion registration routines (all conversions are centrally registered so that they can be easily managed). Most of the actual conversions are implemented in `StdConvs` but additional conversions for translating to and from the Euro currency can be found in `EuroConv`.

A demo application uses the conversion management routines in `ConvUtils` to iterate through and list them all on a form, allowing you to test them out. Figure 10 shows a number of distance conversions, listing the equivalent of one light year in various other units.

`StrUtils` is another new unit which contains a bunch of new string manipulation routines, including routines that deal with Soundex. For example, `AnsiResemblesText` returns `True` if two strings are similar, using a Soundex algorithm by default, but this can be switched to some other kind of algorithm if needed.

`DateUtils` introduces a mass of new ISO 8601 compliant date/time manipulation routines, including `DaysBetween`, `EndOfADay`, `JulianDateToDateTime` and `IsInLeapYear`: I counted more than 150 routines in total.

The old `Math` unit has also been enhanced. Firstly, all `Extended` arguments are now passed as `const` parameters for efficiency. But there are also many new routines present, such as angle unit conversions, hyperbolic functions and inverses, Infinity and NaN testing, floating point comparisons, sign reporters, random values from a specified ranges, easy-to-use conditional functions and FPU exception, precision and rounding management routines.

VCL

Now we turn our attention to the VCL to see what else has changed (we've already seen all the new classes related to web services, actions and databases). The `Additional` page of the Component Palette is host to a group of new custom components. The first is `TValueListEditor`, which presents the user with a way of viewing or editing string lists with the form `Name=Value`. Rather like the IDE's Object Inspector, you can let the user edit the string, provide a drop down list of values, or provide a dialog in which they can edit the string.

Next up is `TLabeledEdit`. This is a simple compound component made of a `TLabel` and a `TEdit`, but solves a lot of tedious setup in many developers' applications. The last new component on this page is the `TColorBox`: a combobox for choosing a colour that looks just like the `Color` property editor on the Object Inspector.

On the `Win32` Palette page, the sole addition is `TComboBoxEx`, which supports extended combobox functionality, such as images on the list entries and indentation of the entries (see Figure 11).

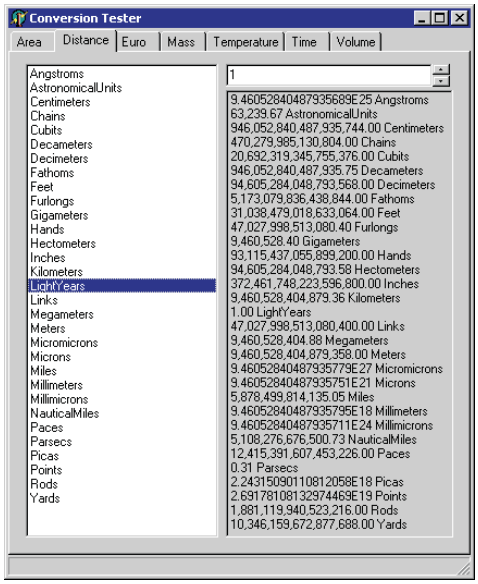
On the `Samples` page of the Palette, there are four new Shell Controls, which are intended to replace the old directory navigation and file selection components on the `Win 3.1` page *[At long last! Ed]*. You can readily link the three visual components together to make a Windows Explorer lookalike, complete with popup menus and icons. The fourth component is a change notifier and alerts you when files are renamed or deleted, etc.

Various components have had additional properties added. Here is a list of some of them. All listbox-based classes now have a new `AutoComplete` property, which makes them automatically complete words that the user types by selecting the first item that begins with the currently typed string. The listbox component itself also supports acting as a virtual listbox (where the items are stored elsewhere). Treeviews and listviews make it easier for developers to add custom nodes inherited from the default node type. The header control supports column dragging and allows you to sort columns when their header is clicked.

Toolbar components have a `Menu` property which makes them automatically absorb all the items from a menu component and replicate its layout. Forms support transparency and translucency thanks to the new `AlphaBlend`, `AlphaBlendValue`, `TransparentColor` and `TransparentColorValue` properties. The containers unit now defines `TBucketList` and `TObjectBucketList` as simple hash tables, and `TStringList` now supports case sensitivity. Also, `THashedStringList` is a string list that uses a hash table for efficient string location.

Finally in this list, `TApplication` has a new event. `OnSettingChange` reacts to changes in system-wide settings and gives all the pertinent information to the event handler.

Components now support published interface properties which will show up in the Object Inspector, so long as the interface is



➤ *Figure 10: The conversion demo program.*

➤ *Figure 11: The TComboBoxEx component.*

implemented by a component that uses the new Component Interface Reference architecture. Also, components can now own their own sub-components and still have the sub-component properties streamed out to form files.

Translation Tools

There's not too much to say about the internationalisation support in Delphi 6. It's similar to the support in Delphi 5, but the old ITE term has been replaced by the term *Translation Tools*. This is probably because the tools are not so much integrated into the IDE any longer, but run from an external utility. The new external executable is called the External Translation Manager (ETM). This can be shipped to your translators around the world without worrying about getting them a full Delphi development licence.

The ETM has a number of enhancements over the ITE. For instance it has a form viewer/designer available to allow translators to both view and resize the form and its components as translations are made. You can see the form viewer being used to look at a form translated into German in Figure 12.

Hardware Requirements

I should mention that because there is a whole lot of new stuff in the product, a full install of the Enterprise Edition now takes 350Mb (though that's without VisiBroker, the BDE, the sample images and the Win32 SDK help files (which total about 100Mb). The Professional Edition will eat up 260Mb by itself and the Personal Edition occupies 160Mb with a full install.

I should probably also mention that, because of its larger size, the performance of the IDE is poorer than the previous version. If you are going to upgrade to Delphi 6, it might also be time to look at upgrading your hardware.

To give you an idea, on my old 300MHz Pentium II desktop machine with 256Mb RAM running Windows 98, it takes just under one full minute to start Delphi 6 Enterprise Edition after a fresh reboot with no other applications running. However, on my newer 850MHz Pentium III laptop, also with 256Mb RAM, but running Windows 2000, it starts in under 20 seconds. Make no mistake, this is a large product and it warrants some decent hardware to run on.

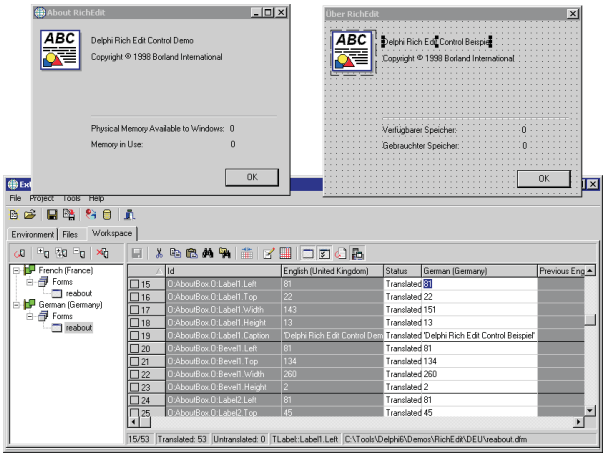
Note that the hardware requirements from Borland state that you need a Pentium 166MHz with 64Mb RAM (snigger!), although they recommend at least a 400MHz PII with 128Mb RAM.

Another couple of points worth noting include the fact that Windows 95 is no longer an officially supported platform (the data sheet lists Windows Me, Windows 98, Windows 2000 and Windows NT 4.0 Service Pack 5 or later).

Also, the massive amount of help supplied (at least in the Enterprise Edition) overflows the WinHelp index limit on Windows 95/98/Me, meaning you see nothing in the `Index` tab of the help system. If you are still developing on Windows 95/98, perhaps it is time to move over to Windows 2000. Having done it myself, I can assure you that it's not that painful.

Pricing

The product is getting pricier as versions come and go. Delphi 6 Enterprise Edition has an RRP of $1,999 with an upgrade RRP of $1,379. The Professional Edition has an RRP of $699, with an upgrade RRP of $269.



➤ *Figure 12: Translating a form into German.*

Finally, the Personal Edition has an RRP of $79. All excluding VAT of course. However, it is possible to get it cheaper elsewhere: check your favourite dealer's prices.

Conclusion

Delphi 6 is a whopper (particularly the Enterprise Edition), and it comes with a price tag to match. If you work for a large company, need to develop enterprise applications calling on web services, or rapidly build web server applications using WebSnap, then I'd imagine the price won't be too much of an issue.

But for lone developers who do not need all the additional stuff in the Enterprise Edition, the question is: does Delphi 6 Professional Edition offer enough to justify the $270 upgrade price tag? In short, assuming you have the hardware for it, I think the answer must be yes. Even for newcomers, for a product so rich in productivity features, it sounds good value to me.

The only worrying thought is that, with so many new features added to this version, how reliable and stable will it be? Time will tell, and hopefully Borland will be quick on the ball to issue updates as and when they are needed.

Now over to Dave for those cross-platform issues...

Brian Long is a freelance trainer and problem solver specialising in Delphi, Kylix and C++Builder work. Visit www.blong.com or email him on brian@blong.com