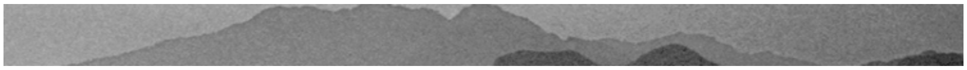
MASTERING™ DELPHI™ 6

Marco Cantù

SYBEX®

MASTERING DELPHI 6



Mastering™ Delphi™ 6

Marco Cantù

SYBEX®

San Francisco • Paris • Düsseldorf • Soest • London

Associate Publisher: Richard Mills
Contracts and Licensing Manager: Kristine O'Callaghan
Acquisitions Editor: Denise Santoro Lincoln
Developmental Editors: Diane Lowery and Denise Santoro Lincoln
Editor: Pete Gaughan
Production Editor: Leslie E. H. Light
Technical Editors: Danny Thorpe and Eddie Churchill
Book Designer: Robin Kibby
Graphic Illustrator: Tony Jonick
Electronic Publishing Specialist: Kris Warrenburg, Cyan Design
Proofreaders: Nanette Duffy, Amey Garber, Jennifer Greiman, Emily Hsuan, Laurie O'Connell, Nancy Riddiough
Indexer: Ted Laux
CD Coordinator: Christine Harris
CD Technician: Kevin Ly
Cover Designer: Design Site
Cover Illustrator/Photographer: Sergie Loobkoff

Library of Congress Card Number: 2001088115
ISBN: 0-7821-2874-2

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

To Lella, the love of my life,
and Benedetta, our love come to life.

Acknowledgments

This edition of *Mastering Delphi* marks the seventh year of the Delphi era, as it took Borland two years to release the latest incarnation of Delphi (along with its Linux twin, Kylix). As it has for many other programmers, Delphi has been my primary interest throughout these years; and writing, consulting, teaching, and speaking at conferences about Delphi have absorbed more and more of my time, leaving other languages and programming tools in the dust of my office. Because my work and my life are quite intertwined, many people have been involved in both, and I wish I had enough space and time to thank them all as they deserve. Instead, I'll just mention a few particular people and say a warm "Thank You" to the entire Delphi community (especially for the Spirit of Delphi 1999 Award I've been happy to share with Bob Swart).

The first official thanks are for the Borland programmers and managers who made Delphi possible and continue to improve it: Chuck Jazdzewski, Danny Thorpe, Eddie Churchill, Allen Bauer, Steve Todd, Mark Edington, Jim Tierney, Ravi Kumar, Jörg Weingarten, Anders Ohlsson, and all the others I have not had a chance to meet. I'd also like to give particular mention to my friends Ben Riga (the current Delphi product manager), John Kaster and David Intersimone (at Borland's Developer Relations), and others who have worked at Borland, including Charlie Calvert, Zack Urlocker and Nan Borreson.

The next thanks are for the Sybex editorial and production crew, many of whom I don't even know. Special thanks go to Pete Gaughan, Leslie Light, Denise Santoro Lincoln, and Diane Lowery; I'd also like to thank Richard Mills, Kristine O'Callaghan, and Kris Warrenburg.

This edition of *Mastering Delphi* has once again had an incredibly picky and detailed review from Delphi R&D team member Danny Thorpe. His highlights and comments in this and past editions have improved the book in all areas: technical content, accuracy, examples, and even readability. Thanks a lot. Previous editions also had special contributions: Tim Gooch worked on Part V for *Mastering Delphi 4*, and Giuseppe Madaffari contributed database material for the Delphi 5 edition. For this edition, Guy Smith-Ferrier rewrote the chapter on ADO, and Nando Dessena helped me with the InterBase chapter. Many improvements to the text and sample programs were suggested by technical reviewers of past editions (Juancarlo Añez, Ralph Friedman, Tim Gooch, and Alain Tadros) and in other reviews over the years by Bob Swart, Giuseppe Madaffari, and Steve Tendon.

Special thanks go to my friends Bruce Eckel, Andrea Provaglio, Norm McIntosh, Johanna and Phil of the BUG-UK, Ray Konopka, Mark Miller, Cary Jensen, Chris Frizelle of *The Delphi Magazine*, Foo Say How, John Howe, Mike Orriss, Chad "Kudzu" Hower, Dan Miser, Marco Miotti, and the entire D&D Team (Paolo, Andrea, Uberto, Nando, Giuseppe, and Mr. Coke). Also, a very big "Thank You" to all the attendees of my Delphi programming courses, seminars, and conferences in Italy, the United States, France, the United Kingdom, Singapore, the Netherlands, Germany, Sweden…

My biggest thanks go to my wife Lella who had to endure yet another many-months-long book-writing session and too many late nights (after spending the evenings with our daughter, Benedetta—I'll thank her with a hug, as Daddy's book looks quite boring to her). Many of our friends (and their kids) provided healthy breaks in the work: Sandro and Monica with Luca, Stefano and Elena, Marco and Laura with Matteo, Bianca, Luca and Elena with Tommaso, Chiara and Daniele with Leonardo, Laura, Vito and Marika with Sofia. Our parents, brothers, sisters, and their families were very supportive, too. It was nice to spend some of our free time with them and our six nephews—Matteo, Andrea, Giacomo, Stefano, Andrea, and Pietro.

Finally, I would like to thank all of the people, many of them unknown, who enjoy life and help to build a better world. If I never stop believing in the future and in peace, it is also because of them.

INTRODUCTION

The first time Zack Urlocker showed me a yet-to-be-released product code-named Delphi, I realized that it would change my work—and the work of many other software developers. I used to struggle with C++ libraries for Windows, and Delphi was and still is the best combination of object-oriented programming and visual programming for Windows.

Delphi 6 simply builds on this tradition and on the solid foundations of the VCL to deliver another astonishing and all-encompassing software development tool. Looking for database, client/server, multitier, intranet, or Internet solutions? Looking for control and power? Looking for fast productivity? With Delphi 6 and the plethora of techniques and tips presented in this book, you'll be able to accomplish all this.

Six Versions and Counting

Some of the original Delphi features that attracted me were its form-based and object-oriented approach, its extremely fast compiler, its great database support, its close integration with Windows programming, and its component technology. But the most important element was the Object Pascal language, which is the foundation of everything else.

Delphi 2 was even better! Among its most important additions were these: the Multi-Record Object and the improved database grid, OLE Automation support and the variant data type, full Windows 95 support and integration, the long string data type, and Visual Form Inheritance. Delphi 3 added to this the code insight technology, DLL debugging support, component templates, the TeeChart, the Decision Cube, the WebBroker technology, component packages, ActiveForms, and an astonishing integration with COM, thanks to interfaces.

Delphi 4 gave us the AppBrowser editor, new Windows 98 features, improved OLE and COM support, extended database components, and many additions to the core VCL classes, including support for docking, constraining, and anchoring controls. Delphi 5 added to the picture many more improvements of the IDE (too many to list here), extended database support (with specific ADO and InterBase datasets), an improved version of MIDAS with Internet support, the TeamSource version-control tool, translation capabilities, the concept of frames, and new components.

Now Delphi 6 adds to all these features support for cross-platform development with the new Component Library for Cross-Platform (CLX), an extended run-time library, the new dbExpress database engine, Web services and exceptional XML support, a powerful Web development framework, more IDE enhancements, and a plethora of new components and classes, as you'll see in the following pages.

Delphi is a great tool, but it is also a complex programming environment that involves many elements. This book will help you master Delphi programming, including the Object Pascal language, Delphi components (both using the existing ones and developing your own), database and client/server support, the key elements of Windows and COM programming, and Internet and Web development.

You do not need in-depth knowledge of any of these topics to read this book, but you do need to know the basics of Pascal programming. Having some familiarity with Delphi will help you considerably, particularly after the introductory chapters. The book starts covering its topics in depth immediately; much of the introductory material from previous editions has been removed. Some of this material and an introduction to Pascal is available on the companion CD-ROM and on my Web site and can be a starting point if you are not confident with Delphi basics. Each new Delphi 6 feature is covered in the relevant chapters throughout the book.

The Structure of the Book

The book is divided into four parts:

- Part I, "Foundations," introduces new features of the Delphi 6 Integrated Development Environment (IDE) in Chapter 1, then moves to the Object Pascal language and to the run-time library (RTL) and Visual Component Library (VCL), providing both foundations and advanced tips.
- Part II, "Visual Programming," covers standard components, Windows common controls, graphics, menus, dialogs, scrolling, docking, multipage controls, Multiple Document Interface, the Action List and Action Manager architectures, and many other topics. The focus is on both the VCL and CLX libraries. The final chapters discuss the development of custom components and the use of libraries and packages.
- Part III, "Database Programming," covers plain database access, in-depth coverage of the data-aware controls, client/server programming, dbExpress, InterBase, ADO and dbGo, DataSnap (or MIDAS), and the development of custom data-aware controls and data sets.

- Part IV, "Beyond Delphi: Connecting with the World," first discusses COM, OLE Automation, and COM+. Then it moves to Internet programming, covering TCP/IP sockets, Internet protocols and Indy, Web server-side extensions (with WebBroker and WebSnap), XML, and the development of Web services.

As this brief summary suggests, the book covers topics of interest to Delphi users at nearly all levels of programming expertise, from "advanced beginners" to component developers.

In this book, I've tried to skip reference material almost completely and focus instead on techniques for using Delphi effectively. Because Delphi provides extensive online documentation, to include lists of methods and properties of components in the book would not only be superfluous, it would also make it obsolete as soon as the software changes slightly. I suggest that you read this book with the Delphi Help files at hand, to have reference material readily available.

However, I've done my best to allow you to read the book away from a computer if you prefer. Screen images and the key portions of the listings should help in this direction. The book uses just a few conventions to make it more readable. All the source code elements, such as keywords, properties, classes, and functions, appear in `this font`, and code excerpts are formatted as they appear in the Delphi editor, with boldfaced keywords and italic comments and strings.

Free Source Code on CD (and the Web)

This book focuses on examples. After the presentation of each concept or Delphi component, you'll find a working program example (sometimes more than one) that demonstrates how the feature can be used. All told, there are about 300 examples presented in the book. These programs are directly available on the companion CD-ROM. The same material is also available on my Web site (`www.marcocantu.com`), where you'll also find updates and examples from past editions. Inside the back cover of the book, you'll find more information about the CD. Most of the examples are quite simple and focus on a single feature. More complex examples are often built step-by-step, with intermediate steps including partial solutions and incremental improvements.

NOTE Some of the database examples also require you to have the Delphi sample database DBDEMOS installed; it is part of the default Delphi installation. Others require the InterBase EMPLOYEE sample database.

Beside the source code files, the CD hosts the ready-to-use compiled programs. There is also an HTML version of the source code, with full syntax highlighting, along with a complete cross-reference of keywords and identifiers (class, function, method, and property names, among others). The cross-reference is an HTML file, so you'll be able to use your browser to easily find all the programs that use a Delphi keyword or identifier you're looking for (not a full search engine, but close enough).

The directory structure of the sample code is quite simple. Basically, each chapter of the book has its own folder, with a subfolder for each example (e.g., `06\Borders`). In the text, the examples are simply referenced by name (e.g., Borders).

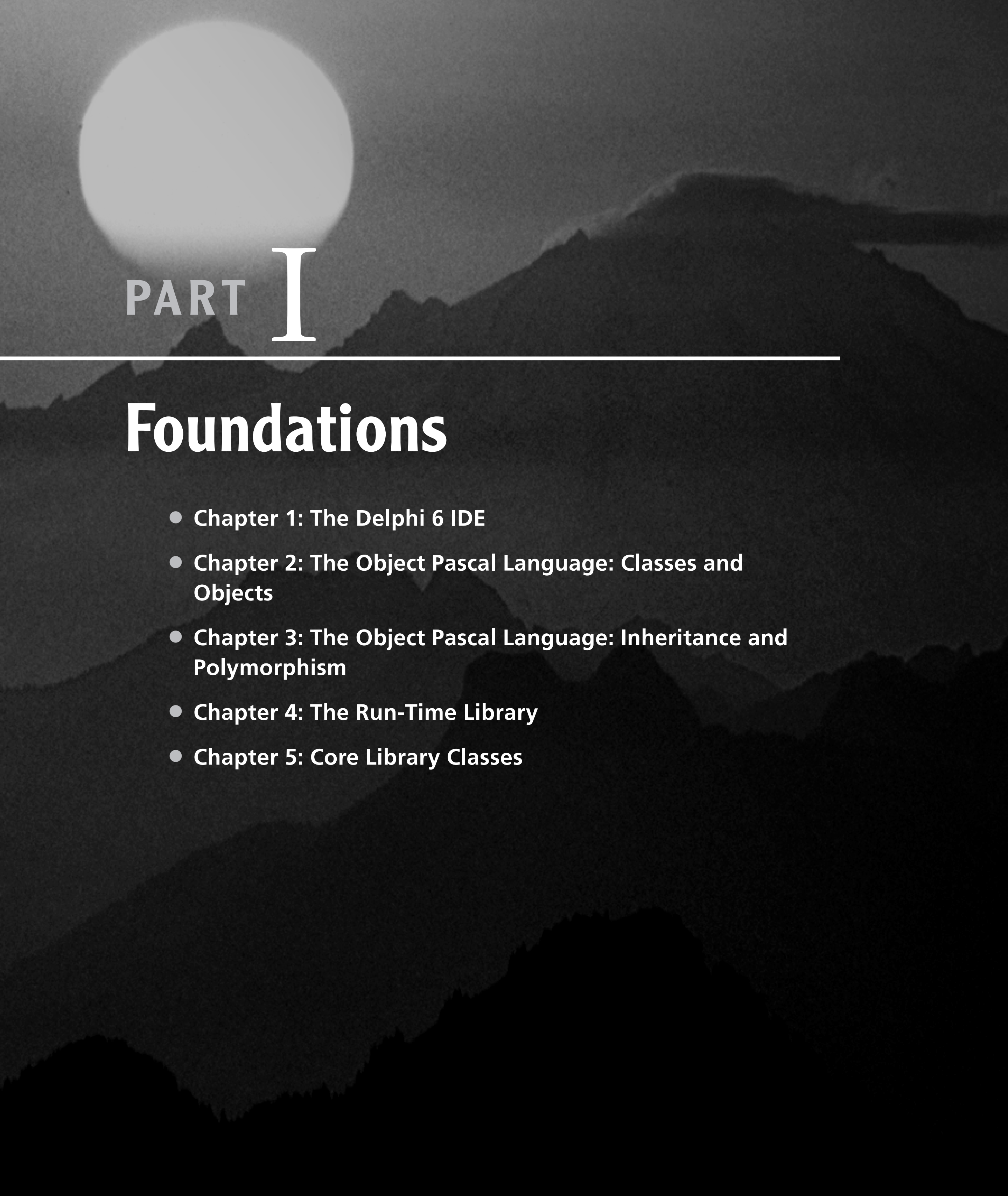
TIP To change an example, first copy it (or the entire `md6code` folder) to your hard disk, but before opening it remember to set the read-only flag to False (it is True by default on the read-only media)

NOTE Be sure to read the source code archive's Readme file, which contains important information about using the software legally and effectively.

How to Reach the Author

If you find any problems in the text or examples in this book, both the publisher and I would be happy to hear from you. Besides reporting errors and problems, please give us your unbiased opinion of the book and tell us which examples you found most useful and which you liked least. There are several ways you can provide this feedback:

- On the Sybex Web site (`www.sybex.com`), you'll find updates to the text or code as necessary. To comment on this book, click the Contact Sybex link and then choose Book Content Issues. This link displays a form where you can enter your comments.
- My own Web site (`www.marcocantu.com`) hosts further information about the book and about Delphi, where you might find answers to your questions. The site has news and tips, technical articles, free online books, white papers, Delphi links, and my collection of Delphi components and tools.
- I have also set up a newsgroup section specifically devoted to my books and to general Delphi Q&A. Refer to my Web site for a list of the newsgroup areas and for the instructions to subscribe to them. (In fact, these newsgroups are totally free but require a login password.) The newsgroups can also be accessed via a Web interface you can find on my site.
- Finally, you can reach me via e-mail at `marco@marcocantu.com`. For technical questions, please try using the newsgroups first, as you might get answers earlier and from multiple people. My mailbox is usually quite full and, regretfully, I cannot reply promptly to every request. (Please write to me in English or Italian.)



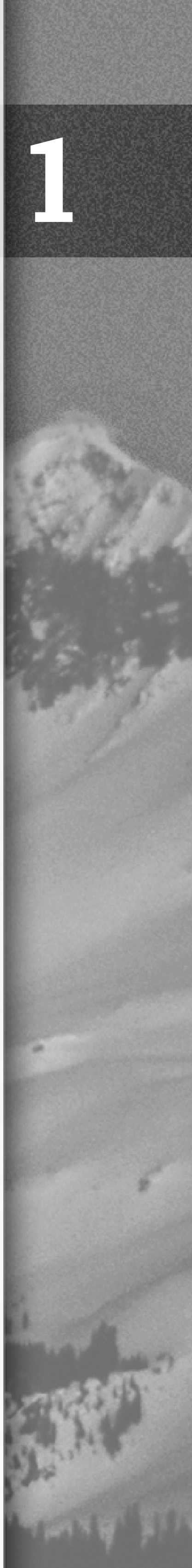
PART I

Foundations

CHAPTER 1

The Delphi 6 IDE

- Object TreeView and Designer view
- The AppBrowser editor
- The code insight technology
- Designing forms
- The Project Manager
- Delphi files



In a visual programming tool such as Delphi, the role of the environment is at times even more important than the programming language. Delphi 6 provides many new features in its visual development environment, and this chapter covers them in detail. This chapter isn't a complete tutorial but mainly a collection of tips and suggestions aimed at the average Delphi user. In other words, it's not for newcomers. I'll be covering the new features of the Delphi 6 Integrated Development Environment (IDE) and some of the advanced and little-known features of previous versions as well, but in this chapter I won't provide a step-by-step introduction. Throughout this book, I'll assume you already know how to carry out the basic hands-on operations of the IDE, and all the chapters after this one focus on programming issues and techniques.

If you *are* a beginning programmer, don't be afraid. The Delphi Integrated Development Environment is quite intuitive to use. Delphi itself includes a manual (available in Acrobat format on the Delphi CD) with a tutorial that introduces the development of Delphi applications. You can also find a step-by-step introduction to the Delphi IDE on my Web site, `http://www.marcocantu.com`. The short online book *Essential Delphi* is based on material from the first chapters of earlier editions of *Mastering Delphi*.

Editions of Delphi 6

Before delving into the details of the Delphi programming environment, let's take a side step to underline two key ideas. First, there isn't a single edition of Delphi; there are many of them. Second, any Delphi environment can be customized. For these reasons, Delphi screens you see illustrated in this chapter may differ from those on your own computer. Here are the current editions of Delphi:

- The "Personal" edition is aimed at Delphi newcomers and casual programmers and has support for neither database programming nor any of the other advanced features of Delphi 6.
- The "Professional" edition is aimed at professional developers. It includes all the basic features, plus database programming support (including ADO support), basic Web server support (WebBroker), and some of the external tools. This book generally assumes you are working with at least the Professional edition.
- The "Enterprise" edition is aimed at developers building enterprise applications. It includes all the new XML and advanced Web services technologies, internationalization, three-tier architecture, and many other tools. Some chapters of this book cover features included only in Delphi Enterprise; these sections are specifically identified.

NOTE In the past, some of the features of Delphi Enterprise have been available as an "up-sell" to owners of Delphi Professional. This might also happen for this version.

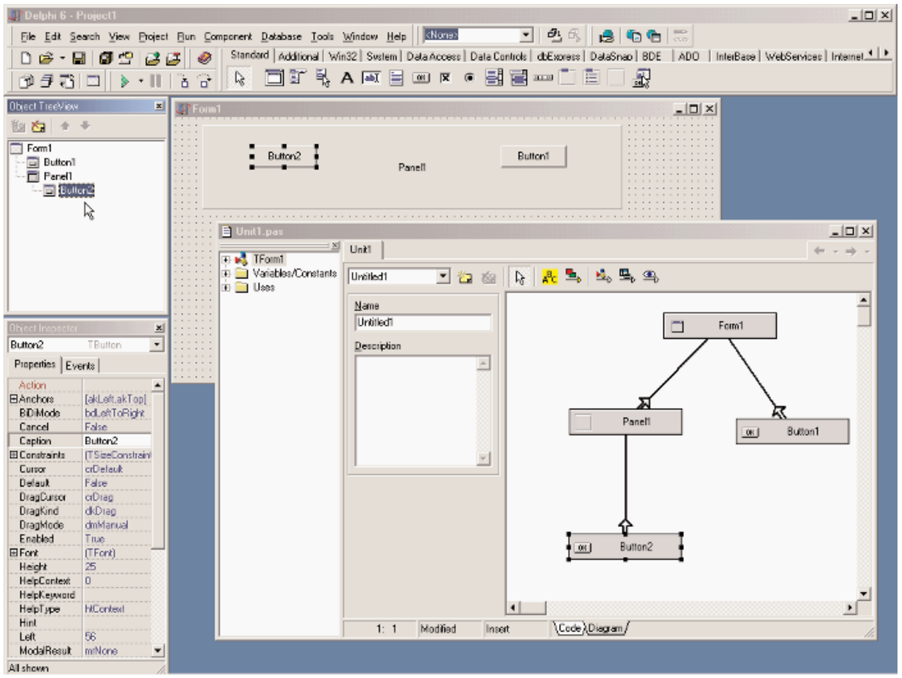
Besides the different editions available, there are ways to customize the Delphi environment. In the screen illustrations throughout the book, I've tried to use a standard user interface (as it comes out of the box); however, I have my preferences, of course, and I generally install many add-ons, which might be reflected in some of the screen shots.

The Delphi 6 IDE

The Delphi 6 IDE includes large and small changes that will really improve a programmer's productivity. Among the key features are the introduction of the Object TreeView for every designer, an improved Object Inspector, extended code completion, and loadable views, including diagrams and HTML.

Most of the features are quite easy to grasp, but it's worth examining them with some care so that you can start using Delphi 6 at its full potential right away. You can see an overall image of Delphi 6 IDE, highlighting some of the new features, in Figure 1.1.

FIGURE 1.1:



Action	
⊟Anchors	[akLeft,akTop]
BiDiMode	bdLeftToRight
Cancel	False
Caption	Button2
⊟Constraints	(TSizeConstraints)
Cursor	crDefault
Default	False
DragCursor	crDrag
DragKind	dkDrag
DragMode	dmManual
Enabled	True
⊟Font	(TFont)
Height	25
HelpContext	0
HelpKeyword	
HelpType	htContext
Hint	
Left	56
ModalResult	mrNone

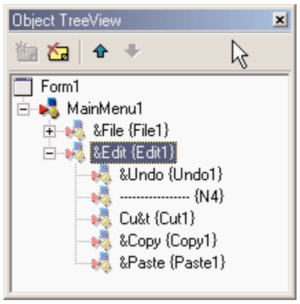
The Delphi 6 IDE: Notice the Object TreeView and the Diagram view.

The Object TreeView

Delphi 5 introduced a TreeView for data modules, where you could see the relations among nonvisual components, such as datasets, fields, actions, and so on. Delphi 6 extends the idea by providing an Object TreeView for every designer, including plain forms. The Object TreeView is placed by default above the Object Inspector; use the View ➢ Object TreeView command in case it is hidden.

The Object TreeView shows all of the components and objects on the form in a tree, representing their relations. The most obvious is the parent/child relation: Place a panel on a form, a button inside it and one outside of the panel. The tree will show the two buttons, one under the form and the other under the panel, as in Figure 1.1. Notice that the TreeView is synchronized with the Object Inspector and Form Designer, so as you select an item and change the focus in any one of these three tools, the focus changes in the other two tools.

Besides parent/child, the Object TreeView shows also other relations, such as owner/owned, component/subobject, collection/item, plus various specific ones, including dataset/connection and data source/dataset relations. Here, you can see an example of the structure of a menu in the tree.



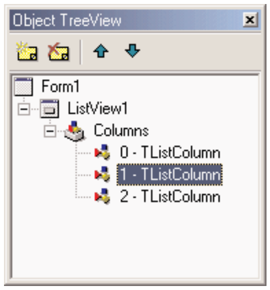
At times, the TreeView also displays “dummy” nodes, which do not correspond to an actual object but do correspond to a predefined one. As an example of this behavior, drop a Table component (from the BDE page) and you’ll see two grayed icons for the session and the alias. Technically, the Object TreeView uses gray icons for components that do not have design-time persistence. They are real components (at design time and at run time), but because they are default objects that are constructed at run time and have no persistent data that can be edited at design time, the Data Module Designer does not allow you to edit their properties. If you drop a Table on the form, you’ll also see items with a red question mark enclosed in a yellow circle next to them. This symbol indicates partially undefined items (there used to be a red square around those items in Delphi 5).

The Object TreeView supports multiple types of *dragging*:

- You can select a component from the palette (by clicking it, not actually dragging it), move the mouse over the tree, and click a component to drop it there. This allows you to drop a component in the proper container (form, panel, and others) regardless of the fact that its surface might be totally covered by other components, something that prevents you from dropping the component in the designer without first rearranging those components.
- You can drag components within the TreeView—for example, moving a component from one container to another—something that, with the Form Designer, you can do only with cut and paste techniques. Moving instead of cutting provides the advantage that if you have connections among components, these are not lost, as happens when you delete the component during the cut operation.
- You can drag components from the TreeView to the Diagram view, as we'll see later.

Right-clicking any element of the TreeView displays a shortcut menu similar to the component menu you get when the component is in a form (and in both cases, the shortcut menu may include items related to the custom component editors). You can even delete items from the tree.

The TreeView doubles also as a collection editor, as you can see here for the `Columns` property of a ListView control. In this case, you can not only rearrange and delete items, but also add new items to the collection.



TIP You can print the contents of the Object TreeView for documentation purposes. Simply select the window and use the File ➢ Print command, as there is no Print command in the shortcut menu.

Loadable Views

Another important change has taken place in the Code Editor window. For any single file loaded in the IDE, the editor can now show multiple views, and these views can be defined programmatically and added to the system, then loaded for given files—hence the name *loadable* views.

The most frequently used view is the Diagram page, which was already available in Delphi 5 data modules, although it was less powerful. Another set of views is available in Web applications, including an HTML Script view, an HTML Result preview, and many others discussed in Chapter 22.

The Diagram View

Along with the TreeView, another feature originally introduced in Delphi 5 Data Modules and now available for every designer is the Diagram view. This view shows dependencies among components, including parent/child relations, ownership, linked properties, and generic relations. For dataset components, it also supports master/detail relations and lookup connections. You can even add your comments in text blocks linked to specific components.

The Diagram is not built automatically. You must drag components from the TreeView to the diagram, which will automatically display the existing relations among the components you drop there. In Delphi 6, you can now select multiple items from the Object TreeView and drag them all at once to the Diagram page.

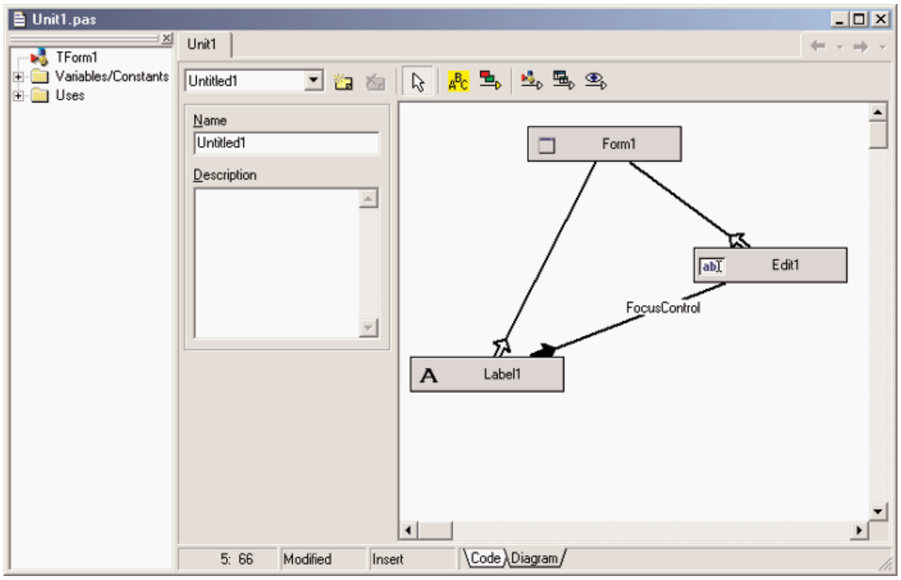
What's nice is that you can set properties by simply drawing arrows between the components. For example, after moving an edit and a label to Diagram, you can select the Property Connector icon, click the label, and drag the mouse cursor over the edit. When you release the mouse button, the Diagram will set up a property relation based on the `FocusControl` property, which is the only property of the label referring to an edit control. This situation is depicted in Figure 1.2.

As you can see, setting properties is *directional*: If you drag the property relation line from the edit to the label, you end up trying to use the label as the value of a property of the edit box. Because this isn't possible, you'll see an error message indicating the problem and offering to connect the components in the opposite way.

In Delphi 6, the Diagram view allows you to create multiple diagrams for each Delphi unit—that is, for each form or data module. Simply give a name to the diagram and possibly add a description, click the New Diagram button, prepare another diagram, and you'll be able to switch back and forth between diagrams using the combo box available in the toolbar of the Diagram view.

FIGURE 1.2:

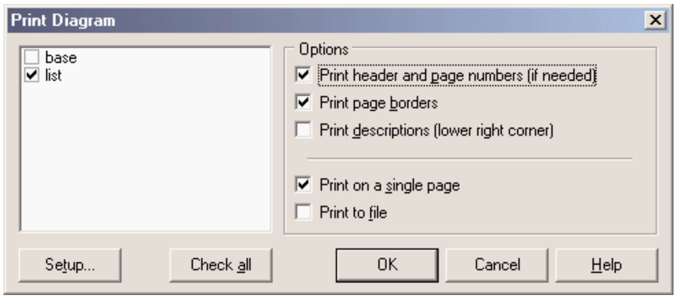
The Diagram view allows you to connect components using the Property connector.



Although you can use the Diagram view to set up relations, its main role is to document your design. For this reason, it is important to be able to print the content of this view. Using the standard File ➢ Print command while the Diagram is active, Delphi prompts you for options, as you can see in Figure 1.3, allowing you to customize the output in many ways.

FIGURE 1.3:

The Print Options for the Diagram view



Print Diagram

- [ ] base
- [✓] list

Options
- [✓] Print header and page numbers (if needed)
- [✓] Print page borders
- [ ] Print descriptions (lower right corner)
- [✓] Print on a single page
- [ ] Print to file

The information in the Data Diagram view is saved in a separate file, not as part of the DFM file. Delphi 5 used design-time information (DTI) files, which had a structure similar to INI files. Delphi 6 can still read the older .DTI format, but uses the new Delphi Diagram Portfolio format (.DDP). These files apparently use the DFM binary format (or a similar one), so they are not editable as text. All of these files are obviously useless at run time (it makes no sense to include them in the compilation of the executable file).

An IDE for Two Libraries

Another very important change I just want to introduce here is the fact that Delphi 6, for the first time, allows you to use to different component libraries, VCL (Visual Components Library) and CLX (Component Library for Cross-Platform). When you create a new project, you simply choose which of the two libraries you want to use, starting with the File ➢ New ➢ Application command for a classic VCL-based Windows program and with the File ➢ New ➢ CLX Application command for a new CLX-based portable application.

Creating a new project or opening an existing one, the Component Palette is rearranged to show only the controls related to the current library (although most of them are actually shared). This topic is fully covered in Chapter 6, so I don't want to get into the details here; I'll just underline that you can use Delphi 6 to build applications you can compile right away for Linux using Kylix. The effect of this change on the IDE is really quite large, as many things "under the hood" had to be reengineered. Only programmers using the ToolsAPI and other advanced elements will notice all these internal differences, as they are mostly transparent to most users.

Smaller Enhancements

Besides this important change and others I'll discuss in later sections, such as the update of the Object Inspector and of code completion, there are small (but still quite important) changes in the Delphi 6 IDE. Here is a list of these changes:

- There is a new Window menu in the IDE. This menu lists the open windows, something you could obtain in the past using the Alt+0 keys. This is really very handy, as windows often end up behind others and are hard to find. (Thanks, Borland, for listening to this and other simple but effective requests from users.)

TIP Two entries of the Main Window registry section of Delphi (under `\Software\Borland\Delphi\6.0` for the current user) allow you to hide this menu and disable its alphabetic sort order. This registry keys use strings (in place of Boolean values) where "-1" indicates true and "0" false.

- The File menu doesn't include specific items for creating new forms or applications. These commands have been increased in number and grouped under the File ➢ New secondary menu. The Other command of this menu opens the New Item dialog box (the Object Repository) as the File ➢ New command did in the past.
- The Component Palette local menu has a submenu listing all of the palette pages in alphabetic order. You can use it to change the active page, particularly when it is not visible on the screen.

TIP The order of the entries in the Tabs submenu of the Component Palette local menu can be set in the same order as the palette itself, and not sorted alphabetically. This is accomplished by setting to "0" (false) the value of the Sort Palette Tabs Menu key of the Main Window registry section of Delphi (under `\Software\Borland\Delphi\6.0` for the current user).

- There is a new toolbar, the Internet toolbar, which is initially disabled. This toolbar supports WebSnap applications.

Updated Environment Options Dialog Box

Quite a few small changes relate to the commonly used Environment Options dialog box. The pages of this dialog box have been rearranged, moving the Form Designer options from the Preferences page to the new Designer page. There are also a few new options and pages:

- The Preferences page of the Environment Options dialog box has a new check box that prevents Delphi windows from automatically docking with each other. This is a very welcome addition!
- A new page, Environment Variables, allows you to see system environment variables (such as the standard path names and OS settings) and set user-defined variables. The nice point is that you can use both system- and user-defined environment variables in each of the dialog boxes of the IDE—for example, you can avoid hard-coding commonly used path names, replacing them with a variable. In other words, the environment variables work similarly to the $DELPHI variable, referring to Delphi's base directory, but can be defined by the user.
- Another new page is called Internet. In this page, you can choose the default file extensions used for HTML and XML files (mainly by the WebSnap framework) and also associate an external editor with each extension.

Delphi Extreme Toys

At times, the Delphi team comes up with small enhancements of the IDE that aren't included in the product because they either aren't of general use or will require time to be improved in quality, user interface, or robustness. Some of these internal wizards and IDE extensions have now been made available, with the collective name of Delphi Extreme Toys, to registered Delphi 6 users. You should automatically get this add-on as you register your copy of the product (online or through a Borland office).

There isn't an official list of the content of the Extreme Toys, as Borland plans to keep extending them. The initial release includes an IDE-based search engine for seeking answers on Delphi across the Internet, a wizard for turning on and off specific compiler warnings, and an "invokamatic" wizard for accelerating the creation of Web services. The Extreme Toys will, in essence, be *unofficial* wizards, code utilities, and components from the Delphi team—or useful stuff from various people.

Recent IDE Additions

Delphi 5 provided a huge number of new features to the IDE. In case you've only used versions of Delphi prior to 5, or need to brush up on some useful added information, this is a short summary of the most important of the features introduced in Delphi 5.

Saving the Desktop Settings

The Delphi IDE allows programmers to customize it in various ways—typically, opening many windows, arranging them, and docking them to each other. However, programmers often need to open one set of windows at design time and a different set at debug time. Similarly, programmers might need one layout when working with forms and a completely different layout when writing components or low-level code using only the editor. Rearranging the IDE for each of these needs is a tedious task.

For this reason, Delphi allows you to save a given arrangement of IDE windows (called a *desktop*) with a name and restore it easily. Also, you can make one of these groupings your default debugging setting, so that it will be restored automatically when you start the debugger. All these features are available in the Desktops toolbar. You can also work with desktop settings using the View ➢ Desktops menu.

Desktop setting information is saved in DST files, which are INI files in disguise. The saved settings include the position of the main window, the Project Manager, the Alignment Palette, the Object Inspector (including its new property category settings), the editor windows (with the status of the Code Explorer and the Message View), and many others, plus the docking status of the various windows.

Here is a small excerpt from a DST file, which should be easily readable:

```
[Main Window]
Create=1
Visible=1
State=0
Left=0
Top=0
Width=1024
Height=105
ClientWidth=1016
ClientHeight=78
```

```
[ProjectManager]
Create=1
Visible=0
State=0
...
Dockable=1

[AlignmentPalette]
Create=1
Visible=0
...
```

Desktop settings override project settings. This helps eliminate the problem of moving a project between machines (or between developers) and having to rearrange the windows to your liking. Delphi 5 separates per-user and per-machine preferences from the project settings, to better support team development.

TIP

If you open Delphi and cannot see the form or other windows, I suggest you try checking (or deleting) the desktop settings. If the project desktop was last saved on a system running in a high-resolution video mode (or a multimonitor configuration) and opened on a different system with lower screen resolution or fewer monitors, some of the windows in the project might be located off-screen on the lower-resolution system. The simplest ways to fix that are either to load your own named desktop configuration after opening the project, thus overriding the project desktop settings, or just delete the DST file that came with the project files.

The To-Do List

Another feature added in Delphi 5 was the to-do list. This is a list of tasks you still have to do to complete a project, a collection of notes for the programmer (or programmers, as this tool can be very handy in a team). While the idea is not new, the key concept of the to-do list in Delphi is that it works as a two-way tool.

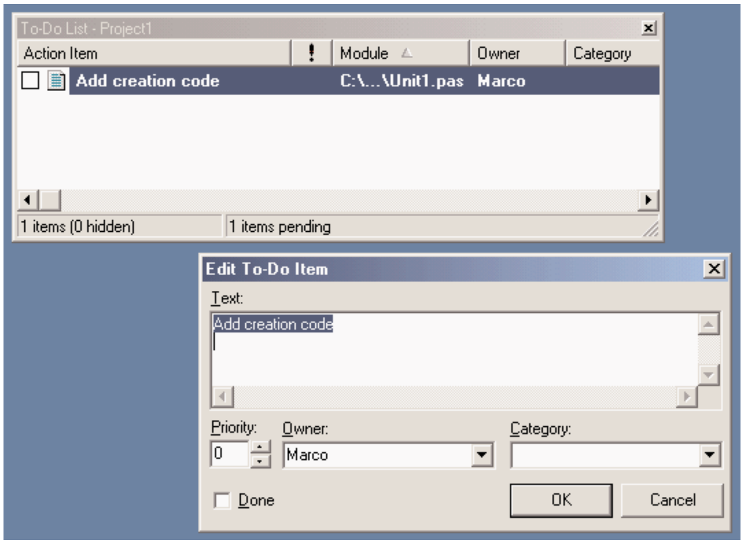
In fact, you can add or modify to-do items by adding special `TODO` comments to the source code of any file of a project; you'll then see the corresponding entries in the list. But you can also visually edit the items in the list to modify the corresponding source code comment. For example, here is how a to-do list item might look like in the source code:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // TODO -oMarco: Add creation code
end;
```

The same item can be visually edited in the window shown in Figure 1.4.

FIGURE 1.4:

The Edit To-Do Item window can be used to modify a to-do item, an operation you can also do directly in the source code.



To-Do List - Project1

Action Item	!	Module	Owner	Category
Add creation code		C:\...\Unit1.pas	Marco	

1 items (0 hidden) 1 items pending

Edit To-Do Item

Text: Add creation code

Priority: 0 Owner: Marco Category:

Done OK Cancel

The exception to this two-way rule is the definition of project-wide to-do items. You must add these items directly to the list. To do that, you can either use the Ctrl+A key combination in the To-Do List window or right-click in the window and select Add from the shortcut menu. These items are saved in a special file with the .TODO extension.

You can use multiple options with a `TODO` comment. You can use `–o` (as in the code excerpt above) to indicate the owner, the programmer who entered the comment; the `–c` option to indicate a category; or simply a number from 1 to 5 to indicate the priority (0, or no number, indicates that no priority level is set). For example, using the Add To-Do Item command on the editor's shortcut menu (or the Ctrl+Shift+T shortcut) generated this comment:

```
{ TODO 2 -oMarco : Button pressed }
```

Delphi treats everything after the colon, up to the end of line or the closing brace, depending on the type of comment, as the text of the to-do item. Finally, in the To-Do List window you can check off an item to indicate that it has been done. The source code comment will change from `TODO` to `DONE`. You can also change the comment in the source code manually to see the check mark appear in the To-Do List window.

One of the most powerful elements of this architecture is the main To-Do List window, which can automatically collect to-do information from the source code files as you type them, sort and filter them, and export them to the Clipboard as plain text or an HTML table.

The AppBrowser Editor

The editor included with Delphi hasn't changed recently, but it has many features that many Delphi programmers don't know and use. It's worth briefly examining this tool. The Delphi editor allows you to work on several files at once, using a "notebook with tabs" metaphor, and you can also open multiple editor windows. You can jump from one page of the editor to the next by pressing Ctrl+Tab (or Shift+Ctrl+Tab to move in the opposite direction).

TIP In Delphi 6, you can drag-and-drop the tabs with the unit names in the upper portion of the editor to change their order, so that you can use a single Ctrl+Tab to move between the units you are mostly interested in. The local menu of the editor has also a Pages command, which lists all of the available pages in a submenu, a handy feature when many units are loaded.

Several options affect the editor, located in the new Editor Properties dialog box. You have to go to the Preferences page of the Environment Options dialog box, however, to set the editor's AutoSave feature, which saves the source code files each time you run the program (preventing data loss in case the program crashes badly).

I won't discuss the various settings of the editor, as they are quite intuitive and are described in the online Help. A tip to remember is that using the Cut and Paste commands is not the only way to move source code. You can also select and drag words, expressions, or entire lines of code. You can also copy text instead of moving it, by pressing the Ctrl key while dragging.

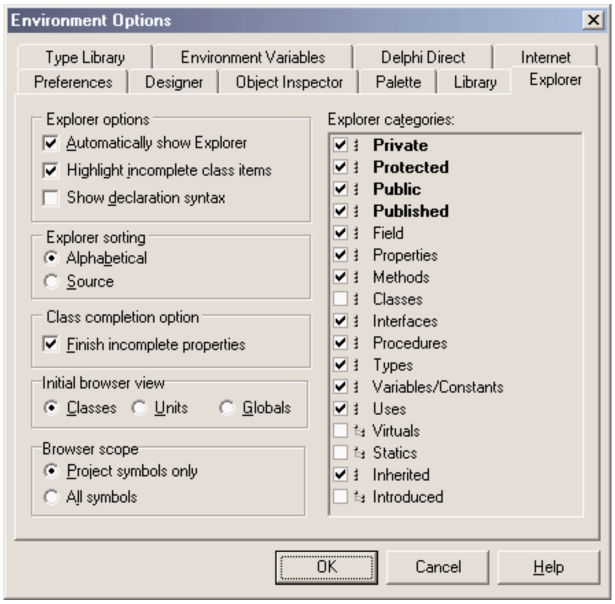
The Code Explorer

The Code Explorer window, which is generally docked on the side of the editor, simply lists all of the types, variables, and routines defined in a unit, plus other units appearing in `uses` statements. For complex types, such as classes, the Code Explorer can list detailed information, including a list of fields, properties, and methods. All the information is updated as soon as you start typing in the editor. You can use the Code Explorer to navigate in the editor. If you double-click one of the entries in the Code Explorer, the editor jumps to the corresponding declaration.

TIP In Delphi 6 you can modify variables, properties, and method names directly in the Code Explorer.

While all that is quite obvious after you've used Delphi for a few minutes, some features of the Code Explorer are not so intuitive. One important point is that you have full control of the layout of the information, and you can reduce the depth of the tree usually displayed in this window by customizing the Code Explorer. Collapsing the tree can help you make your selections more quickly. You can configure the Code Explorer by using the corresponding page of the Environment Options, as shown in Figure 1.5.

FIGURE 1.5:
You can configure the Code Explorer in the Environment Options dialog box.



Explorer options
- [✓] Automatically show Explorer
- [✓] Highlight incomplete class items
- [ ] Show declaration syntax

Explorer sorting
- (●) Alphabetical
- ( ) Source

Class completion option
- [✓] Finish incomplete properties

Initial browser view
- (●) Classes ( ) Units ( ) Globals

Browser scope
- (●) Project symbols only
- ( ) All symbols

Explorer categories:
- [✓] **Private**
- [✓] **Protected**
- [✓] **Public**
- [✓] **Published**
- [✓] Field
- [✓] Properties
- [✓] Methods
- [ ] Classes
- [✓] Interfaces
- [✓] Procedures
- [✓] Types
- [✓] Variables/Constants
- [✓] Uses
- [ ] Virtuals
- [ ] Statics
- [✓] Inherited
- [ ] Introduced

Notice that when you deselect one of the Explorer Categories items on the right side of this page of the dialog box, the Explorer doesn't remove the corresponding elements from view—it simply adds the node in the tree. For example, if you deselect the Uses check box, Delphi doesn't hide the list of the used units from the Code Explorer. On the contrary, the used units are listed as main nodes instead of being kept in the `Uses` folder. I generally disable the Types, Classes, and Variables selections.

Because each item of the Code Explorer tree has an icon marking its type, arranging by field and method seems less important than arranging by access specifier. My preference is to show all items in a single group, as this requires the fewest mouse clicks to reach each item. Selecting items in the Code Explorer, in fact, provides a very handy way of navigating the source code of a large unit. When you double-click a method in the Code Explorer, the focus moves to the definition in the class declaration (in the interface portion of the unit). You can use the Ctrl+Shift combination with the Up or Down arrow keys to jump from the definition of a method or procedure in the interface portion of a unit to its complete definition in the implementation portion (or back again).

NOTE Some of the Explorer Categories shown in Figure 1.5 are used by the Project Explorer, rather than by the Code Explorer. These include, among others, the Virtuals, Statics, Inherited, and Introduced grouping options.

Browsing in the Editor

Another feature of the AppBrowser editor is *Tooltip symbol insight*. Move the mouse over a symbol in the editor, and a Tooltip will show you where the identifier is declared. This feature can be particularly important for tracking identifiers, classes, and functions within an application you are writing, and also for referring to the source code of the Visual Component Library (VCL).

WARNING Although it may seem a good idea at first, you cannot use Tooltip symbol insight to find out which unit declares an identifier you want to use. If the corresponding unit is not already included, in fact, the Tooltip won't appear.

The real bonus of this feature, however, is that you can turn it into a navigational aid. When you hold down the Ctrl key and move the mouse over the identifier, Delphi creates an active link to the definition instead of showing the Tooltip. These links are displayed with the blue color and underline style that are typical of Web browsers, and the pointer changes to a hand whenever it's positioned on the link.

For example, you can Ctrl+click the `TLabel` identifier to open its definition in the VCL source code. As you select references, the editor keeps track of the various positions you've jumped to, and you can move backward and forward among them—again as in a Web browser. You can also click the drop-down arrows near the Back and Forward buttons to view a detailed list of the lines of the source code files you've already jumped to, for more control over the backward and forward movement.

How can you jump directly to the VCL source code if it is not part of your project? The AppBrowser editor can find not only the units in the Search path (which are compiled as part of the project), but also those in Delphi's Debug Source, Browsing, and Library paths. These directories are searched in the order I've just listed, and you can set them in the Directories/Conditionals page of the Project Options dialog box and in the Library page of the Environment Options dialog box. By default, Delphi adds the VCL source code directories in the Browsing path of the environment.

Class Completion

The third important feature of Delphi's AppBrowser editor is *class completion*, activated by pressing the Ctrl+Shift+C key combination. Adding an event handler to an application is a fast operation, as Delphi automatically adds the declaration of a new method to handle the event in the class and provides you with the skeleton of the method in the implementation portion of the unit. This is part of Delphi's support for visual programming.

Newer versions of Delphi also simplify life in a similar way for programmers who write a little extra code behind event handlers. The new code-generation feature, in fact, applies to general methods, message-handling methods, and properties. For example, if you type the following code in the class declaration:

```
public
  procedure Hello (MessageText: string);
```

and then press Ctrl+Shift+C, Delphi will provide you with the definition of the method in the implementation section of the unit, generating the following lines of code:

```
{ TForm1 }
procedure TForm1.Hello(MessageText: string);
begin
end;
```

This is really handy, compared with the traditional approach of many Delphi programmers, which is to copy and paste one or more declarations, add the class names, and finally duplicate the `begin...end` code for every method copied.

Class completion can also work the other way around. You can write the implementation of the method with its code directly, and then press Ctrl+Shift+C to generate the required entry in the class declaration.

Code Insight

Besides the Code Explorer, class completion, and the navigational features, the Delphi editor supports the *code insight* technology. Collectively, the code insight techniques are based on a constant background parsing, both of the source code you write and of the source code of the system units your source code refers to.

Code insight comprises five capabilities: code completion, code templates, code parameters, Tooltip expression evaluation, and Tooltip symbol insight. This last feature was already covered in the section "Browsing in the Editor"; the other four will be discussed in the following subsections. You can enable, disable, and configure each of these features in the Code Insight page of the Editor Options dialog box.

Code Completion

Code completion allows you to choose the property or method of an object simply by looking it up on a list or by typing its initial letters. To activate this list, you just type the name of an object, such as `Button1`, then add the dot, and wait. To force the display of the list, press Ctrl+spacebar; to remove it when you don't want it, press Esc. Code completion also lets you look for a proper value in an assignment statement.

In Delphi 6, as you start typing, the list filters its content according to the initial portion of the element you've inserted. The code completion list uses colors and shows more details to help you distinguish different items. Another new feature is that in the case of functions with parameters, parentheses are included in the generated code, and the parameters list hint is displayed immediately.

As you type `:=` after a variable or property, Delphi will list all the other variables or objects of the same type, plus the objects having properties of that type. While the list is visible, you can right-click it to change the order of the items, sorting either by scope or by name, and you can also resize the window.

In Delphi 6, code completion also works in the interface section of a unit. If you press Ctrl+spacebar while the cursor is inside the class definition, you'll get a list of: virtual methods you can override (including abstract methods), the methods of implemented interfaces, the base class properties, and eventually system messages you can handle. Simply selecting one of them will add the proper method to the class declaration. In this particular case, the code completion list allows multiple selection.

TIP

When the code you've written is incorrect, code insight won't work, and you may see just a generic error message indicating the situation. It is possible to display specific code insight errors in the Message pane (which must already be open; it doesn't open automatically to display compilation errors). To activate this feature, you need to set an undocumented registry entry, setting the string key `\Delphi\6.0\Compiling\ShowCodeInsiteErrors` to the value '1'.

There are advanced features of Delphi 6 code completion that aren't easy to spot. One that I found particularly useful relates to the discovery of symbols in units not used by your project. As you invoke it (with Ctrl+spacebar) over a blank line, the list also includes symbols from common units (such as Math, StrUtils, and DateUtils) not already included in the `uses` statement of the current one. By selecting one of these *external* symbols, Delphi adds the unit to the `uses` statement for you. This feature (which doesn't work inside expressions) is driven by a customizable list of extra units, stored in the registry key `\Delphi\6.0\CodeCompletion\ExtraUnits`.