

INTRODUCTION

This short ebook covers the differences between Delphi 2005 and Delphi 2006. It is meant as an update of the book “Mastering Borland Delphi 2006”, written by Marco Cantù and published by Sybex (now an imprint of Wiley).

Although you can read it by itself, there are frequent references to the book, so you need the book at hand to fully benefit from it. In fact, I avoided repeating material already in the book.

The updates in this ebook are organized by the printed book chapter they refer to. The examples discussed here are available online on www.marcocantu.com/md2005.

The Author

Marco Cantù is the author of the best-selling Mastering Delphi book series and a prominent figure of the Delphi community. He's also one of the recipients of the “Spirit of Delphi” award. Marco teaches and consults on Delphi, but also in XML-related and Web technologies.

Marco's web site is on www.marcocantu.com and his blog on blog.marcocantu.com. You can reach him at marco.cantu@gmail.com. Marco lives in Italy, with his wife Lella (an interior designer), their two kids Benedetta and Jacopo, and the dog Lillo.

CHAPTER 1: THE DELPHI 2006 IDE

The “2006” version of Delphi has the formal name of “Borland Developer Studio 2006.” Borland has kept the *product-specific* names in this release, but is likely to stop them in the future, to underline the fact you are buying a multi-language studio IDE, and bring this in line with Microsoft's offering. In this section we are focusing on the most relevant new features of the IDE.

Delphi 2006 IDE is not as revolutionary as the previous version, as it is based on the same architecture and has a very similar look-and-feel. Borland has made a lot of effort to the the new version of the IDE much more stable, as this was the most frequent complain user had for Delphi 2005. But along with more stability and speed, Delphi 2006 IDE adds a relevant number of new features.

Flexible Installation and Execution

First of all, when you install Borland Developer Studio 2006 (or BDS 2006) you can pick only some of the four personalities that are available: Delphi for Win32, Delphi for .NET, C++Builder for Win32, and C# (obviously for .NET).

Even in case you decide to install all of the personalities, you can choose which one you want to activate when you start the IDE. A set of related shortcuts are installed in the BDS 2006.

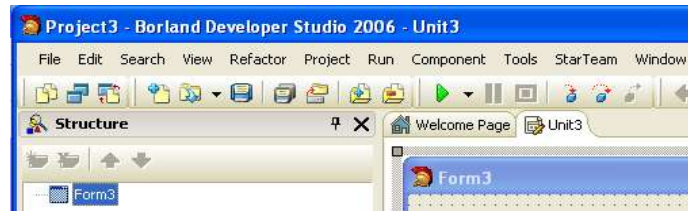
Starting only a personalities result in shorter startup time, less memory usage, and a slightly faster IDE overall. Also, the IDE will be less cluttered with options you don't case for, like in the File New menu. Starting a single personality is supported by a new command line shortcut of the BDS.EXE program, `-p`. For example, you can invoke the Delphi for Win32 personality with the `-pDelphi` option. What is not well known is that you can use the same shortcut to activate multiple personalities at once, separating them with a comma. For example, a Win32 developer might be interested in running both Delphi and C++Builder, which can be done with `-pCBuilder, Delphi`.

This new option extends the features of the `-r` option discussed in the section “Starting the IDE with Multiple Configurations” of Chapter 1 (page 6) in “Mastering Borland Delphi 2005”.

📖 Differently from the `-r` flag, if you use `-p` any change in the IDE configuration saved in the registry will be available also as you start the full IDE or other personalities. This is an advantage as you don't have to apply the settings multiple times but also a drawback, as you cannot use this option to install experimental components and add-ins you don't want around in your main development environment. In such a case the `-r` option is still the way to go.

The IDE Look-and-Feel

Even if this doesn't really affect the IDE capabilities, it is interesting to notice that the IDE has been updated in the way it looks, with some rounded toolbars and tabs, as you can see here:

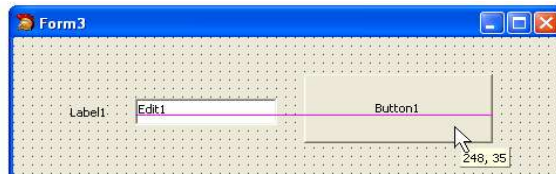


From my point of view the most relevant changes of the IDE are those affecting my everyday work, that is the updates to the form designer and to the source code editor.

The VCL Form Designer

As soon as you place a new controls on a VCL form, you'll immediately see something new. The corners of the control and the central portion of each side have a small light blue circle. These mark the resize areas of the control: moving the mouse over them you can resize it (like in the past).

Now if you place a second control in the form, you'll be able to see a more relevant new feature, “Design Guidelines”. This is a visual helper to properly align the controls on a form:



Not only you can align the sides of a control with those of another one, but you can even align the *text baseline*, as in the example above. Controls automatically snap to the guidelines when they are close to them. They also snap when they are at a given margin to the border of the container control.

Another new feature is the “Form Positioner” in the bottom right corner of the designer surface. There you can see, in small, the position of the form on the screen, which is relevant if the form uses absolute positioning. You can also use the Form Positioner to modify the `Top` and `Left` properties of the form visually. This tool is relevant only when you use the default embedded designer.

Delphi 2006 Editor

Compared to the form designer, the source code editor in Delphi 2006 has seen a higher number of new features. As soon as you open a source code file you can notice that the line numbers on the side of the source code file have changed. Instead of seeing the number of each and every line, now you see only each tenth line, with an hyphen indicating the intermediate lines (each fifth line), and a single dot for the others. The only exception is the line with the edit cursor, which has the line number displayed. Here is an example (the cursor is on line 17):

```
10  type
    TForm3 = class(TForm)
        Label1: TLabel;
        Edit1: TEdit;
        Button1: TButton;
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;
20  var
```

Notice that the line numbers are not placed in the gutter with breakpoints and bookmarks any more, but have their separate area. On the side of the editor you can also notice some color lines. These mark the source code lines modified after the last time the file was saved (yellow line) or within the current editing session (green line).

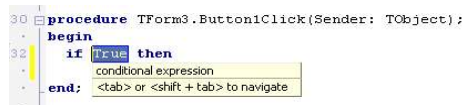
Beside these graphical elements, the most relevant changes in the editor are related to the several code writing helpers, from Block Completion to the new Live Templates, to the updated of the Code Completion technology. What is relevant to notice is that most of these new features are not activated using shortcut keys you have to learn, but show up automatically as you type, being triggered by simple keys like Enter, the Space Bar, or the Tab. Having new features embedded in Code Completion (manually triggered by Ctrl+Space, but often displayed automatically) is much better than having to invoke a specific request, like you had to do with the old Code Templates pressing Ctrl+J (a much less intuitive sequence).

Block Completion

The first new feature I want to cover is Block Completion. It automates the writing of the closing element of many code blocks. The most obvious example is the `begin-end` block. Simply type `begin` and press Enter to see the corresponding `end` show up below it (followed by a semicolon, unless the block is inside a `if-then-else` block). Block Completion helps you also with `try` blocks (adding `finally` and `end`), `case` (adds `end`), `repeat` (adds `until`), `record` and `class` (adds `end`).

Live (Code) Templates

While the Enter key triggers Block Completion, the Space and tab keys activate the new Code Templates, now called Live Templates. For example, if you type `if` followed by a space the statement will be completed with the `then` portion and a suggestion related with the test expression the user has to fill:



```
30 procedure TForm3.Button1Click(Sender: TObject);
31 begin
32   if if then
   conditional expression
33 end;
   <tab> or <shift + tab> to navigate
```

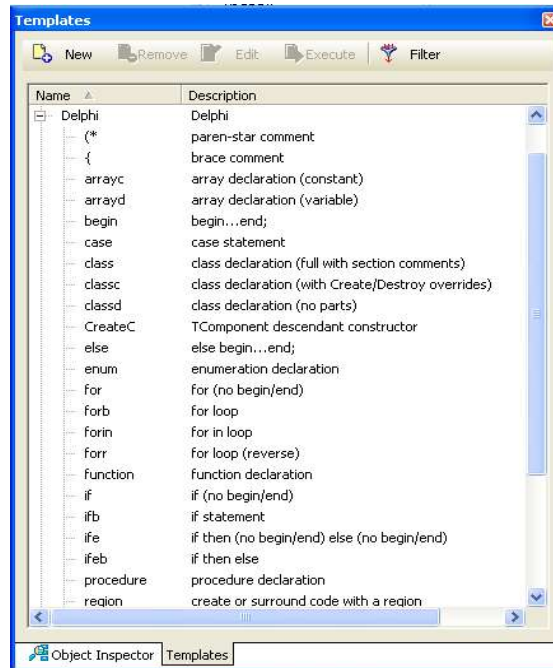
Code Templates account for the definition of complex and partial blocks of code, which need the user intervention to complete them in one of more locations. You can move around these locations with the Tab key. For example, there are multiple templates for the `try` statement. One of these is called “try finally (with Create/Free)” and when it is triggered it generates code like this:


```
:= T.Create(Self);
try
```

```
finally  
    .Free;  
end;
```

What is nice is that the editor enters in SyncEdit mode on the name of the variable, so that it is automatically repeated on the first line and on the line with the `Free` call. After typing it, you can move to the next editing location with the `Tab` key, to enter the name of the class you want to call `Create` for. When you are done, press the `Tab` key again and by exiting the Live Template this will invoke the add variable refactoring to add the proper variable declaration at the beginning of the code block (for a description of refactoring in the IDE see Chapter 11).

First, templates configured in manual mode are invoked with the `Tab` key, while automatic templates are triggered by the `Space` key. Second, Code Templates can be listed, picked, and edited in the new Template View (use the `View > Templates` menu item to show it):



 You can also work directly on them by editing their XML source code files available in the `Objrepos\code_templates\delphi` subfolder of the `BDS4.0` folder. From the IDE you can create a new Code Template with the command `File > New > Other > Other Files > Code Template` or from the Template View.

The Live Templates can also be used in *Surround mode*. The idea is to place the initial and final portion of the template around the selected code. For example, you can create a new begin-end block, comment an entire section of code, place a try-finally block around a code fragment, wrap some code in a REGION, and so on. To use a Surround template you have to select a block of code and use the entries in the Surround submenu in the editor local menu.

The joint use of Block Completion, Live Code Templates, and Surround templates can really change the way you write code in Delphi. And, as I already mentioned, all of this without having to change the way you work, as many of these helpers popup automatically.

Other Minor Editor Changes

There are some other nice but less relevant changes:

- If you right click on a tab of the editor, you can use the new command “Close all other pages”.
- You middle-click (or click the mouse wheel) on a tab of the editor to close it.
- You can move from one method to the next one (or the previous one) using the key combination Ctrl+Alt+Down Arrow and Ctrl+Alt+Up Arrow. Ctrl+Alt+Home and Ctrl+Alt+End bring you to the first and last method. This is called Class Navigation. You can also restrict moving within the current class by activating the Class Lock, press Ctrl+Q and then L (keeping Ctrl pressed).

The UML Designer

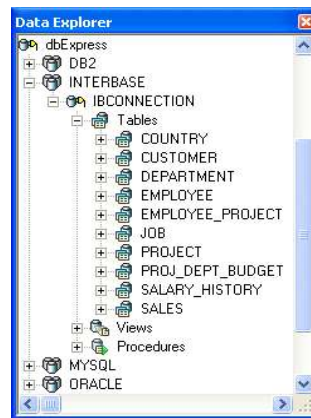
The UML designer (that is, the Together designer embedded in Delphi 2006) has been extended in a very relevant way compared to the 2005 versions. In particular, it now features bi-directional *live code synchronization*: you can edit the source code and see the diagram reflect the changes right away, but you can also edit the diagram to update the source code.

You can add methods, properties, fields, create new classes, inherit new classes from existing ones, change the base class of an existing class, implement interfaces, and more. Together in Delphi 2006 is a full-blown tool, not a cut-down version as in the past.

This is why, beside class diagrams, you can also work on many other UML diagrams: Use Case, Sequence, Collaboration, State Chart, and more. Together has its own set of refactorings (see the update to Chapter 11, later in this document), but features also support for design patterns, and code navigation. You can use Together to generate complete documentation for the project, to check the source code consistency with audits, and to extract the project metrics. All this goes well beyond what I can cover in this book update.

IDE's Database Management Tools

Compared to what I discussed in the corresponding section of the book on Delphi 2005 (page 37), the integrated database management tools of the Delphi 2006 IDE have been extended to support the dbExpress technology, a much better solution for developers involved with Win32 client/server applications. Here you can see an example of the information available in this window:



Like for the BPD drivers, you can access metadata, perform free query, copy database data, and the like. Further information in Chapter 14, which covers the dbExpress architecture.

A New Personality: C++

One of the relevant new features of Delphi 21006 (or better, of Borland Developer Studio 2005) is that the Delphi and C# languages are now complemented by the C++ language. There is a new version of the Borland compiler for this language, a new version of the VCL library for C++ (available in the past in C++Builder 6), new C++ specific libraries, CORBA support, and much more. Compared to C++Builder 6, the IDE

and the libraries have been largely extended. Notice, though, the the C++ support is for Win32 only and not for the .NET architecture of for other operating systems.

The Debugger in Delphi 2006

The IDE has a number of new features related with the debugger, that I want to mention shortly considering that the book doesn't cover debugging at all. First of all, Borland has reintroduced the “Remote Debugger” for Win32 (like there was in Delphi 7 but not in Delphi 2005) which can now be used also for ASP.NET applications.

Another new feature is the object data navigation capability of most debug views (including watches, inspector, and even the fly-by evaluation hint). You can also sort modules in the Module view, do cut and paste operations in the CPU View, and (the most intuitive feature of all) let the editor automatically close all of the source code files opened during a debugging section. This is one of my favorite features of the entire Delphi 2006 IDE.