The Delphi Idiom

by Steve Teixeira

Now you have Delphi in your hot little hands you can understand what the big deal has been about for the past several months! Delphi can handle practically any application development task you throw at it – and handle it faster and more elegantly than most any other tool. But you probably already know that or you wouldn't be here, right? I'm here to tell you what you need to know to be a Delphi programmer.

It's Not Just For Prototyping

Like most products of its genre, Delphi gives you a simple, productive point-and-click interface to program creation. Unlike with some products, though, your application doesn't have the silhouette of a cow nor the speed of a sloth. Delphi combines the advantages of visual development with the performance of a true compiler.

Borland didn't cook up the Delphi compiler out of the blue, though, it's an improved version of the compiler in Borland Pascal 7. So, although Delphi is a new product, it's an evolution of one of the oldest and most trusted PC compilers.

The code generated when you compile a program in Delphi is on a par in terms of speed with that generated by a C or C++ compiler. In short, Delphi isn't a prototyping or "front-end" tool, but an all-round tool. Using it only for prototyping would be like buying a Ferrari and never pulling it out of your drive!

It's Pascal

If you're new to Delphi, you should find it easy moving up to Object Pascal. It's a sort of happy medium of languages, so whether you prefer the structure and verbosity of Basic or the flexibility and power of C++, you'll soon feel at home.

One language feature that can take some getting used to is Object Pascal's strongly typed nature. When you call a procedure, the compiler will make sure that you pass correct types, array sizes and pointers. If you come from a C/C++ background, this will probably annoy you at first, but I'll bet you will soon appreciate the number of bugs it prevents from entering your code. After all, wouldn't the ideal be to have the compiler find all your bugs? Strict type-checking is a step in that direction.

If you're moving to Delphi from Borland Pascal, nearly 100% of your code will compile without problem in Delphi. I've found the best approach to porting an Object Windows Library (OWL) application to Delphi is to redo the User Interface portions in Delphi's IDE, and hook the new UI into your application's existing back-end.

Visual Component Library

Visual Component Library, or VCL, is Delphi's object-oriented framework. In this rich library, you'll find classes for Windows objects such as windows, buttons, etc, and you'll also find classes for custom controls such as gauge, timer and multimedia player, along with non-visual objects such as string lists, database tables, and streams.

Each VCL class usually has a set of properties – such as color, size, position, caption – that can be modified in the Delphi IDE or in your code, and a collection of events – such as a mouse click, keypress, or component activation – for which you can specify some additional behavior. You'll spend most of your time in the Delphi IDE interacting with components' properties and events.

VCL is also remarkably platform-independent. VCL encapsulates even low-level Windows concepts such as Device Contexts, bitmaps, and timers. It is for this reason that the code you write in 16-bit Delphi will recompile with little or no changes in 32-bit Delphi when it is released. It's best to keep with VCL as much as possible to give your code maximum portability. You'll be surprised how much you can do without calling the Windows API.

Message Handling

Although VCL's events account for most of your needs, directly handling Windows messages is a piece of cake with Delphi's new `message` keyword. Simply create a method that takes one parameter of type `TMessage` (or other message record), and use the magic word. Let's say, for example, you want to write a handler for the `wm_Paint` message. The code would look like:

```
procedure WMPaint(
  var M: TWMPaint);
  message wm_Paint;
```

`TWMPaint` is a record type based on a `TMessage` whose fields are defined specifically for a `wm_Paint` message. Delphi defines a record type like this for every Windows message. The record type is always the same as the message name with a "T" in front and without the underscore.

Windows At Your Fingertips

Unlike some similar products, Delphi offers you the flexibility to easily call any Windows API procedure. Actually, Delphi doesn't just support this feature, but it makes it a trivial task: you just call the API procedure as if it were a procedure defined in your program.

Delphi also allows you to call procedures out of any other DLL, no matter what language it's written in. Although Windows DLLs generally use the Pascal calling convention for parameter passing, Delphi supports the C calling convention too: just use `cdecl` on your function declaration.

Exception Handling And Runtime Type Information

You can handle error conditions in your Delphi code using C++-like Exception handling. Exception handling enables you to gracefully handle specific or general error conditions by enclosing potentially dangerous parts of your code in a `Try..Except` or `Try..Finally` block. The general structure of an exception handling block looks like this:

```
try
  {some stuff }
except
  {if an exception occurs... }
  on ESomeException do
    Something;
    {Handle the exception }
end;
```

Exceptions provide a significant advantage over general error procedures in that protection can be placed where it's needed in your code. Instead of trying to handle a whole variety of possible error conditions in one place, you can tailor `Try` blocks to your needs. It's also built into the Win32 API, so you'll need to get used to it!

Runtime Type Information (RTTI) is the ability to obtain information on class instances while your program is running. RTTI is perhaps the single most important feature in Object Pascal. In VCL, most classes are passed between functions and procedures as the `TObject` base class (from which all classes are derived), which satisfies the compiler's type-checking, and RTTI is used inside functions and procedures to determine the type of and typecast these classes.

Client/Server

Delphi comes with the Borland Database Engine, a high-performance database-access layer that transparently connects you to different data sources: Paradox, dBASE, ODBC, or servers like Oracle, Interbase, Sybase, and Informix. The buzzword here is scaleability: you can start off with Paradox or dBASE tables and then transparently move to Oracle or Sybase with very little change in your application. Back-end independence means productivity, and it is the wave of the future.

Component Design

Delphi was written in Delphi. Delphi components are written in Delphi. That's not to say that Delphi is xenophobic – Delphi also allows you to use VBX controls and other Windows custom controls. Simply stated, Delphi does it all, and because of that, you can do it all. There's no line between the application developer and the component writer. All components are extensible Object Pascal classes, so you can crank out a custom component any time you need to.

Power

Because Delphi is a true compiler, you have no limits. Need to write a DLL? No problem, it's hardly any different to writing a regular unit. Callback functions? Not a problem either. Just tag that procedure with the `export` directive and you're on your way. How about inline assembly language? Sure, Delphi's Built-in Assembler makes it a snap.

Whether you want to write applications, controls, or database front-ends, Delphi is your tool. Now that you know the score, what are you waiting for? Go forth and hack.

Steve Teixeira is a Senior Technical Support Engineer for Delphi at Borland International. He can be reached via the internet at steixeira@wpo.borland.com or via CompuServe at 74431,263