



Quick Start



Borland[®]
Delphi[™] 4
for Windows 95 and Windows NT

Inprise Corporation, 100 Enterprise Way
Scotts Valley, CA 95066-3249

Refer to the file DEPLOY.TXT located in the root directory of your Delphi 4 product for a complete list of files that you can distribute in accordance with the No-Nonsense License Statement.

Inprise may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1998 Inprise Corporation. All rights reserved. All Inprise and Borland products are trademarks or registered trademarks of Inprise Corporation. Other brand and product names are trademarks or registered trademarks of their respective holders.

Printed in the U.S.A.

HDA1340WW21000 1E0R598

9899000102-9 8 7 6 5 4 3 2 1

D4

Contents

Chapter 1

Introduction 1-1

What is Delphi?	1-1
Where to find information	1-1
Online Help	1-2
Printed documentation	1-4
Inprise developer support services	1-4
Inprise Web site	1-4
Manual conventions	1-4

Chapter 2

A tour of the environment 2-1

Starting Delphi	2-1
Placing components on a form	2-2
Changing component appearance and behavior	2-3
Navigating among events	2-4
Editing code	2-4
Browsing with the editor	2-5
Navigating within your project	2-5
Navigating within your code	2-6
Getting help while coding	2-7
Organizing the environment	2-8
Debugging applications	2-9
Exploring databases	2-11
Storing objects as templates	2-11
Project management tool	2-13
Handy pop-up menus	2-14
Toolbars	2-14
Getting help	2-15

Chapter 3

Your first application—a brief tutorial 3-1

Starting a new application	3-1
Setting property values	3-3
Adding objects to the form	3-3
Accessing a database	3-5
Adding support for a toolbar and a menu	3-7
Adding a menu	3-8
Adding a toolbar	3-10
Displaying an image	3-11
Final touches	3-12
Hooking up an event handler	3-13

Chapter 4

Customizing the environment 4-1

Organizing your work area	4-1
Organizing tools	4-2
Organizing menus and toolbars	4-4
Setting project options	4-5
Setting options for all new projects	4-6
Restoring Delphi's original default settings	4-6
Creating project defaults	4-6
Specifying a default project	4-6
Displaying a default form	4-7
Setting tool preferences	4-7
Customizing the Code editor	4-8
Customizing the Component palette	4-8
Rearranging the Component palette	4-9
Adding components to Delphi	4-10
Installing component packages	4-10
Adding ActiveX controls	4-10
Creating component templates	4-11
Customizing Delphi Help	4-11

Chapter 5

Programming with Delphi 5-1

Delphi development environment	5-1
Designing applications	5-2
Using the VCL	5-2
Creating the application user interface	5-4
Using components	5-4
Changing component behavior	5-5
Designing menus	5-5
Developing applications	5-6
Creating Windows GUI applications	5-6
Creating packages and DLLs	5-7
Handling exceptions	5-7
Writing database applications	5-8
Connecting to databases	5-8
Using database tools	5-8
Browsing databases	5-9
Storing data information	5-9
Editing existing database tables	5-9
Configuring databases	5-10
Understanding database application architecture	5-10

Developing distributed applications.5-11

Developing CORBA applications.5-12

Developing distributed applications
using COM and MTS.5-12

Creating COM applications with
wizards5-13

Using MTS with Delphi. 5-13

Creating and editing type libraries 5-14

Deploying applications. 5-14

Building custom components 5-15

Index

I-1

Tables

1.1 Online Help documentation 1-2

1.2 Typefaces 1-4

2.1 Code Insight tools 2-7

4.1 Project options4-5

5.1 Delphi product versions5-2

1

Introduction

Welcome to Delphi! This *Quick Start* provides an overview of the Delphi development environment and features to get you started using the product right away. It also tells you where to look for details on using the product and the many tools that are available from within the Delphi environment.

What is Delphi?

Delphi is an object-oriented, visual programming environment for rapid application development (RAD). You can use it for developing all kinds of applications from general-purpose utilities to sophisticated data access programs, including client/server applications. Using Delphi, you can create highly efficient Microsoft Windows 95, Windows 98, and Windows NT applications with a minimum of manual coding.

Delphi provides a comprehensive library of reusable components and a suite of RAD design tools, including application and form templates, and programming wizards. These tools simplify application prototyping and development and shorten development time.

When you start Delphi, you are immediately placed within the visual programming environment. It is within this environment that Delphi provides all the tools you need to design, develop, test, and debug applications.

Where to find information

Extensive information on Delphi is available in a variety of forms:

- Online Help
- Printed documentation
- Inprise developer support services
- Inprise Web site

Refer to What’s New in Delphi? in online Help and the Inprise Web site for information about new features for this release.

Online Help

You can use online Help to get detailed information about using Delphi while using the product. Referring to Help is a convenient way to learn about the extensive language features, programming tasks, compiler options, and other development tools.

Delphi provides detailed online reference information for numerous reusable components such as the Visual Component Library and Win32 APIs. Online reference is the fastest way to get information on language syntax and component properties, methods, and events as you are developing applications.

Delphi includes the following Help files:

Table 1.1 Online Help documentation

Help file	What’s it about?	Who’s it for?
Using Delphi (Delphi4.hlp)	Describes how to use the Delphi development environment and explains fundamental concepts such as components, properties, methods, and events, and working with forms, projects, and packages.	New Delphi developers, people with questions about the IDE
Visual Component Library Reference (Del4vcl.hlp)	Presents detailed reference on Delphi objects, components, global routines, types, and variables. Individual entries for objects and components include the unit where each is declared, show the position in the hierarchy, list all available properties and methods, and include relevant examples.	All Delphi developers
Programming with Delphi (Del4prog.hlp)	Provides programming details for Delphi developers using components available in the VCL and illustrates the implementation of common programming tasks such as handling exceptions, creating toolbars, cool bars, drag-and-drop controls, and using graphics.	All Delphi developers for detailed programming examples
Developing Database Applications (Del4dbd.hlp)	Discusses issues for developers who are designing single and multi-tiered database applications including background on database architecture, datasets, fields, tables, queries, and decision support.	Database developers
Developing Distributed Applications (Del4dap.hlp)	Supplies information on using Delphi for creating distributed applications including information on CORBA, DCOM, MTS, HTTP, and Sockets.	Developers writing Client/server applications
Creating Custom Components (Del4cw.hlp)	Provides information for developers who want to write customized components including how to design, build, test, and install the component into the Delphi environment.	Developers who are implementing components

Table 1.1 Online Help documentation (continued)

Help file	What's it about?	Who's it for?
Developing COM-based Applications (Del4com.hlp)	Describes the concepts and skills necessary for building distributed applications, including COM objects, ActiveX controls and automation objects. Describes writing COM objects for the MTS runtime environment. Covers Delphi's Type Library Editor, which provides an easy way to modify the automatically generated type library.	Developers writing client/server applications
Object Pascal Reference (Del4op.hlp)	Provides a formal definition of the Object Pascal language as used in Delphi and includes topics such as file I/O, string manipulation, program control, data types, and language extensions.	Developers who need Object Pascal language details
Customizing Help (OpenHelp.hlp)	Explains how to use OpenHelp to configure your Windows Help (.HLP) files to include or remove Help files from the Help system.	Developers wanting to customize the Delphi Help system
What's New in Delphi? (Del4new.hlp)	Introduces the new features and enhancements to Delphi for the current release and includes links to detailed information.	Developers who upgraded to this release

You will also find Help on additional products that are supplied with some versions of Delphi, such as the following:

- Borland Database Engine Help
- Borland Database Engine Administrator Help
- Database Explorer Help
- PVCS Version Manager Help
- Local SQL Help
- SQL Builder Help
- WinSight User's Guide Image Editor Help
- Win32 Help files
- Help Author's Guide (Help Workshop)
- NEWT Intranet ActiveX Help
- QuickReport Help
- TeeChart Help
- InterBase Help
- Help for miscellaneous components (FastNet Time, DayTime, Echo, Finger, HTTP, NNTP, POP3, Powersock, SMTP, UDP, URL Encode/Decode, UUprocessor, Stream and Msg components)

The Help files are located in the Help directory under the main Delphi directory.

Printed documentation

This *Quick Start* is meant to get you started with using Delphi. For information about ordering additional printed documentation, refer to the Inprise Web site at www.inprise.com.

Inprise developer support services

Inprise also offers many support options if you need more information. To find out about Inprise's developer support services, see Inprise Online at <http://www.inprise.com/devsupport>, or call Inprise Assist at (800) 523-7070.

Inprise Web site

Additional Delphi Technical Information documents and Frequently Asked Questions (FAQs) are available through Inprise Online at www.inprise.com. The Technical Information documents are white papers that focus on particular aspects of Delphi.

From the Web site, you can access many newsgroups where Delphi developers exchange information, tips, and techniques. The site also includes a useful bibliography of additional books published on Delphi.

Manual conventions

This manual uses the typefaces described in Table 1.2 to indicate special text.

Table 1.2 Typefaces

Typeface	Meaning
Monospace type	Monospaced text represents text as it appears on screen or in code. It also represents anything you must type.
Boldface	Boldfaced words in text or code listings represent reserved words or compiler options.
<i>Italics</i>	Italicized words in text represent Delphi identifiers, such as variable or type names. Italics are also used to emphasize certain words, such as new terms.
<i>Keycaps</i>	This typeface indicates a key on your keyboard. For example, "Press <i>Esc</i> to exit a menu."

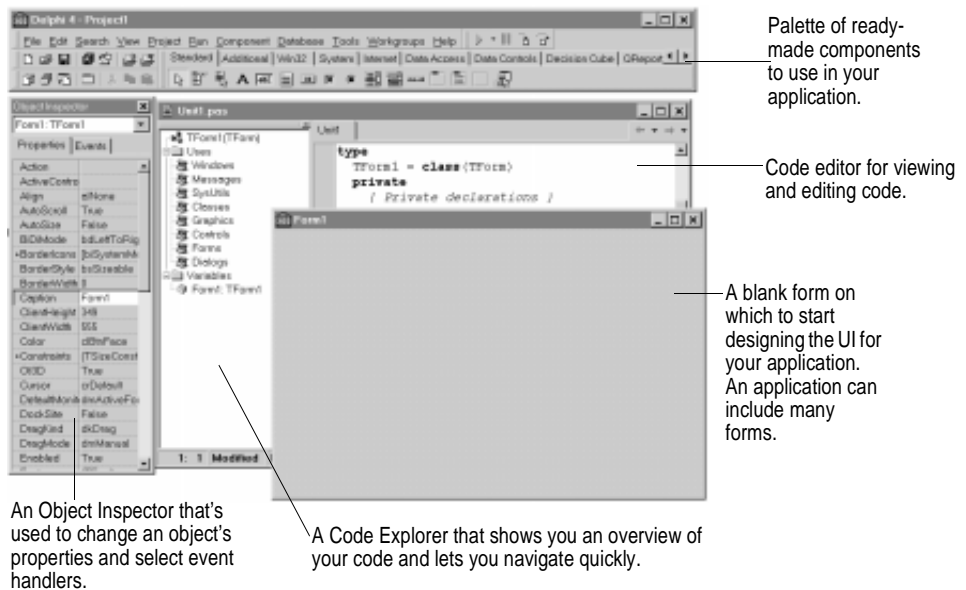
A tour of the environment

Starting Delphi

The best way to get familiar with Delphi is to start it up. You start Delphi the same way you start any Windows-based application. Here are some of the common ways:

- Double-click on the Delphi icon (if you've created a shortcut).
- From the Windows Start menu, choose Programs | Delphi 4.0.

You'll see some of the major tools provided in the Delphi development environment:



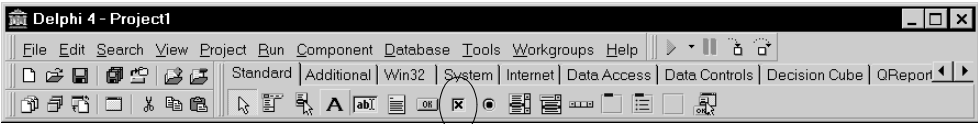
The environment is called the IDE (integrated development environment). Within the IDE, all your programming tools are within easy reach—you can manage

projects, develop applications, write program code, search databases, debug, compile, and browse through Delphi objects without leaving the IDE.

Placing components on a form

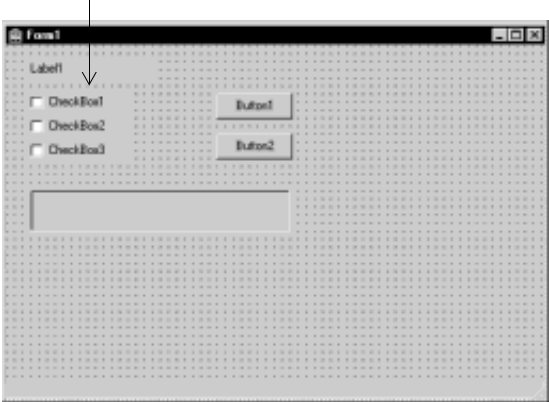
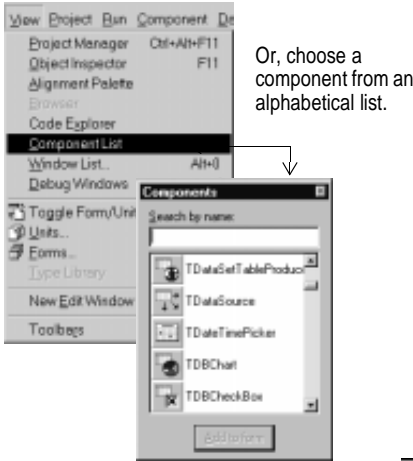
To build a Delphi application interface, you place components on a form, set their properties, and code their event handlers.

Many components are provided on the Component palette, grouped by function.

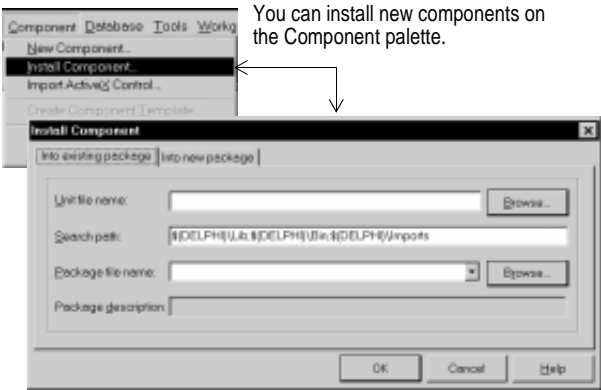
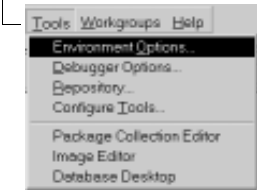


Click a component on the Component palette.

Then click where you want to place it on the form.



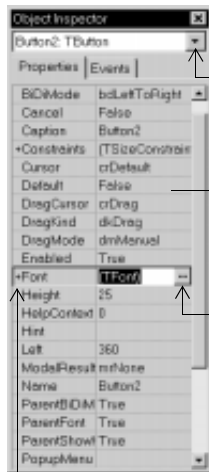
You can also rearrange the palette and add new pages. Choose Environment Options, then the Palette page.



You can install new components on the Component palette.

Changing component appearance and behavior

You can easily customize the way a component appears and behaves in your application by using the Object Inspector. When a component is selected on the form, its properties and events are displayed in the Object Inspector:

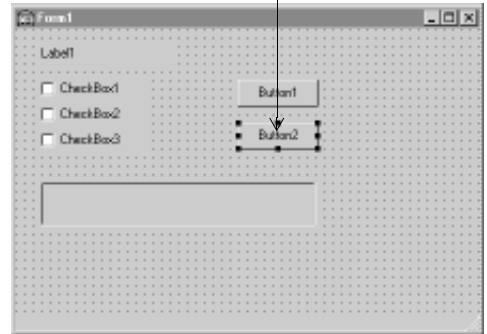


You can select an object on the form by clicking on it...

... or use this drop-down list to select an object. Here, Button2 is selected, and its properties are displayed.

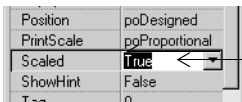
Select a property and change its value in the right column.

Click an ellipsis to open a dialog box where you can change several properties at once.

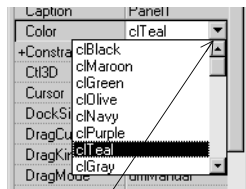


Double-click a plus sign to open a detail list.

Many of the properties have standard values, such as colors, *True* or *False*, or positions. For properties that have binary values like *True* or *False*, you can click on the word to change the value to its opposite. Other properties have associated property editors to set more complicated sets of properties. When you click on a property value, you'll see an ellipsis.

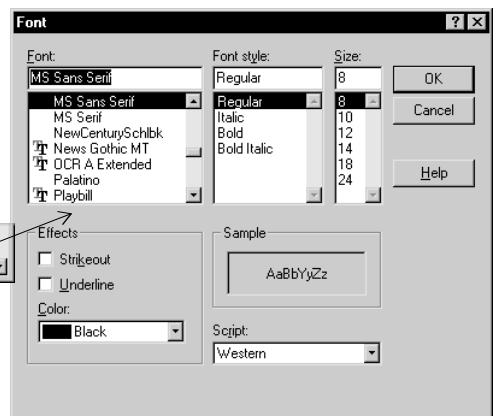
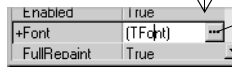


Double-click here to change the value from *True* to *False*.

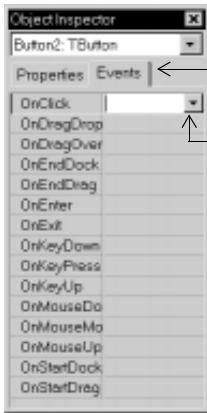


Click on the down arrow to see the default property values. Select the one you want.

Click any ellipsis to display a property editor for that property.



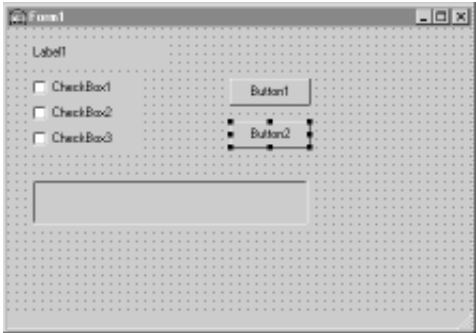
Navigating among events



Click the Events tab in the Object Inspector to navigate among events that each component can handle. Here, Button2 is selected, and its type is displayed: *TButton*.

Select an event from the drop-down list.

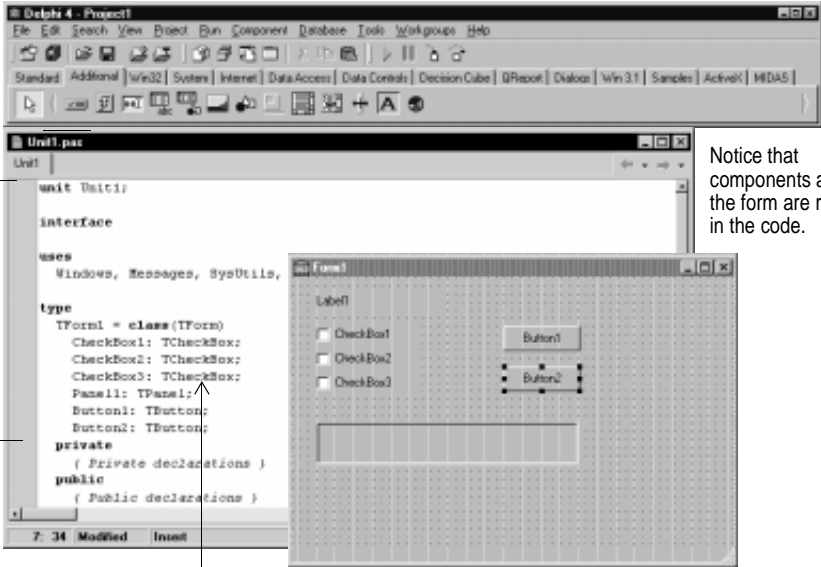
Or, double-click in the value column, and Delphi generates the skeleton code for the event handler, ready for your input.



Editing code

As you visually design the user interface for your application, Delphi generates the underlying Pascal code. When you select and modify the properties of forms and components, the results of those changes are automatically reflected in the project or the form file.

You can also add code to the source files directly using the built-in Code editor.



Delphi-generated code.

Notice that components added to the form are reflected in the code.

Click on any element of the Object Pascal language or VCL, and press *F1* to get help.

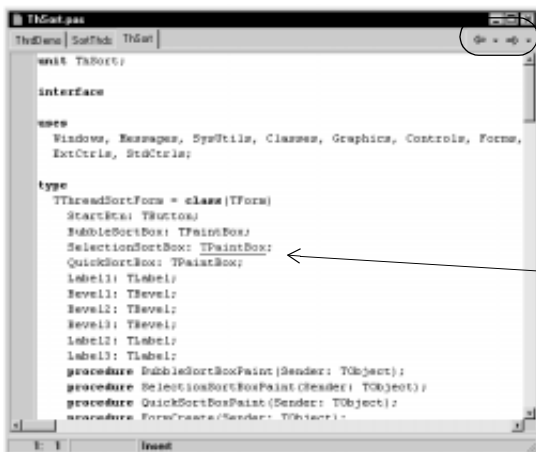
The Delphi Code editor is a full-featured ASCII editor. If using the visual programming environment, a form is automatically displayed as part of a new project. The contents of the form, all its properties, and its components and their properties can be viewed and edited as text in the Code editor by selecting the View as Text option in the form designer's context menu.

The Delphi code generation and property streaming systems are completely open to inspection. The source code for everything that is included in your final EXE—all of the VCL objects, RTL sources, all of the Delphi project files can be viewed and edited in the Code editor.

Browsing with the editor

The Code editor has forward and back buttons like you've seen on a Web browser. You can use them to move to various places that you were working in your code.

By clicking the left arrow, you can move to the last place you were working in your code. You can return to the place you were looking at by clicking on the right arrow.



Use the editor like a Web browser.

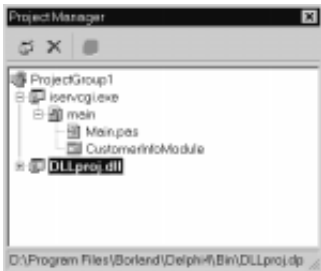
Press **Ctrl** and point to any identifier. The cursor turns into a hand, and the identifier turns blue and is underlined.

Click to display the definition of the identifier.

After navigating, you can move forward and back to various locations where you were in your code.

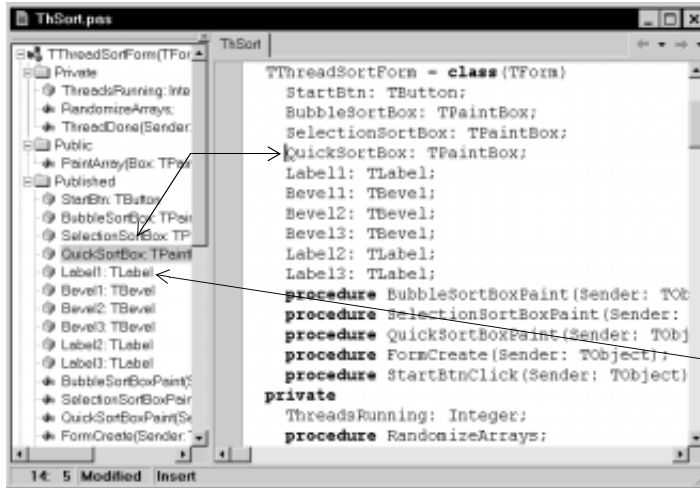
Navigating within your project

Delphi makes it easy to move around within a project. You can display the Project Manager by choosing View | Project Manager. From the Project Manager, you can view any form or unit contained in the project by simply clicking on it. See “Project management tool” on page 2-13 for more about the Project Manager.



Navigating within your code

When the code for a unit is displayed in the Code editor, you can use the Code Explorer to see a structured table of contents for the code. The Code Explorer diagrams all the types, classes, properties, methods, global variables, and global routines defined in your unit. It also shows the other units listed in the uses clause.



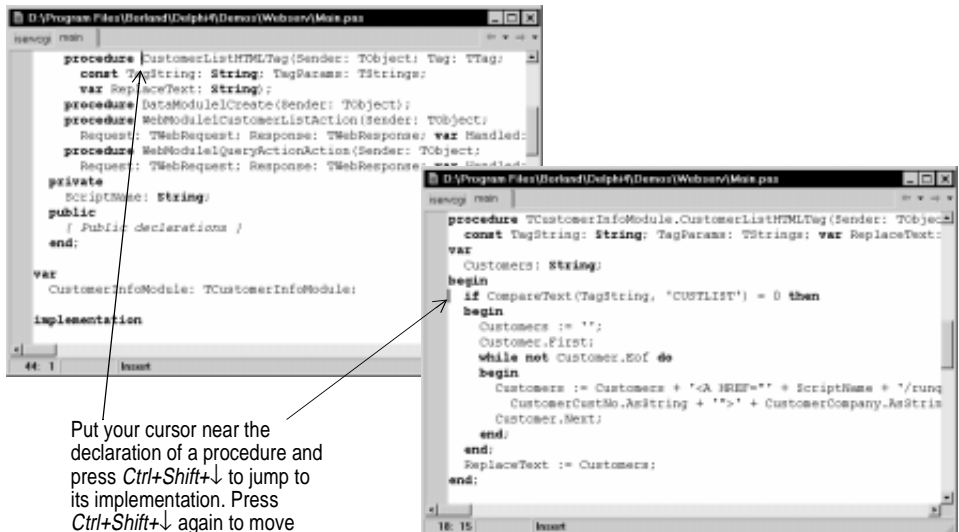
Select an item in the Code Explorer and the cursor moves to that item's implementation in the Code editor.

Move the cursor in the Code editor and the highlight moves to the appropriate item in the Code Explorer.

The Code Explorer shows a bird's eye view of your code.

To search for a class, property, method, variable, or routine, just type its name.

Within the Code editor, you can also move between the declaration of a procedure to its implementation and back again. Type *Ctrl+Shift+↑* (or *Ctrl+Shift+↓*).



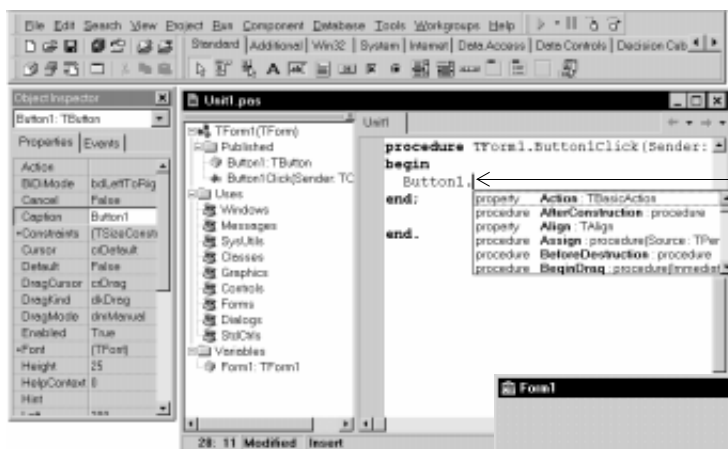
Put your cursor near the declaration of a procedure and press *Ctrl+Shift+↓* to jump to its implementation. Press *Ctrl+Shift+↑* again to move back.

Getting help while coding

While you are writing code, Delphi provides various tools to help. You need to configure these tools using Tools | Environment options.

To get help with writing classes in the **interface** section of a unit, you can use the class completion tool. You just type the class declaration then press **Ctrl+Shift+C**. Delphi automatically adds private read and write specifiers for properties that need them and generates skeleton code in the **implementation** section for all the class's methods. Configure class completion on the Explorer page of the Environment Options.

You can get more detailed help while completing the code. Code Insight displays context sensitive pop-up windows while you're editing in the Code editor.



If using code completion, as soon as you type the dot, Delphi automatically displays a pop-up box that lists the valid properties, methods, and events for the class.

Select an item on the list and press **Enter** to add it to the code.

After dropping a component onto the form, double-click it to display the code that was generated.



Code Insight includes several features that you configure from the Environment Options dialog box. Select Tools | Environment Options and select the Code Insight page. These options are on by default when you install Delphi. Code Insight tools are described in Table 2.1.

Table 2.1 Code Insight tools

Tool	How it works
Code completion	Enter a class name then a period to display a list of properties, methods, and events appropriate to the class; enter an assignment statement and press Ctrl+space to display a valid list of arguments for the variable; type a procedure, function, or method call to bring up a list of arguments.
Code parameters	Type a method call and an open parenthesis to display the syntax for the arguments to the method.
Code templates	Press Ctrl+J to list common programming statements that you can insert into your code. You can set up as many templates as you need.

Table 2.1 Code Insight tools (continued)

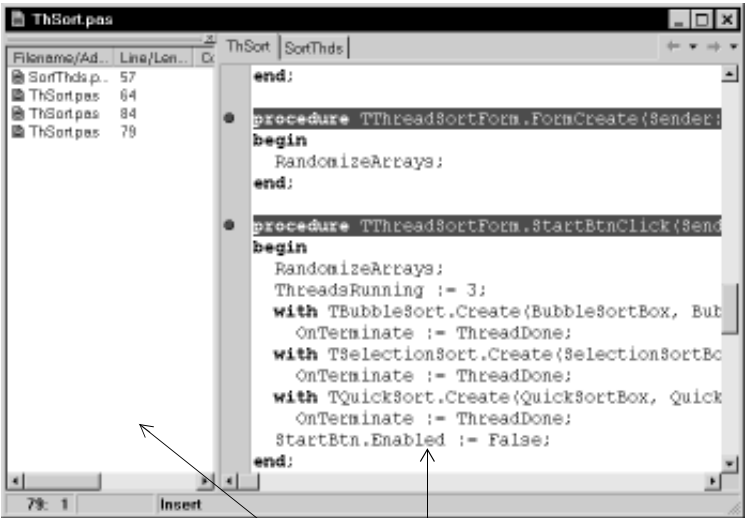
Tool	How it works
Tooltip symbol insight	While editing code, point to any identifier to show the source of its definition.
Tooltip expression evaluation	While paused during debugging, point to a variable to display its current value.

For more information...

Search for “class completion,” “Code Insight,” and “code completion” in the Help index.

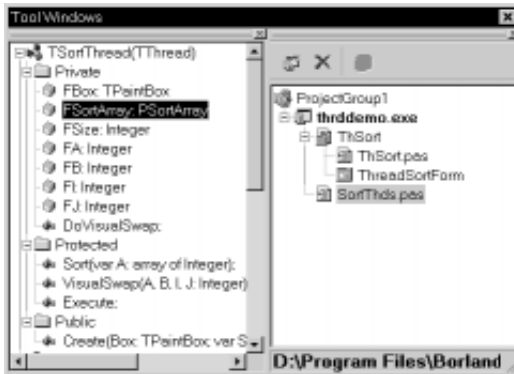
Organizing the environment

Delphi has configurable windows that you can *dock* to other windows in the IDE. Docking lets you make full and efficient use of your screen space as you work on your project. From the View menu, you can bring up any window and then dock it directly onto the Code editor for use while coding and debugging. The docked window attaches to the editor window.



You can dock tool windows onto other windows. Here the Breakpoint list is docked onto the Code editor.

You can also dock two or more windows next to each other to maintain fast one-step access to the tools.



Dock two windows next to each other by dropping a window onto another when a vertical rectangle is displayed.

Here the Code Explorer is docked alongside the Project Manager.

Debugging applications

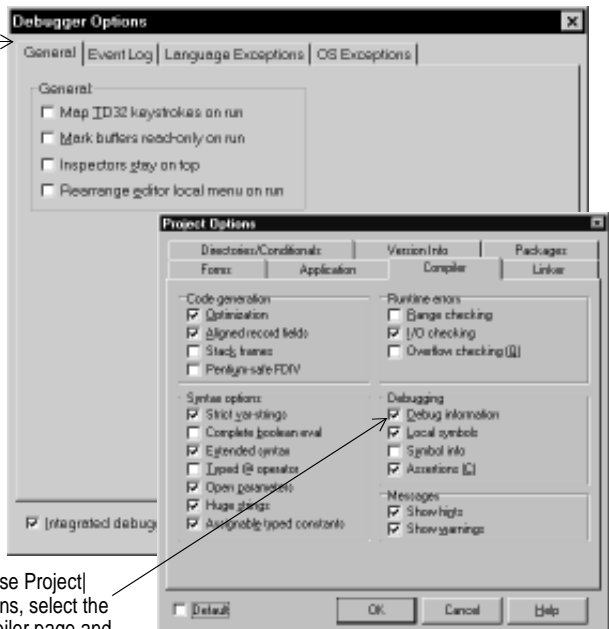
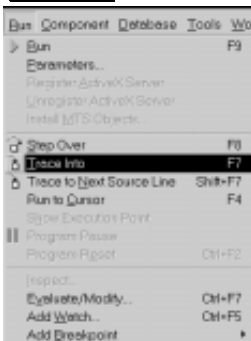
Delphi provides an integrated debugger that helps you locate and fix errors in your applications and dynamic link libraries (DLLs). The integrated debugger lets you control program execution, monitor variable values and items in data structures, and modify data values while debugging.

Choose Tools|Debugger Options and set debugging options on all four pages.

Choose any of the debugging commands from the Run menu and some are also on the toolbar.

Run button

Debugger buttons



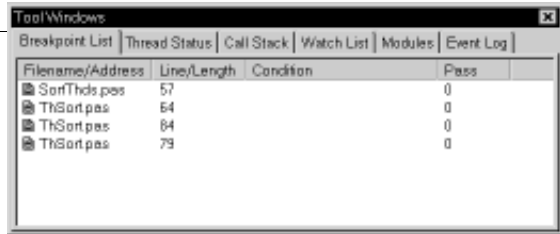
Choose Project|Options, select the Compiler page and check Debug Information

After you compile your program with debug information, you can begin a debugging session by running your program from the IDE. Many debugging views are available to you including a Breakpoint List, Call Stack, Watch List, Local Variables, Thread Status, Modules, CPU view, and an event log. Display them by choosing View | Debug Windows.

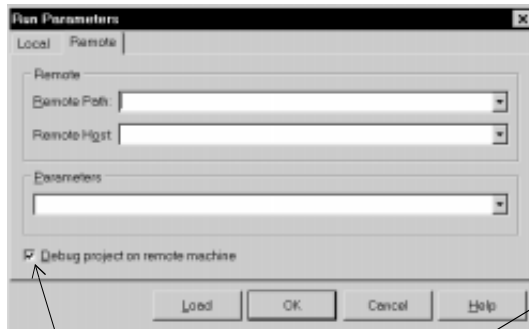
You can view the values of variables, functions on the call stack, and program output to check that the area of code you are examining is performing as designed. You can also step through your code line by line examining the state of the program, viewing the program output, and modifying program data values.

Lots of information is available to you while debugging. Choose View | Debug Windows and select the ones you want.

You can dock many of the debugging views together overlaying each other. Click the tabs to see the different views.



Delphi also supports multiprocess and remote debugging of distributed applications from either the client or server. You turn on remote debugging from the Run | Parameters | Remote page by checking “Debug project on remote machine” and then checking “Include remote debug symbols” on the Project | Options Linker page.



You can use remote debugging to test and debug distributed applications together as if they were one application.

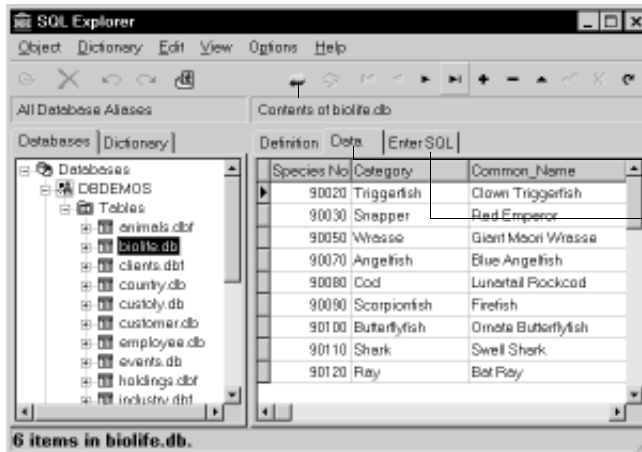
You can set remote run parameters if you include remote debug symbols on the Linker options page.

For more information...

See “Using Delphi” in the Help contents or search for “debugging” in the Help index.

Exploring databases

The SQL Explorer (or Database Explorer in the Standard and Professional editions of Delphi) lets you work directly with a remote database server during application development. For example, you can create, delete, or restructure tables, or import constraints while you are developing a database application.



Choose Database| Explore to display the Explorer. You can see and change the data in a table.

And you can query a database directly.

For more information...

See “Developing Database Applications” in the Help contents or search for “databases” in the Help index.

Storing objects as templates

The Object Repository contains many application objects such as forms, data modules, wizards, and DLLs that simplify development. Choose File | New to display the Object Repository when you are about to begin a new project. Check the repository first to see if there is an object that resembles one you want to create and start from there.

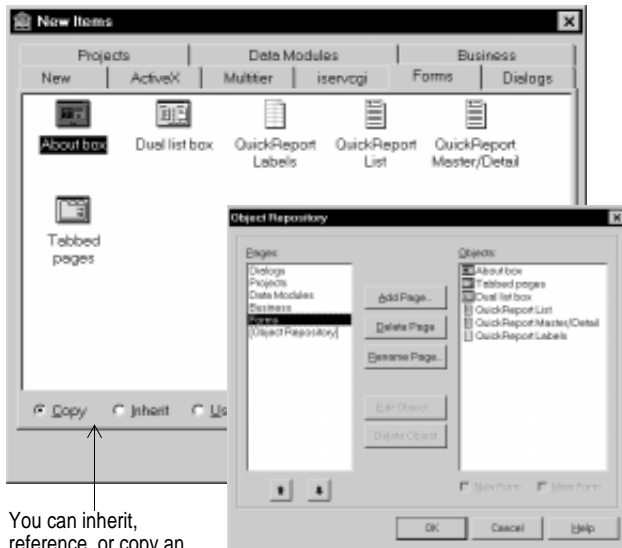
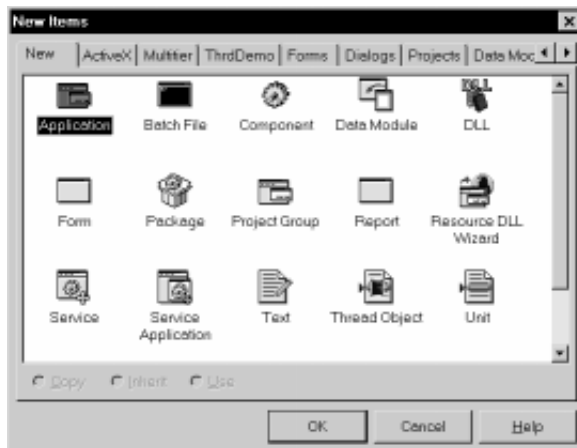
You can easily reuse objects that Delphi provides and objects that you build by accessing them in the Object Repository. Reusing objects lets you build families of applications with common user interfaces and functionality. Building on an existing foundation also reduces development time and improves quality.

The Object Repository also provides a central location for application development tools that all members of a development team can access over a network. Therefore, you can develop forms that you can use in more than one application and save them in the Object Repository. Or, in a shared programming environment, you can develop a standard set of forms for other developers to use within their applications.

You can use the Tools | Repository command to add objects to the Object Repository. Those objects are then available using the File | New command, which opens the New Items dialog box:

Delphi includes a centralized repository for storing and reusing data models, business rules, objects, and forms.

Many of the pages include wizards that simplify development of that type of application or object.



You can inherit, reference, or copy an existing object.

To add your own pages, right-click in the New Items dialog box and choose Properties from the menu. This opens the Object Repository dialog box.

You can also access the many objects and wizards that Delphi provides on the tabbed pages. The number and type of objects available to you is dependent on the version of Delphi that you purchased.

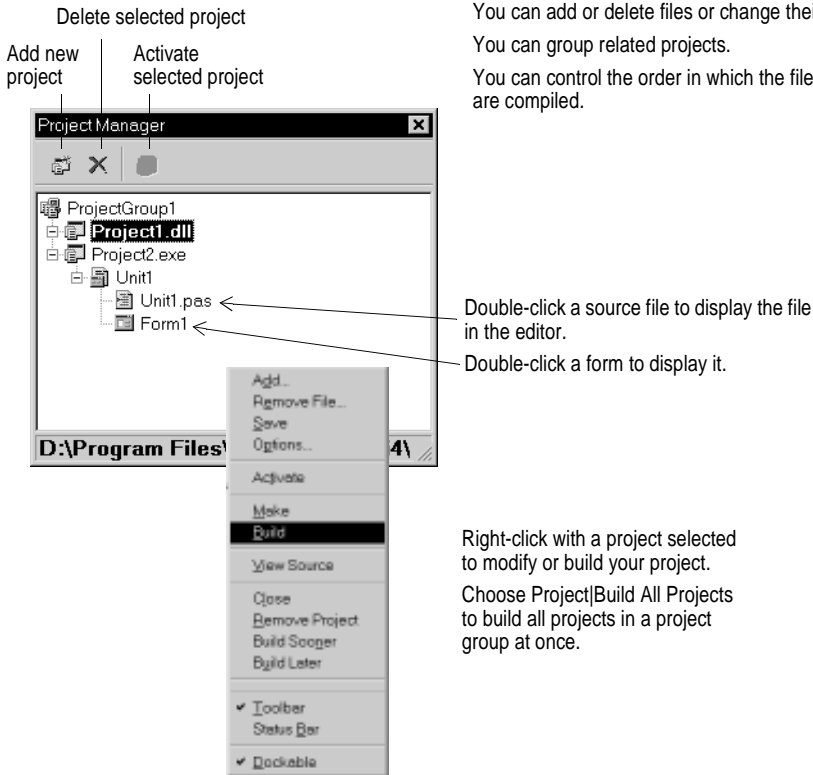
For more information...

See “Using Delphi” in the Help contents or search for “Object Repository” in the Help index. Also choose File | New and browse around in the Object Repository to see the kinds of business wizards, form and dialog templates that you can use as a starting point for your application.

Project management tool

Use the Project Manager to keep track of and access the form and unit files that make up a Delphi application. Choose View | Project Manager to see a list of files in your application and easily navigate among files.

You can use the Project Manager to combine related projects into a single project group. Project groups allow you to organize and work on interdependent projects such as separate tiers in a multi-tiered application or DLLs and executables that work together.



You can add or delete files or change their order.

You can group related projects.

You can control the order in which the files or related projects are compiled.

Right-click with a project selected to modify or build your project.

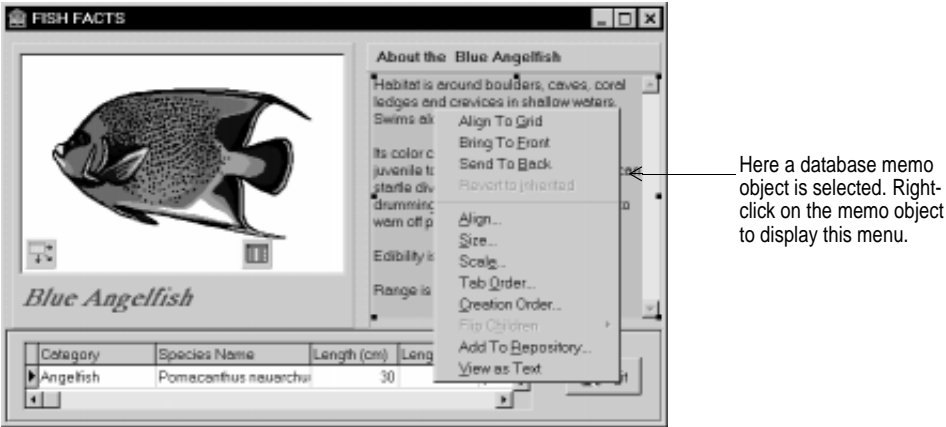
Choose Project|Build All Projects to build all projects in a project group at once.

The Project Manager shows you a high-level view of the projects contained in a project group, and of the form, unit files, resource, object, and library files contained in the project file. You can use the Project Manager to open, add, save, and remove project files. You can also use the Project Manager to access default project settings.

The Project Manager is a useful tool if you share files among different projects because it lets you quickly find each file in the project. It can also be used for other common project management tasks such as package management and resource (.RC) management.

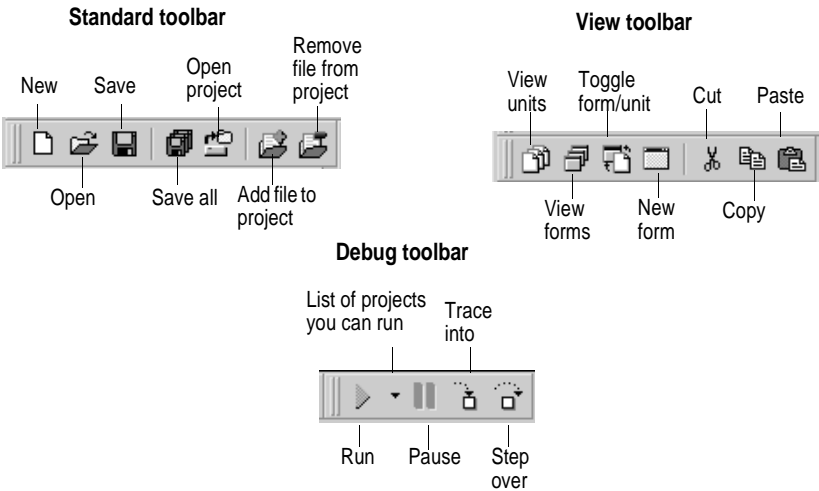
Handy pop-up menus

You can right-click on objects and on most Delphi tools to view a menu of commands appropriate to the context you are working in. These are called *context menus*.



Toolbars

The Delphi toolbars, located in the main window, contain buttons that provide quick access to common operations and commands (Open, Save, Run, and so on).

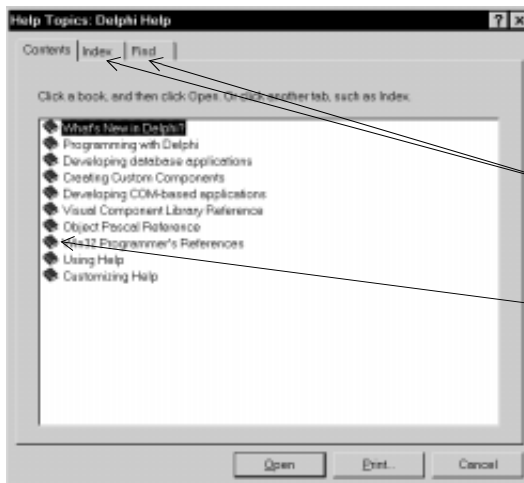


To find out what any button does, point to it for a moment and a hint is displayed.

You can also use the right-click menu to hide the toolbars. To display a toolbar if it's not showing, choose View|Toolbars and check the one you want.

Getting help

Delphi's online Help provides extensive documentation on Delphi.

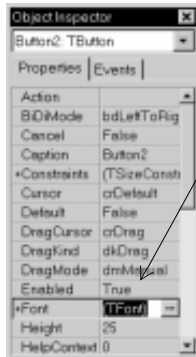


Choose Help|Contents to open the master contents for the Delphi Help system. Browse through topics by category.

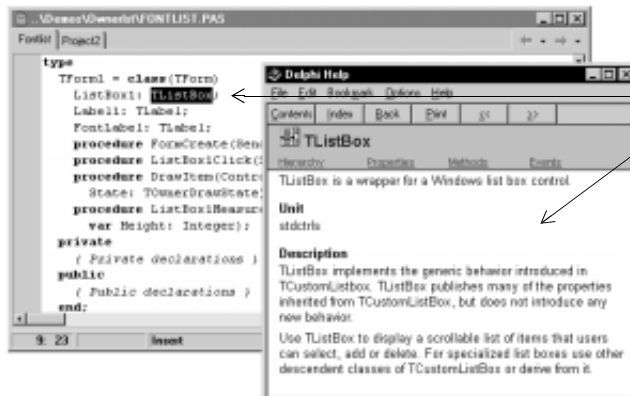
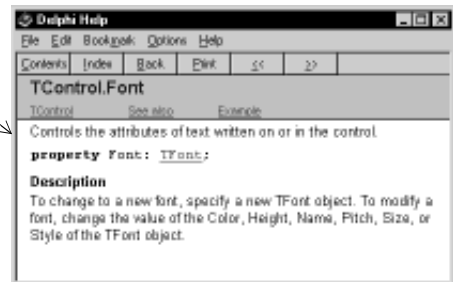
Click Index or Find to search for specific keywords.

Click on a book to see what's in it.

Here are some of the many ways you can display help:



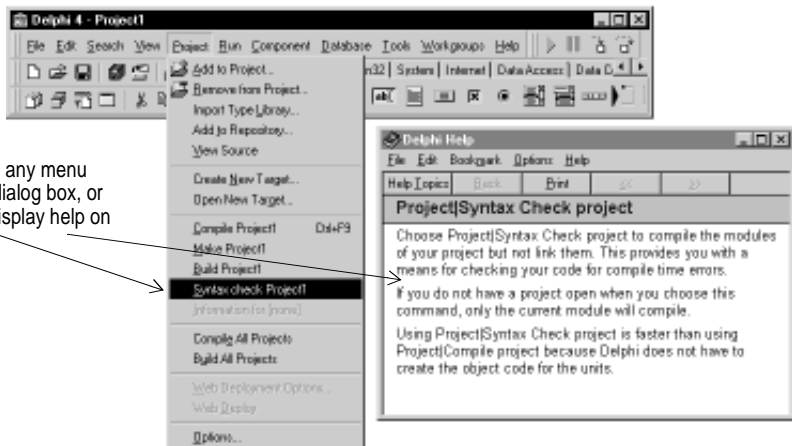
Press **F1** on a property or event name in the Object Inspector to display help.



Press **F1** on a property, event, method, function, procedure, or type in the Code editor to display help.

You can display help on any part of the development environment including menu items, dialog boxes, windows, screens, toolbars, and components.

Press **F1** on any menu command, dialog box, or window to display help on that item.



Your first application—a brief tutorial

The quickest way to introduce yourself to the Delphi environment is to write an application. This tutorial guides you through the creation of a Delphi program that lets you navigate the fields in a marine-life database table that comes with Delphi. After you set up access to the table, you'll write an event handler that opens the standard File Save dialog box. This allows you to write information from the database table to a file.

Starting a new application

Before beginning any new application, create a folder to hold the application source files. This way, you don't mix your application source files with other types of files, and a unique folder lets you easily track and maintain all the files contained in this application.

- 1 Create the folder MySource in the Projects directory off the main Delphi directory to hold the project files you'll create with this simple application.
- 2 Open a new project.

Each application is represented in Delphi by a *project*. When you start Delphi, it opens a blank project by default. If another project is already open, choose File | New Application to create a new project.

Whenever you open a new project, Delphi automatically creates the following files:

- Project1.DPR: a project file.
- Unit1.PAS: a source file associated with the main project form.
- Unit1.DFM: a resource file that stores the properties and objects you place in the current form. (If you create another form, you'll see another .DFM file.)

- 3 Choose File|Save All to save your files to disk. When the Save dialog appears, navigate to your MySource folder, and save the Delphi files using their default file names. (You can also use the Delphi Save As dialog box to create the new MySource directory.)

In addition to the files already mentioned, Delphi creates other files associated with your project, as you can see by looking in your MySource directory.

In Delphi, you design the user interface for your applications using forms. Forms can contain menus and context menus, they can be put together to make application dialog boxes, and they can be parent or child windows. Essentially, forms are the canvases on which you create applications.

When you open a new project, Delphi displays a graphical representation of the project form, named *Form1* by default.



The default new form has Maximize and Minimize buttons, a Close button, and a Control menu.

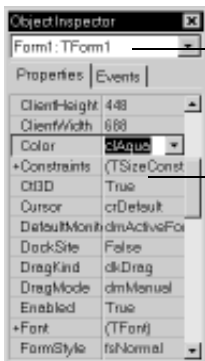
If you run the form now by pressing F9, you'll see that these buttons all work.

To return to design time, click the **X** to close the form.

You place *objects* on forms to create user interfaces. Objects, for example, can be standard interface controls (such as check boxes and drop-down lists), or they can be full-featured components (such as data grids, bar charts, and editors).

In addition to the form, Delphi also displays the Object Inspector. With the Object Inspector, you can set values for the objects you've placed on your forms.

Figure 3.1 Object Inspector



The drop-down list at the top of the Object Inspector shows the currently selected object; in this case, the object is *Form1* and its type is *TForm1*.

When a form is selected, the Object Inspector shows the properties of the form.

Setting property values

To design the application interface, drop objects on a form, then set the properties of the object in the Object Inspector. Setting the properties while creating the application interface is called making *design-time* settings.

- Set the *Color* property of *Form1* to *clAqua*.

To set the *Color* property, find the form's *Color* property in the Object Inspector and click the drop-down list displayed to the right of the property. To change the background color of the form to aqua, choose *clAqua* from the list of predefined colors (see the previous figure).

Note When programming with Delphi, you should set object property values using the Object Inspector; resist the urge to set initial property values in the source code. This way, Delphi sets up and maintains your source code.

Adding objects to the form

The Delphi Component palette (see the following figure) includes many of the components that you can use to create your application. Components are grouped onto different palette pages for easy access. You select different pages of the palette by clicking the corresponding Component palette tabs. Icons on each tab represent the components that you can use in your application.



Component palette tabs

Components

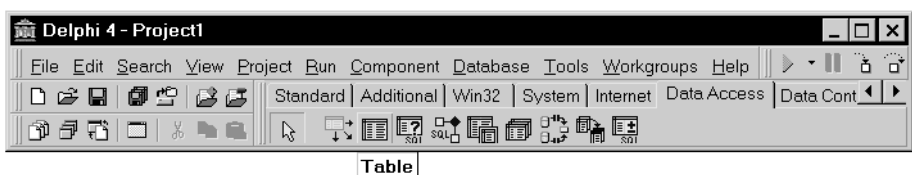
Add components to the interface by selecting the component on the palette then clicking on the form where you want to place it.

You can also double-click a component to put the component in the middle of the form.

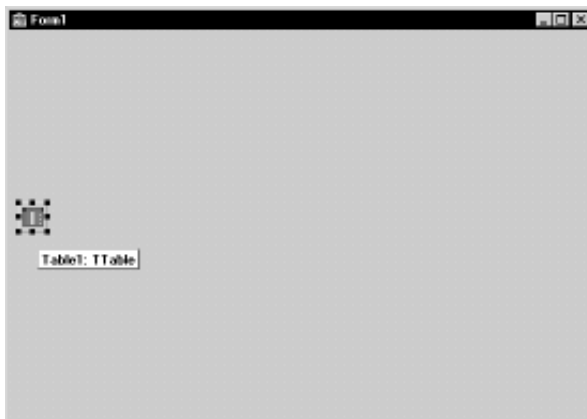
Using the components on the Component palette, Delphi enables you to quickly create an interface for your application.

- 1 Drop a *Table* object onto the form.

Click the Data Access tab on the Component palette. To find the *Table* component, point at a component in the Component palette for a moment; Delphi displays a Help hint showing the name of the component, as shown here.



When you find the *Table* component, click it once to select it then click on the form to drop the component onto the form. Delphi adds that component to the middle of the current form. (The *Table* component is nonvisual so it doesn't matter where you put it.) Delphi names the object *Table1* by default.

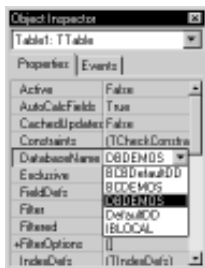


Nonvisual components are shown as icons on the screen.

When you point to an object on the screen, Delphi displays its name – *Table1* – and the type of object it is – *TTable*.

Note Placing a component on a form creates an *object instance* of that component. Once the component is on the form, Delphi generates all the Object Pascal code necessary to create that object in your application. Here is the great advantage of using Delphi: you don't have to worry about the code that creates or maintains the objects you use in your application—Delphi does all that work for you.

- 2 Set the *DatabaseName* property of the table to *DBDEMOS*. (DBDEMOS is the alias to a database that contains the table you're going to use.) You can select DBDEMOS from *DatabaseName* property drop-down list.



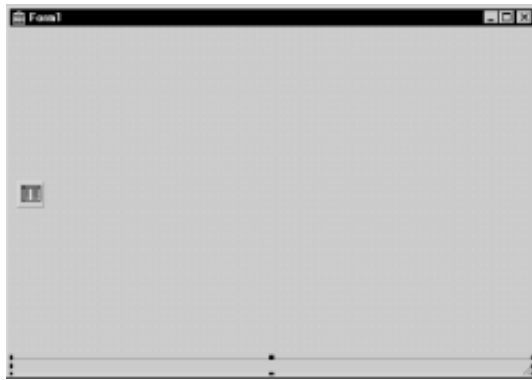
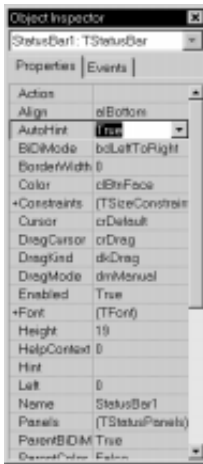
Click the down arrow to display the property drop-down list.
Select *DBDEMOS*.

Setting this property value provides access to an existing database table.

- 3 Double-click the *StatusBar* component on the Win32 page of the Component palette. This adds a status bar to the bottom of the application where you can view help hints for menu items.



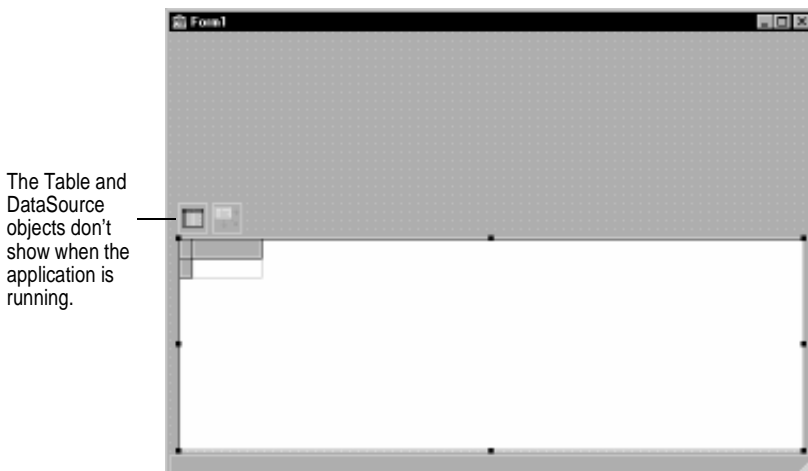
- 4 Set the *AutoHint* property of the status bar to *True*. The easiest way to do this is to point and double-click on *False* next to *AutoHint*. (This will cause help hints to display later when you point to a tool on a toolbar.)



Accessing a database

You're now ready to connect database controls to the database, your data source.

- 1 Any time, you can enlarge the size of you application's window by dragging the lower right corner of the form.
- 2 From the Data Access page of the Component palette, drop a *DataSource* component on the form. The *DataSource* component is nonvisual so it doesn't matter where you put it on the form. Set its *DataSet* property to *Table1*.
- 3 From the Data Controls page, choose the *DBGrid* component and drop it onto your form. Position it in the lower left corner of the form above the status bar and expand it. Your form should now resemble the following figure.



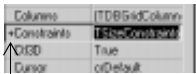
The Table and DataSource objects don't show when the application is running.

The Data Controls page on the Component palette holds the components that let you control how you view database data in your applications.

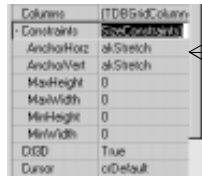
To display all the fields in a database table, use a *DBGrid* component.

4 Set DBGrid properties to anchor the grid to the form:

- Click on the plus next to *Constraints* to expand and show its subproperties.
- Set *AnchorHorz* to *akStretch*.
- Set *AnchorVert* to *akStretch*.



Click on the plus sign to display the *Constraints* subproperties.



Then set these to *akStretch*.

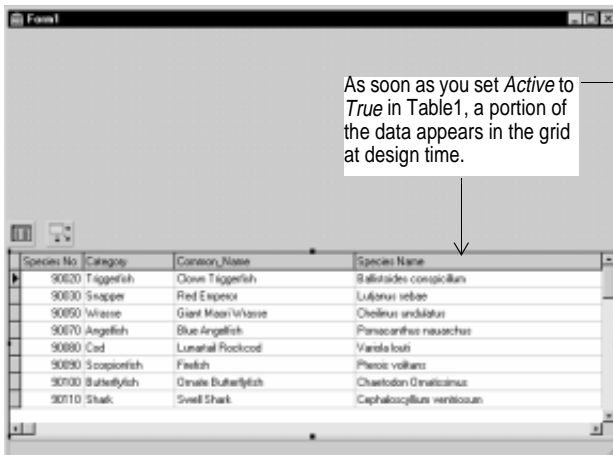
5 Set the *DataSource* property of DBGrid to *DataSource1* to access the database you set up with the *Table* and *DataSource* objects.

Now you can finish setting up the *Table1* object you previously dropped on the form.

6 Give the *Table1* object focus by clicking on it in the form, then set its properties as follows:

- Set *TableName* to *BIOLIFE.DB* (Name is still *Table1*).
- Set *Active* to *True*.

When you set the *Active* property to *True*, the grid fills with the data contained in the BIOLIFE.DB database table. (If the grid doesn't fill as shown in the following figure, make sure you've correctly set the properties for all of the application objects as explained in the previous instructions.)



As soon as you set *Active* to *True* in *Table1*, a portion of the data appears in the grid at design time.



Since the *DBGrid* control is *data aware*, it displays the data in the table while you are designing your application. The data display gives you a visual check,

showing that you've correctly hooked up to the database. However, note that being data aware doesn't mean you can edit the data at design time. To edit the data in the table, you'll have to run the application.

- 7 Press *F9* to compile and run the project.

Adding support for a toolbar and a menu

When you run the application, Delphi opens the program in a window like the one you designed on the form. The grid is not shown on the application interface.

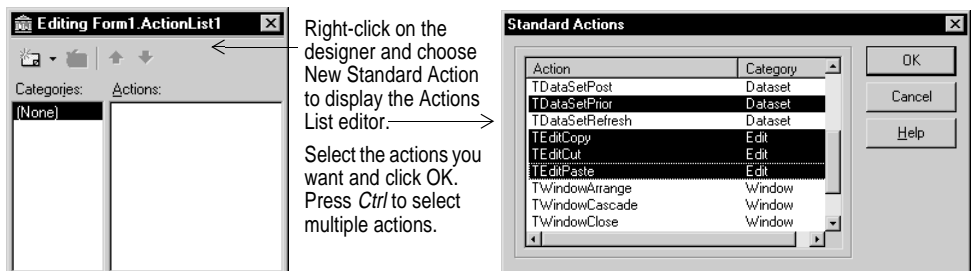
Your program is a full-fledged Windows program, complete with Minimize, Maximize, and Close buttons, and a Control menu that you can also use to close the application. The data grid contains the values from the table, and you can scroll the data grid to see the values in the BIOLIFE.DB table.

Even though your program contains all this, it's still lacking a few details. Most Windows applications implement toolbars and menus to make them easy to use.

- 1 Click the **X** in the upper right corner to close the application and return to the design-time view of the form.
- 2 From the Win32 page of the Component palette, drop an *ImageList* onto the form. This is another nonvisual control so it doesn't matter where you place it. It will include any images used by your application.
- 3 From the Standard page of the Component palette, drop an *ActionList* onto the form.

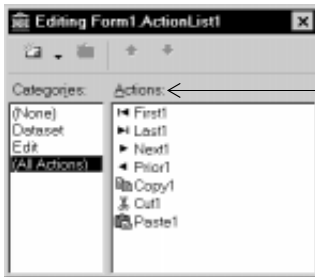
Tip An *action list* lets you centralize the response to user commands (actions), providing control over menus and toolbars.

- 4 Set the action list's *Images* property to *ImageList1* (so you'll automatically get icons for standard actions on the toolbar).
- 5 Double-click the action list to display the Action List editor:



- 6 Right-click on the Action List designer and choose New Standard Action. The Standard Actions list box is displayed.

- 7 Select the following actions: *TDataSetFirst*, *TDataSetLast*, *TDataSetNext*, *TDataSetPrior*, *TEditCopy*, *TEditCut*, and *TEditPaste*. (**Tip:** Use the *Ctrl* key to multi-select.) Then click OK.



You've added standard actions that Delphi provides along with standard images.

You'll use these on a toolbar and menu.

- 8 Click on the **X** to close the Action List designer.

You've added standard actions. Now you're ready to add the menu and toolbar.

Adding a menu

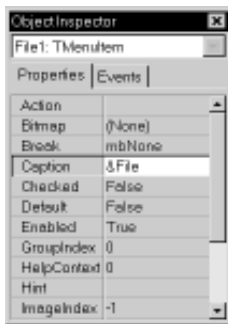
In this section, you're going to add a menu with three top-level items (File, Edit, and Record). You'll add menu items to each using some of the standard actions you added to the action list.

- 1 From the Standard page of the Component palette, drop a *MainMenu* component onto the form. It's nonvisual at this point so anywhere is OK.
- 2 Set its *Images* property to *ImageList1*.
- 3 Double-click the menu component to display the Menu Designer.



You can set up your menu using the Menu Designer.

- 4 Type **&File** to set the *Caption* property of the first top-level menu item and press *Enter*.

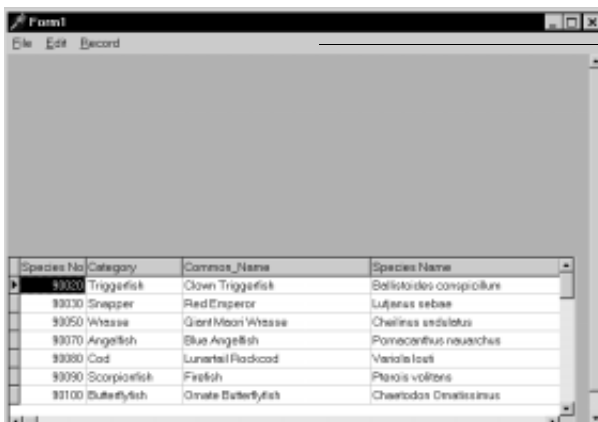


When you type **&File** and press **Enter**, the top-level File command appears ready for you to add the first menu item.



- 5 Type **&Save** and press **Enter** to create a Save menu item under File.
- 6 Type a hyphen in the next item under the File menu and press **Enter** to create a separator bar on the menu.
- 7 Type **E&xit** and press **Enter** to create an Exit menu item under File.
- 8 Click on the second top-level menu item (to the right of File) and type **&Edit** and press **Enter**. The menu item under Edit is selected.
 - In the Object Inspector, set its *Action* to *Cut1* and press **Enter**.
 - Select the next menu item under *Cut*, set its *Action* to *Copy1* and press **Enter**.
 - Select the next menu item under *Copy*, set its *Action* to *Paste1* and press **Enter**.
- 9 Click on the third top-level menu item (to the right of Edit) and type **&Record** and press **Enter**. The menu item under Record is selected.
 - In the Object Inspector, set its *Action* to *First1* and press **Enter**.
 - Select the next menu item under *First*, set its *Action* to *Prior1*.
 - Select the next menu item under *Prior*, set its *Action* to *Next1*.
 - Select the next menu item under *Next*, set its *Action* to *Last1*.
- 10 Click on the **X** to close the Menu Designer.

You can then press **F9** to compile and run the program to see how it looks:



The menu bar is shown on top and you can't see the non-visual controls when the program is running.

Close the application when you're ready to continue.

Adding a toolbar

- 1 On the Win32 page of the Component palette, double-click the toolbar to add it to the form.
 - Set the *Indent* property to 4.
 - Set the *Images* property to *ImageList1*.
 - Set *ShowHint* to *True*.
- 2 Add buttons to the toolbar:
 - With the toolbar selected, right-click and choose New Button three times.
 - Right-click and choose New Separator.
 - Right-click and choose New Button four more times.
- 3 Assign actions to the first set of buttons:
 - Select the first button and set its *Action* to *Cut1*.
 - Select the second button and set its *Action* to *Copy1*.
 - Select the third button and set its *Action* to *Paste1*.
- 4 Assign actions to the second set of buttons:
 - Select the first button and set its *Action* to *First1*.
 - Select the second button and set its *Action* to *Prior1*.
 - Select the third button and set its *Action* to *Next1*.
 - Select the last button and set its *Action* to *Last1*.

Here's how it looks:



The toolbar is set up and it uses standard actions provided with Delphi.

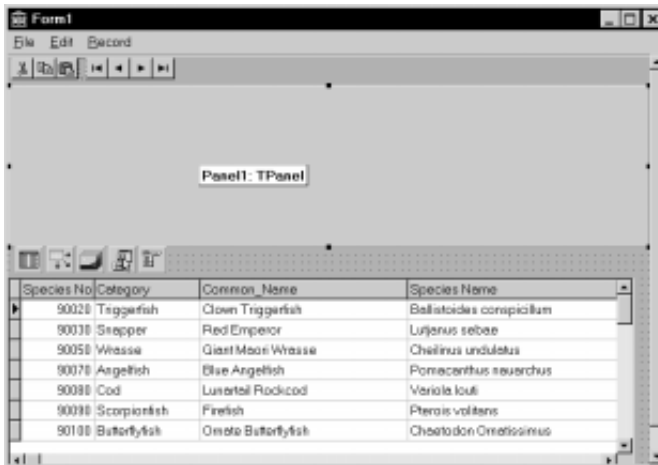
- 5 Press *F9* to compile and run the project.

Check out the toolbar. The first, next, previous, and last buttons work. To make the cut, copy, and paste buttons work, you need to select text within a table cell. Close the application when you're ready.

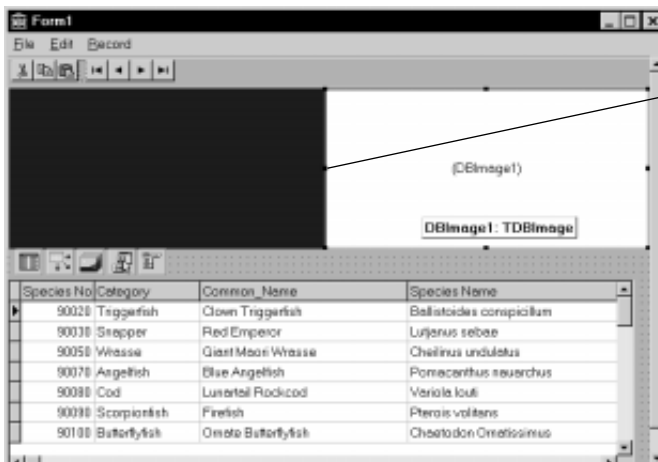
Displaying an image

Now you can associate a picture with each record in the table.

- 1 From the Standard page of the Component palette, drop a *Panel* component onto the top of the form below the toolbar. Delphi names this *Panel1* by default. To remove this caption from your running application, clear the *Panel1* string from the panel's *Caption* property.
- 2 Align *Panel1* to the top of the form by setting its *Align* property to *alTop*. Next drag the bottom of the panel down so it fills the top portion of the form.



- 3 Set the panel's color to *clBlue*.
- 4 From the Data Controls palette page, drop a *DBImage* component on top of *Panel1* and size it so your form resembles the one shown in the following figure by setting its *Align* property to *alRight* and dragging out the left side of the image.



5 Set the following *DBImage* properties:

- Set *DataSource* to *DataSource1*.
- Set the *DataField* property to *Graphic* (the drop-down list shows the fields in the Biolife table; *Graphic* is one of the field names).

Again, because the *DBImage* component is data aware, the component displays the image of the fish in the first record of the table. This shows that you are indeed correctly hooked up to the database.



As soon as you set *DataField* to *Graphic*, you see the fish from the database.

6 Click the Run button (the arrow) on Delphi's toolbar to compile and run your application.

Final touches

Now when you run the application, you can easily move through your database table using the buttons on the toolbar. You can add a couple of other touches to complete the application. Close the running application to return to the design mode of Delphi.

- 1 Select *Panel1*.
- 2 From the Data Controls page of the Component palette, drop a *DBMemo* component onto *Panel1* and position it so it occupies the upper left corner of the panel (below the menus and toolbar). Next, set the following property values:
 - Set *DataSource* to *DataSource1*.
 - Set *DataField* to *Notes* (use the drop-down list of fields).
 - Set *ScrollBars* to *ssVertical*.
- 3 Drop a *DBText* object on *Panel1* under the *DBMemo* object. Enlarge the *DBText* object so it fills the area under the *DBMemo*, then set its properties as follows:
 - Set *DataSource* to *DataSource1*.
 - Set *DataField* to *Common_Name*.
 - Set *Alignment* to *taCenter*.

4 Customize the *Font* property of the *DBText* object using the Font editor.

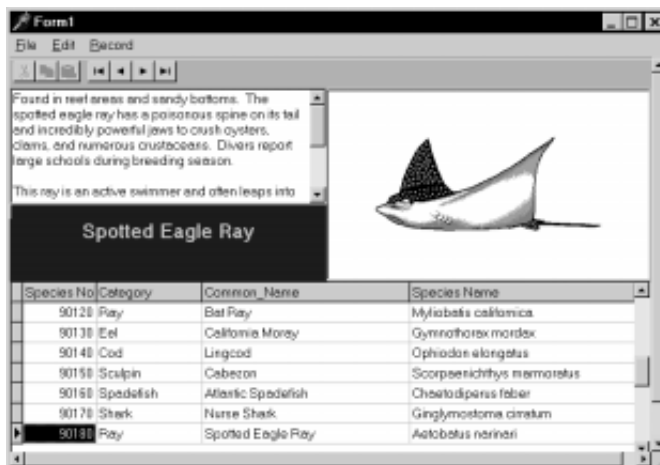
You can access several different types of property editors through the Object Inspector. For example, you can use the Menu editor, the Font editor, and the Picture editor to edit form menus, label fonts, and bitmap pictures and glyphs, respectively.

When you click the *Font* property of the *DBText* object, Delphi displays an ellipsis button on the right side of the property setting, indicating that you can use a property editor to set this property. Clicking anywhere on the property value displays the Font editor, a dialog box that lets you edit several characteristics of fonts.

Modify the following *DBText* settings using the Font editor, then click OK when you're done:

- Set the *Font Style* to *Bold*.
- Set the *Color* to *Silver*.
- Set the *Size* to 12.

5 You can adjust the form so it looks the way you want it. Then, compile and run your application by pressing *F9*.



Your application is shaping up. But you still need to hook up actions to the commands.

You now have a fully functioning Windows application that accesses a database table and displays images, text, and individual data fields from the database. There's even a toolbar and a menu with active commands on them.

Hooking up an event handler

Up until now, however, you have not typed a single line of program code. And this is how it should be if you want to take complete advantage of the Delphi environment. By using the Object Inspector to set the design-time values of your object properties, you let Delphi maintain the code that it generates. In other words, let Delphi do the

initialization work and save your coding efforts for the event handlers, the code that makes your program perform the tasks in your application.

Your next programming feat will be to hook up an event handler to a menu item, a task you'll encounter often when designing user interfaces with Delphi. You'll program the command so that when clicked, an event handler will call the standard Windows File | Save dialog box, which lets you save information from the database into a file.

- 1 From the Dialogs page of the Component palette, drop a *SaveDialog* object onto the form.

Now comes the most important part of your Delphi programming: creating and writing an event handler. The easy part is creating the event handler; technical programming comes into play as you write the Pascal code for the event.

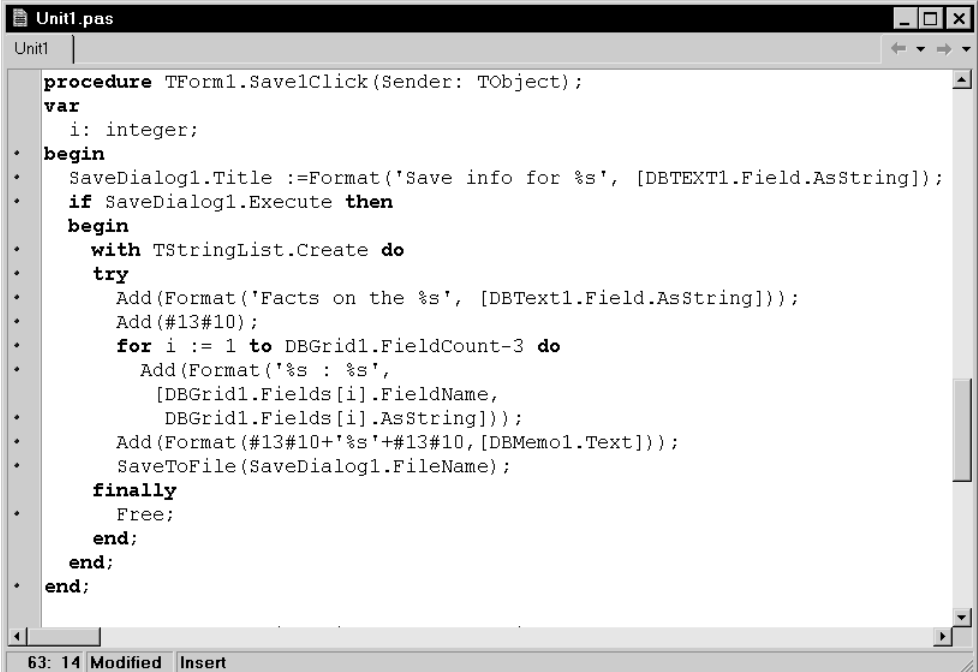
- 2 Choose File | Save from your form to create the default event handler for the button.
- 3 Whenever you select a menu item or double-click an object on a Delphi form, Delphi generates the code for that object's *default event handler*. The Code editor opens with the cursor placed within the skeleton code for the event handler:



Up until now, you've used the Object Inspector to set object properties. You can also use the Object Inspector to access and create object event handlers. By clicking the Events tab on the Object Inspector, you can see the available events for the object currently in focus on the Delphi form.

Most components on the Component palette have a default event. When you double-click an object on a form, Delphi creates the handler for the default event. For most components, the *OnClick* event is default; this is the event that gets called whenever you click an object in a running application.

- 4 Complete the event handler by adding the code that is shown in the **var** section and between the **begin** and **end** of the event handler:



```

Unit1.pas
Unit1

procedure TForm1.Save1Click(Sender: TObject);
var
  i: integer;
begin
  SaveDialog1.Title := Format('Save info for %s', [DBTEXT1.Field.AsString]);
  if SaveDialog1.Execute then
  begin
    with TStringList.Create do
    try
      Add(Format('Facts on the %s', [DBText1.Field.AsString]));
      Add(#13#10);
      for i := 1 to DBGrid1.FieldCount-3 do
        Add(Format('%s : %s',
          [DBGrid1.Fields[i].FieldName,
            DBGrid1.Fields[i].AsString]));
      Add(Format(#13#10+'%s'+#13#10, [DBMemo1.Text]));
      SaveToFile(SaveDialog1.FileName);
    finally
      Free;
    end;
  end;
end;

```

63: 14 Modified Insert

Detailing the workings of this code is beyond the scope of this tutorial. What the code does is call the File Save dialog box (the *SaveDialog1* object) when you choose the File | Save. When you specify a file name, your application writes data on the currently selected fish to a text file.

- 5 To add code for the Exit command, choose File | Exit. Delphi generates and displays the skeleton event handler in the editor:

```

procedure TForm1.Exit1Click(Sender: TObject);
begin

end;

```

Right where the cursor is positioned (between **begin** and **end**), type

```
close;
```

- 6 To run the application, press *F9*.

Congratulations! You have now completed a working Delphi application.

To save the new program to disk, choose File | Save Project As. You can exit the program using the newly activated File | Exit command.

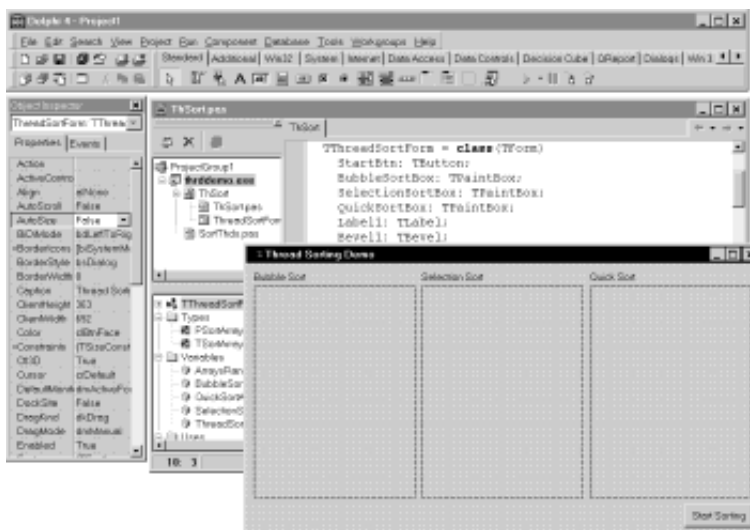
Customizing the environment

This chapter introduces some of the useful ways you can customize many aspects of the Delphi environment so that it supports the way you like to work.

Organizing your work area

By now, you should have a good idea of the many tools and windows that support your development work in Delphi. You've got debugging tools, a Code Explorer to view the structure of your code, a Project Manager to view your project files and group related projects, the Object Inspector to set object properties and events, forms where you design the user interface, a Code editor, and many more.

So many tools are available, you will want to organize your work area so that you can work most efficiently.



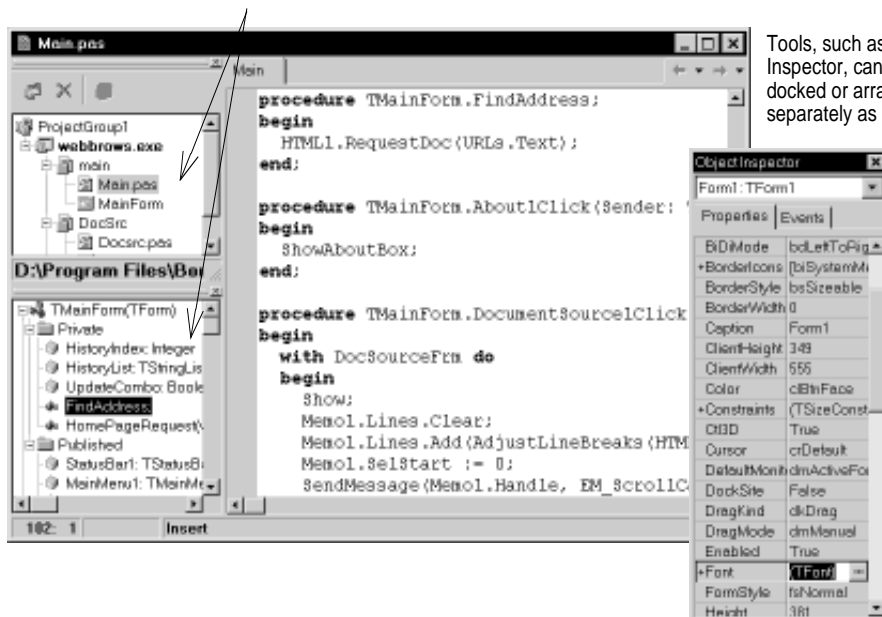
Here a sample thread sorting demo is shown with several tools displayed and organized for easy access.

Organizing tools

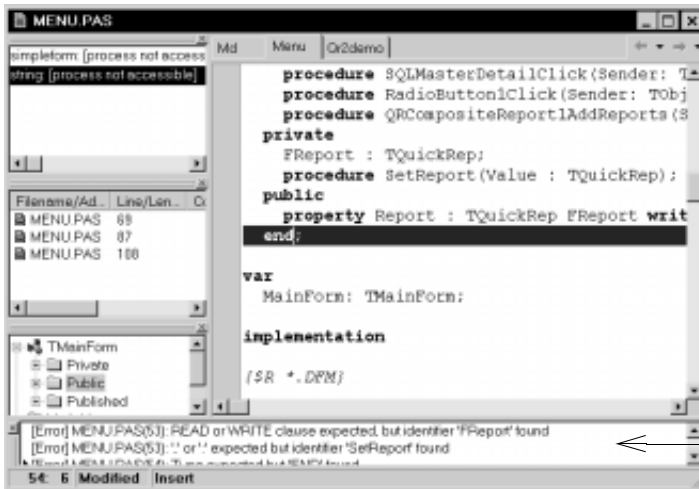
Delphi lets you organize all of the tools and tool windows as you want to on the desktop. The windows and tools are all separate and can be displayed or closed, however you like to work. Many of the windows can be docked (attached) to other windows and some of the windows can be docked on top of other windows forming tabbed tool windows.

For example, while you're designing the application interface, you might want to dock the Project Manager and the Code Explorer onto the Code editor.

You can dock windows onto others. Here the Project Manager and the Code Explorer are docked onto the Code editor.



While debugging code, you can dock watch and breakpoint views onto the editor. From these views, you can click on a breakpoint or watch and go right to that location in the code.



Just point to the window and drag it until it's where you want it.

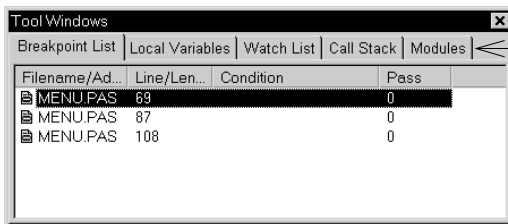
To dock onto another window, drag until you see a light box showing where it will go. If the box is darker, the window will stay separate.

Click on any error messages to jump to the problem in the code.

With a compiler error selected, press F1 to display information about the error.

You can also dock multiple tools together.

To dock tools together, drag a window over another until a box appears inside the window and let go.



Use the tabs to display the different tools.

Here various debugging views are docked together.

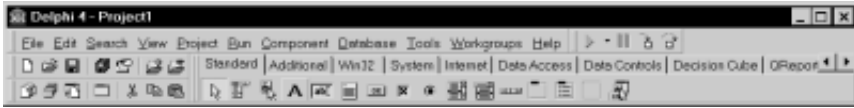
There are infinite ways to organize Delphi tools. As you're working, you'll have to experiment to discover the most convenient way for you to work.

For more information...

Search for "docking" in the Help index.

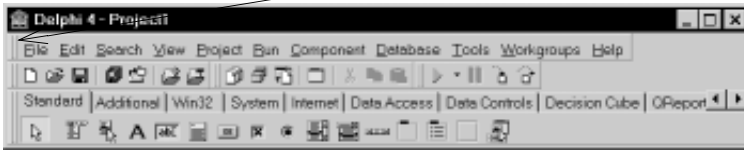
Organizing menus and toolbars

The main window, which occupies the top of the screen, contains the menu, toolbars, and the Component palette. You can reorganize its contents.



Main window showing the default organization.

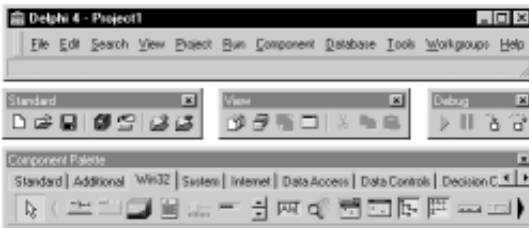
You can move the parts around within the main window. Click the grabber (the double bar on the left of the menu or toolbars) and drag it to where you want it.



Main window organized differently.

You can pull these parts off the main window organizing them any way you want or removing them from the desktop altogether.

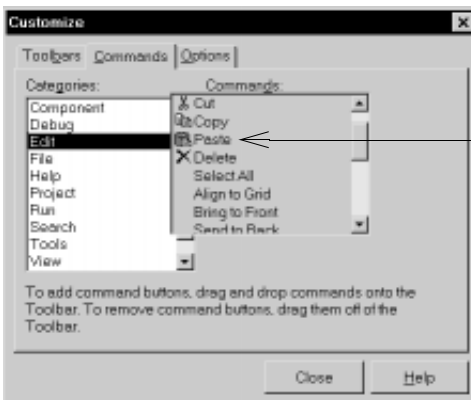
You can break up the parts of the main window. Click the grabber (the double bar on the left of the menu or toolbars) and drag it to where you want it.



You can even take the menu off.

You can position the parts or close them. Redisplay them using View|Toolbars.

Further, you can customize all the toolbars adding or deleting available tools.

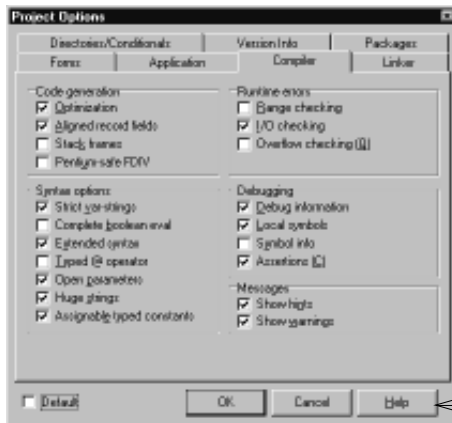


Right-click on any toolbar and choose Customize.

From the Commands page, you can select any command listed and drag it onto the toolbar.

Setting project options

You can change project settings that affect compiling, linking, where files are stored, and other general application details. You can make these changes from the pages of the Project Options dialog box displayed by choosing Project | Options.



← The pages in the Project Options dialog box let you customize compiler options for your project.

Click Help or press F1 while pointing to any page to get details on the options.

Following are the kinds of options you can set.

Table 4.1 Project options

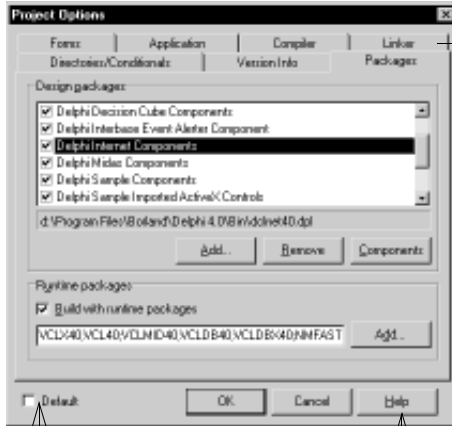
Project options page	Types of options
Directories/ Conditionals	Control the location of the project's compiled output, specify location of application resources, enter conditional compiler directives.
Version Info	Specify version information for the project and whether to automatically increment the build number.
Packages	List the design-time packages installed in the IDE and the runtime packages required by your project.
Forms	List available forms; select the main form for applications; choose which of the available forms are automatically created and in which order.
Application	Specify a title, a Help file, an icon, and an extension for your application.
Compiler	Set options for how you want your program to compile. These options correspond to switch directives that you can also set directly in your program code. Includes options for optimization, debugging information, runtime errors, and many other compiler options.
Linker	Set options for how you want your program to link including options for a map file, linker output files, and EXE and DLL generation.

For more information...

For details about the options on any page of the Project Options dialog box, click the Help button on that page or search for "Project Options dialog box" in the Help index.

Setting options for all new projects

The Project Options dialog box contains a check box labeled Default. Checking Default writes the current settings from the Compiler, Linker, Directories/Conditionals, Packages, and VersionInfo pages of the Project Options dialog box to the options file DEFPROJ.DOF. Delphi then uses the project options settings stored in this file as the default for any new projects you create.



The pages in the Project Options dialog box let you customize compile and link options for your project.

Put a check here to use the runtime packages specified here by default for all new projects you create.

Click Help or press F1 while pointing to any dialog box to get details on the options.

For more information...

See “projects, options” in online Help.

Restoring Delphi’s original default settings

To restore Delphi’s original default settings, delete or rename the DEFPROJ.DOF file which is located in the project directory.

Creating project defaults

You can use default projects and forms to standardize a set of related projects.

Specifying a default project

A default new project opens whenever you choose File | New Application. If you haven’t specified a default project, Delphi creates a blank project with an empty form. You can specify a project you’re using as a template to be the default new project.

You can also designate a project wizard to run by default when you start a new project. A *project wizard* is a program that enables you to build a project based on your responses to a series of dialog boxes.

You always have the option to override the defaults by choosing File | New and selecting any item from any of the tabbed pages available through the New Items dialog box.

For more information...

Search for “projects, specifying default” in the Help index.

Displaying a default form

A default new form opens whenever you choose File | New Form or use the Project Manager to add a new form to an open project. If you haven’t specified a default form, Delphi uses a blank form. You can specify any form as the default new form. Or you can designate a form wizard to run by default when a new form is added to a project.

You can also specify a form template or expert to use as the default main form whenever you begin a new project.

For more information...

Search for “forms, specifying default” in the Help index.

Setting tool preferences

You can customize tools such as the editor, designer, debugger, and compiler that you use in the Delphi environment. The options that you set affect all Delphi projects.

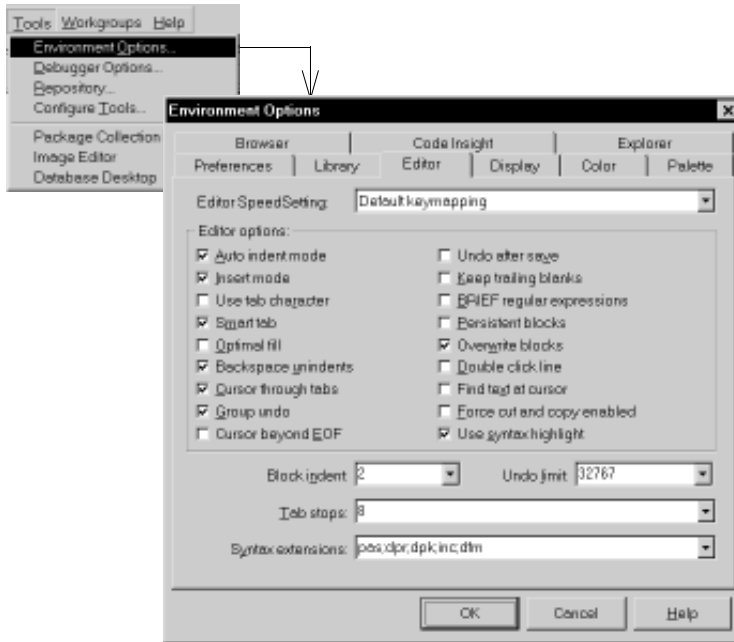
To specify environmental preferences, choose Tools | Environment Options. This displays the Environment Options dialog box. There are literally hundreds of environment options that let you control tool usage for all projects on the 10 tabbed pages. The best way to learn about the options is to click on the tabs and look at the different options.

For more information...

Click the Help button on any page of the Environment Options dialog box for help with that page, or search for “Environment Options dialog box” in the Help index.

Customizing the Code editor

One of the tools you may want to customize right away is the editor.



Several pages in the Environment Options dialog box have Delphi Code editor options you can set.

You have a choice of four keystroke mappings that set editor options. Or you can choose your own custom set of options.

Set your own colors for the editor including syntax highlighting on the Colors page. Set font and other properties on the Display page.

For more information...

Click the Help button on the Editor page of the Environment Options dialog box or search for “Editor” in the Help index.

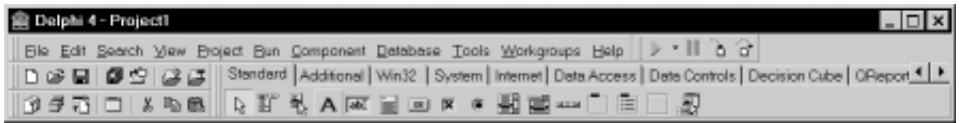
Customizing the Component palette

Delphi includes many components that are part of a class hierarchy called the Visual Component Library (VCL). Components are the elements you use to build your Delphi applications. They include all the visible parts of an application interface, such as edit controls and buttons as well as those that aren’t visible, such as datasets or timers.

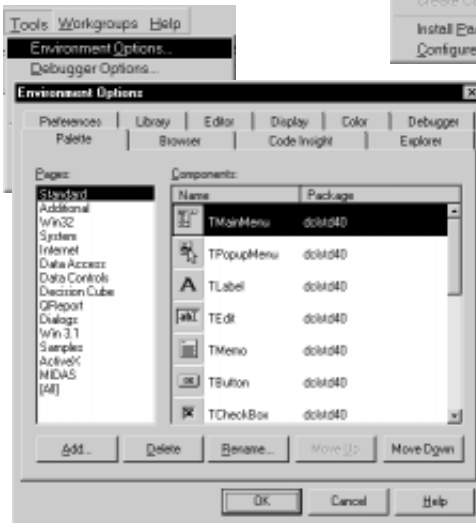
Some useful components are placed on the Component palette and they are organized logically onto tabs. However, the components that you want easy access to may not be included in the default palette organization. You have control over which components appear on the Component palette and in what order. You can

- Hide, rearrange, and rename components
- Install additional components

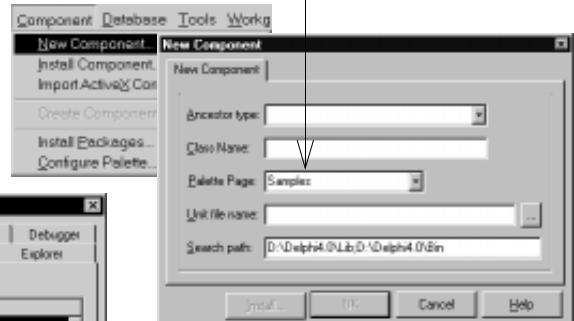
- Create component templates (standard or custom) and add them to the palette



You can also rearrange the palette and add new pages. Choose Tools|Environment Options, then the Palette page.



You can create new components and add them to the Component palette.



Rearranging the Component palette

To add, delete, or rearrange Component palette pages or to rearrange components on the pages, choose Component|Configure Palette. The Palette Properties dialog box lists each page on the palette and the components that appear on that page.

You can do the following:

- Move, delete, or rename a page
- Add a page
- Rearrange components on a page
- Move components from one page to another,
- Display a previously hidden component

Adding components to Delphi

You can use design-time packages to add components to the IDE. The DCLUSR30 design-time package is provided as a default container for new components.

You can install your own component packages or component packages from third-party developers. Then, in the Project | Options dialog box you specify all the packages that you want your project to use.

Note Writing components generally requires a more in-depth knowledge of Object Pascal and object-oriented programming than using the components provided in Delphi's VCL.

For more information...

Search for “components” in online Help for instructions on how to add components to Delphi. See “Creating Custom Components” in the Help contents for information on writing components.

Installing component packages

Components you're installing to the Delphi IDE must be contained in a *package*. The unit file that contains your component source code must follow the model for component source files. To install packages, choose Component | Install Package.

The components in the package are installed on the Component palette pages specified in a call to the *RegisterComponents* procedure.

For more information...

Refer to “Programming with Delphi” in the Help contents. You can search for “packages” in the Help index.

Adding ActiveX controls

You can add ActiveX controls to the Delphi component library as long as they conform to Microsoft specifications. To do so, specify the ActiveX library file that contains the control you want to add. Delphi then automatically creates a “wrapper” unit (a .PAS file) for the control to make it into a recognizable object type, and adds it to the list of installed units. You can specify a name for the control and which palette page you'd like it to appear on.

To add an ActiveX control to the component library, choose Component | Import ActiveX Control. The Import ActiveX Control dialog displays the ActiveX controls that are registered on your system so you can add them to your Delphi projects.

For more information...

From the Import ActiveX Control dialog box, click Help for instructions on adding ActiveX controls. Search for “ActiveX controls” in the Help index.

Creating component templates

You can create component templates that are made up of a number of components. After arranging components on a form, setting their properties, and writing code for them, you can save them as a component template. Later, by selecting the template from the Component palette, you can place the preconfigured components on a form in a single step; all associated properties and event-handling code are added to your project at the same time.

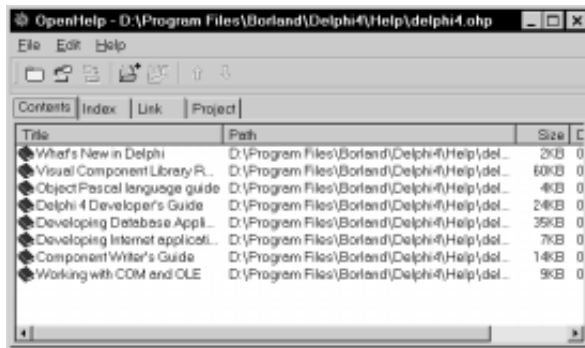
Once you place the template on a form, you can reposition the components independently, reset their properties, and create or modify event handlers for them just as if you had placed each component in a separate operation.

For more information...

Search for “component template” in the Help index.

Customizing Delphi Help

Delphi comes with numerous Help files and a tool called OpenHelp that helps you to configure Windows Help (.HLP) files. You start OpenHelp by choosing Help | Customize Help (or clicking on OH.EXE in the Bin directory).



Delphi's default Help system is set up in the Delphi4.ohp project file in the Help directory. You can customize this project file by adding or deleting Help files.

This list controls which Help files appear in the Help Contents in Delphi.

Delphi provides a default Help system that includes all the Help files in the master table of contents. Choosing Help | Contents in Delphi displays this table of contents. With OpenHelp, you can customize the Help system so it displays the information you want it to. You can add or remove Help files. You can even add Help for additional tools you work with in the Delphi development environment. OpenHelp also allows you to remove references to obsolete Help files from the system registry.

OpenHelp stores information about your Help system in a project. The project defines a master table of contents, master index, and a context-sensitive Help search range for a set of Windows Help files.

For more information...

Choose Help | Contents while running OpenHelp for details about using OpenHelp to customize your Help system.

Programming with Delphi

Delphi provides a visual programming environment for developing Windows 95, Windows 98, and Windows NT applications. Delphi includes a comprehensive class library called the VCL and a suite of RAD design tools, including application and form templates, and programming wizards. Delphi supports truly object-oriented programming: the class library includes objects that encapsulate the Windows API as well as other useful programming techniques.

This chapter describes the features of the Delphi development environment and touches on many of the tools that are available to you.

Delphi development environment

When you start Delphi, you are immediately placed within the visual development environment, called the IDE. This environment provides all the tools you need to design, develop, test, debug, and deploy applications.

Delphi's development environment includes a visual form designer, Object Inspector, Component palette, Project manager, Code Explorer, source code editor, debugger, and installation tool. You can move freely from the visual representation of an object (in the form designer), to the Object Inspector to edit the initial runtime state of the object, to the source code editor to edit the execution logic of the object. Changing code-related properties in the Object Inspector, such as the name of an event handler, automatically changes the corresponding source code. Likewise, changes to the source code, such as renaming an event handler method in a form class declaration, are immediately reflected in the Object Inspector.

For more information...

See "Using Delphi" in the Help contents and search for "IDE," "Object Inspector," "Code Explorer," "Component palette," or "debugging" in the Help index.

Another way to find out about the IDE is to select any window, tool, or dialog box and press *F1*. You can also use the online tutorials provided with Delphi.

Designing applications

You can use Delphi to design any kind of 32-bit application—from general-purpose utilities to sophisticated data access programs. Delphi’s database tools and data-aware components let you quickly develop powerful desktop database and client/server applications. Using Delphi’s data-aware controls, you can view live data while you design your application and immediately see the results of database queries and changes to the application interface.

The type of application you can design is somewhat determined by the specific Delphi product version you purchased. Different product versions support different types of program development. Table 5.1 describes the main features of each version.

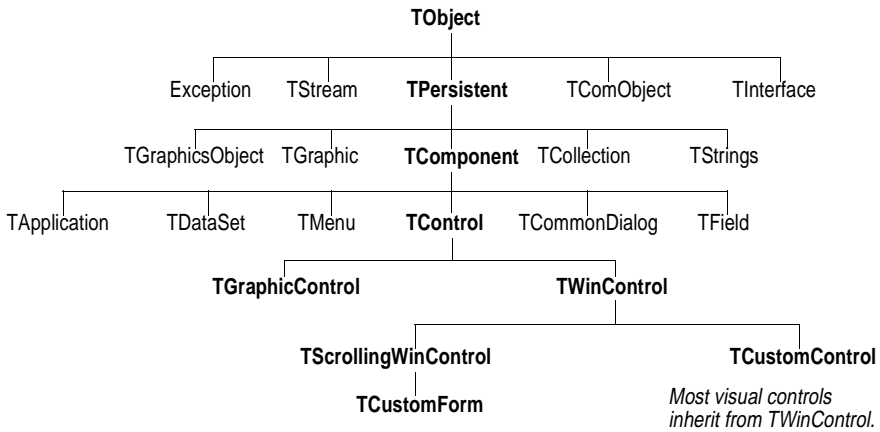
Table 5.1 Delphi product versions

Product	Development tools
Standard	General-purpose 32-bit Windows programming; OLE automation; COM and interface support; multi-threading; data-aware components; and report generation.
Professional	Same as Standard plus enhanced support for database development using Access, Paradox, or dBASE files; ODBC; ActiveX support; local Interbase for SQL development; VCL source code for object development; charting components; Internet components.
Client/Server	Same as Professional plus support for two-tier and multi-tier database development; CORBA support with VisiBroker; MTS support; Oracle8 object relational database connectivity; integrated suite of components for building HTML server applications; and decision support. Includes SQL drivers, SQL Monitor, SQL Explorer, Interbase (4-user license), InstallShield, PVCS Version Manager.

Refer to Inprise Online (www.inprise.com) for up-to-date product information.

Using the VCL

The Visual Component Library (VCL) includes many classes that you can use in your applications. The classes are organized into the VCL object hierarchy. Figure 5.1 shows the relationship of some of the classes that make up the VCL.

Figure 5.1 Visual Component Library object hierarchy

You can learn about the available objects in the hierarchy by browsing through the *VCL Reference* online, or by glancing at the VCL poster included with the product.

The VCL is intimately tied to the Delphi IDE, and is what gives you the ability to quickly develop applications. *Components* are the elements you use to build your Delphi applications. They include all the visible parts of an application interface, such as dialog boxes and buttons, as well as those that aren't visible while the application is running, such as system timers or COM servers.

All VCL objects, and in fact all objects in Object Pascal, are derived from *TObject*. *TObject* is unique in that it is an abstract object that has no properties or events, only methods that allow you to derive objects from this base class. Use *TObject* as the immediate base class when writing simple objects that are not components.

Components are objects that you can manipulate at design time. All components in the VCL are derived from the abstract component type *TComponent*. *TComponent* provides the minimal properties and events necessary for a component to work in Delphi (for example, so the component can be installed on the Component palette and added to forms visually by dropping it). The various branches of the library provide other, more specialized capabilities.

The VCL components you will likely use the most are the VCL's controls, such as *TForm* or *TSpeedButton*. Controls are visual components derived from the abstract component type *TControl*.

Of the over 600 objects in the VCL, most are not visual. The Delphi IDE allows you to visually add some nonvisual components to your programs. For example, to write a database application that connects to a table, you could drop a *TDataSource* component on your form. *TDataSource* is a nonvisual component, but is represented on the form by an icon (which doesn't show up at runtime), and you can manipulate the properties and events of *TDataSource* in the Object Inspector just as you would a visual control.

For more information...

To learn about specific VCL components, see the “VCL Reference” in the Help contents. Point to any VCL component, property, event, method, function, procedure, or type in the Code editor and press *F1*.

For detailed information on using specific components to accomplish programming tasks, see “Programming with Delphi” in the Help contents. For general information about the VCL, search for VCL in the Help index.

Creating the application user interface

All visual design work in Delphi takes place on *forms*. When you open Delphi or create a new project, a blank form is displayed on the screen. You can use it to start building your application interface including windows, menus, and common dialogs.

You design the look and feel of the graphical user interface for an application by placing and arranging visual components such as buttons and list boxes on the form. Delphi takes care of the underlying programming details. You can also place nonvisible components on forms to handle a variety of different tasks such as activating specific code at scheduled intervals, capturing information from databases, performing calculations, and managing other interactions. You can manipulate all components—visual or nonvisual—by changing their properties.

As you are designing the interface, you can use *action lists* to standardize responses to user actions. Drop an action list (from the Standard page of the Component palette) onto a form and choose from a list of commonly used actions or create your own. For example, Cut, Copy, and Paste actions are some of the actions you can use with existing controls such as *TEdit*. By standardizing, you create an application that is easier to maintain and contains reusable bits of code.

For more information...

See “Using Delphi” and “Programming with Delphi” in the Help contents or search for “forms” and “action lists” in the Help index.

Using components

Many of the VCL components are provided in the development environment itself on the Component palette. You select components from the Component palette and drop them onto the form to design the application user interface. Once a visual component is on the form, you can adjust its position, size, and other design-time properties using the Object Inspector without having to write a single line of code.

Delphi components are grouped functionally on the different pages of the Component palette. For example, commonly used components such as those to create menus, edit boxes, or buttons are located on the Standard page of the Component palette. Handy controls such as a timer, paintbox, and media player are on the System page.

Components achieve a high degree of encapsulation. For example, consider the use of a dialog containing a push button. In Delphi, the push button component is pre-programmed to respond to a mouse click using its built-in *OnClick* event handler. Your program does not have to determine that a button has been clicked. You only provide the routine that is called when the button is clicked, and through the Object Inspector, assign that routine to the *OnClick* event of the button.

Similarly, most Windows messages are handled by Delphi components. When you want to respond to a Windows message, you need only provide an event handler.

For more information...

See “Using Delphi” in the Help contents or search for “components” or “forms” in the Help index.

Changing component behavior

You can easily customize the way a component appears and behaves in your application by using the Object Inspector. When a component is selected on the form, its properties and events are displayed in the Object Inspector.

You use the Properties page of the Object Inspector to set the initial program startup values for the components you’ve placed on the form. You use the Events page of the Object Inspector to quickly navigate among events that each component can implement. By clicking on a particular event, Delphi generates the event handler code for that specific component event. In Delphi, you will spend most of your programming time writing the event handlers for the objects that you place on your application forms, rather than managing the details of Windows programming.

For more information...

See “Using Delphi” in the Help contents or search for “component,” “Object Inspector,” and “event handler” in the Help index.

Designing menus

After you add a menu component to a form, you can use the Menu Designer to create and edit menu bars and pop-up menus. You need to add a menu component to your form for every menu you want to include in your application. Delphi provides predesigned menu templates that you can use to design menus, or you can build the menu structure of your program from scratch.

The menus you design are immediately visible in the form without having to run the application to see the results. You can also change menus at runtime to provide additional options for the application user.

For more information...

See “Programming with Delphi” in the Help contents or search for “menus” and “Menu Designer” in the Help index.

Developing applications

One of Delphi's strengths is that it simplifies Windows application development. Delphi includes several wizards and other specialized tools that speed up the development process. Delphi is an ideal tool for developing all types of business applications, games, custom controls, utilities, Web-enabled applications, and complex database applications.

Following are the three basic types of Windows applications you can use Delphi to develop:

- Windows GUI applications
- Console applications
- Service applications

Windows GUI applications are the most common type of software available for the Windows platform. The Graphical User Interface (GUI) consists of windows and dialog boxes which work together to perform a group of functions. Word processors, spreadsheets, and Delphi are examples of GUI applications. Delphi is also well-suited for developing console applications (such as Grep) and service applications (such as NT servers).

You can develop dynamic link libraries (DLLs) and packages, a special type of DLLs, to add features to the design environment or to the application itself. See "Creating packages and DLLs" on page 5-7.

You can also use Delphi to develop other specialized types of applications. Refer to "Writing database applications" on page 5-8 for information about some of the tools available to help with database application development. See "Developing distributed applications" on page 5-11 for information on client/server applications.

For more information...

See "Using Delphi" in the Help contents or search for "applications," "DLLs," and "packages" in the Help index.

Creating Windows GUI applications

Lots of developers use Delphi to develop Windows GUI applications because Delphi makes it so easy to design an integrated user interface. You can use the Form Designer to visually create the user interface using the many components provided in the VCL. You can control the design-time and runtime behaviors of your applications by setting project options in the Delphi IDE.

You can design both single document interface (SDI) or multiple document interface (MDI) applications using Delphi. In an MDI application, more than one document or child window can be opened within a single parent window. This is common in applications such as spreadsheets or word processors. An SDI application, by contrast, normally contains a single document view.

For more information...

See “Using Delphi” in the Help contents or search for “applications” and “user interface” in the Help index.

Creating packages and DLLs

Dynamic link libraries (DLLs) are modules of compiled code that work with an application to provide distinct features. They typically contain code that can be used for more than one application.

A *package* is a special dynamic link library used by Delphi applications, the IDE, or both. Runtime packages provide functionality when a user runs an application. Design-time packages are used to install components onto the Component palette in the IDE and create special property editors for custom components. A single package can work at both design time and runtime. To distinguish them from other DLLs, package libraries are stored in files that end with the .DPL (Delphi package library) extension.

Like other runtime libraries, packages contain code that can be shared among applications. For example, most of Delphi’s commonly used components reside in a package called VCL30. Each time you create an application, you typically specify that it uses VCL30. When you compile an application created this way, the application’s executable image contains only the code and data unique to it; the common code is in VCL30.DPL. A computer with several package-enabled applications installed on it needs only one copy of VCL30.DPL, which is shared by all the applications and the Delphi IDE itself.

For more information...

See “Developing Delphi Applications” in the Help contents. Search for “packages” in the Help index.

Handling exceptions

Error conditions in Delphi are indicated by exceptions. An *exception* is an object that contains information about what error occurred and where it happened. Exceptions are built into many classes and they are raised automatically when something unexpected occurs. You need to use exception handling to recognize, locate, and deal with programming errors.

Delphi includes many exception classes for automatically handling errors such as divide-by-zero and file input/output errors. All of the exception classes descend from one root object called *Exception*. *Exception* encapsulates the fundamental properties and methods for all exceptions and provides a consistent interface for applications to handle exceptions.

For more information...

See “Developing Delphi Applications” in the Help contents or search for “exception handling” and “Exception” in the Help index. You can also type “Exception” in the Code editor and press F1 to view specific reference information.

Writing database applications

Database applications allow users to interact with information that is stored in databases. Databases provide structure for the information, and allow it to be shared among different applications.

The Borland Database Engine (BDE) supports scaling from desktop to client/server applications. In fact, one of Delphi’s strengths is its support for creating advanced database applications. Delphi includes built-in tools that allow you to connect natively to Oracle (including Oracle8 with its object-relational extensions), Sybase, Informix, DB2, dBASE, Paradox, FoxPro, Access and Access97, and other servers. While connected, Delphi enables transparent data sharing between applications.

For more information...

See “Developing Database Applications” in the Help contents.

Connecting to databases

The BDE includes drivers to connect to different databases. The Standard version of Delphi includes drivers for local databases: Paradox, dBase, FoxPro, and Access.

With the Professional version, you also get access to Open DataBase Connectivity (ODBC). The goal of ODBC is to make it possible to access any data from any application, regardless of which database management system is handling the data. An ODBC adapter allows the BDE to use vendor-supplied ODBC drivers. By using the ODBC driver, your application can access any ODBC-compliant database.

The Client/Server and Enterprise versions include drivers for remote database servers. You can use the drivers installed with SQL Links to communicate with remote database servers such as Interbase, Oracle, Sybase, Informix, Microsoft SQL server, and DB2.

Using database tools

Tools, such as the SQL Explorer (or Database Explorer), Data Dictionary, Database Desktop, and the BDE Administrator simplify the database application development process. These database tools make it easy for the client to connect to databases, browse existing schema, create new schema, and so on.

Browsing databases

SQL Explorer in Delphi Client/Server and Enterprise (or Database Explorer in the other editions) is a hierarchical browser for inspecting and modifying database server-specific schema objects including tables, fields, stored procedure definitions, triggers, references, and index descriptions.

Through a persistent connection to a database, the explorer lets you

- Create and maintain database aliases
- View schema data in a database, such as tables, stored procedures, and triggers
- View table objects, such as fields and indexes
- Create, view, and modify data in tables
- Enter SQL statements to directly query any database
- Create and maintain data dictionaries to store attribute sets

For more information...

See “Developing Database Applications” in the Help contents or search for “Database Explorer” or “SQL Explorer” in the Help index. For detailed information, see the Database Explorer Help file (Dbexplr4.hlp).

Storing data information

The Data Dictionary provides a customizable storage area, independent of your applications, where you can create extended field attribute sets that describe the content and appearance of data.

For example, if you frequently develop financial applications, you may create a number of specialized field attribute sets describing different display formats for currency. When you create datasets for your application at design time, rather than using the Object Inspector to set the currency fields in each dataset by hand, you can associate those fields with the extended fields attribute set in the Data Dictionary. Using the Data Dictionary also ensures a consistent data appearance within and across the applications you create.

In a client/server environment, the Data Dictionary can reside on a remote server for additional sharing of information.

For more information...

See “Developing Database Applications” in the Help contents or search for “Data Dictionary” in the Help index.

Editing existing database tables

You can use the Database Desktop (DBD) to browse and modify existing Paradox and dBASE tables or create and populate new ones, create indexes, define referential integrity, and create database-level validation and business rules for them. You can browse and create BDE aliases as well.

The DBD is a standalone utility that runs outside the Delphi IDE.

For more information...

For detailed information, see the Database Desktop Help file (Dbddesk.hlp). See also “Database Desktop” in the Help index.

Configuring databases

The BDE Administrator is included with Delphi. You use the BDE Administrator to perform tasks such as the following:

- Configure the BDE
- Configure standard (Paradox, dBASE, FoxPro, and ASCII text), SQL, Access, and ODBC drivers; create and delete ODBC drivers
- Create and maintain database aliases

For more information...

See “Developing Database Applications” in the Help contents. For detailed information, see the BDE Administrator Help file (BDEAdmin.hlp).

Understanding database application architecture

You can use Delphi to write database applications of varying complexity. When writing applications that use information that is not shared among several users, you may want to use a local database in a *single-tiered application*. Writing a *two-tiered application* provides more multi-user support and lets you use large remote databases that can store far more information.

Note Support for two-tiered applications requires SQL Links, which is available only in the Client/Server, and Enterprise versions.

When the database information includes complicated relationships, or when the number of clients grows, you may want to use a *multi-tiered application*. In the multi-tiered database model, an application is partitioned into parts that reside on different machines. A client application provides a user interface to the data and passes data requests through an application server.

In Delphi, support for multi-tiered applications is based on the Multi-tiered Distributed Application Services Suite (MIDAS). MIDAS is used to build distributed applications on the Windows platform with DCOM, TCP/IP, or OLE Enterprise. MIDAS provides a suite of advanced components for Delphi, services, and core technologies for multi-tier application development. With MIDAS, you can build thin client applications whose business rules can be maintained on the server and automatically updated on the client. MIDAS helps to implement

- High server availability with fail-over safety
- Load balancing
- Distributed datasets and transaction processing
- Thin client applications
- Automatic database constraint propagation
- High-speed database connectivity
- Reduced network traffic

Choose File|New and look at the Multi-tier page to see available project templates and wizards that you can use as a starting point for developing multi-tiered applications.

Note Support for multi-tiered applications is available only in the Client/Server and Enterprise versions.

For more information...

See “Developing Database Applications” in the Help contents or search for “databases,” “client applications,” and “server applications” in the Help index.

Developing distributed applications

Distributed applications are applications that you can run on various machines and platforms. They work together, typically over a network, to perform a set of related functions. For example, a purchasing application for tracking purchases for a nationwide company might require individual client applications for all the outlets, a main server that would process the requests of those clients, and an interface to a database that stores all the information regarding those transactions. By building a distributed client application (such as a Web-based application), maintaining and updating the individual clients is vastly simplified.

Delphi provides several options for implementing distributed applications:

- TCP/IP applications
- COM and DCOM applications
- CORBA applications
- Database applications

The Client/Server edition of Delphi provides tools that help you to create Web server applications as CGI applications or dynamic-link libraries (DLLs) using a message-oriented approach. In these applications, a client sends a message to a server and gets a message back. Special components on the Internet Component palette page make it easy to create event handlers that are associated with a specific Uniform Resource Identifier (URI) and, when processing is complete, to programmatically construct HTML documents and transfer them to the client.

Delphi Client/Server also provides socket components that let you create an application that can communicate with other systems using TCP/IP or related protocols. Using sockets, you can read and write over connections to other machines without worrying about the details of the networking software. Sockets provide connections based on TCP/IP protocol, but they can also work with related protocols such as Xerox Network System (XNS), Digital’s DECnet, or Novell’s IPX/SPX family.

A server or client application is usually dedicated to a single service such as Hypertext Transfer Protocol (HTTP) or File Transfer Protocol (FTP). Using server sockets, an application that provides one of these services can link to client applications that want to use that service. Client sockets allow an application that uses one of these services to link to server applications that provide the service.

Delphi also supports an object-oriented approach to writing distributed applications. In this model, the client interacts with an object that could be on any machine in the network. CORBA and COM are examples of this model. They are explained in the following sections.

For more information...

See “Developing Distributed Applications” in the Help contents for details on developing client/server applications.

Developing CORBA applications

Delphi provides wizards and classes to make it easy to create distributed applications based on the Common Object Request Broker Architecture (CORBA). CORBA is a specification adopted by the Object Management Group (OMG) to address the complexity of developing distributed object applications. CORBA provides an object-oriented approach to writing distributed applications in contrast to a message-oriented approach such as the one used for HTTP applications.

VisiBroker is Borland’s ORB technology used for developing distributed applications. VisiBroker implements the CORBA 2.0 and IIOP standards developed by the OMG, and can interoperate with other CORBA-compliant Object Request Brokers (ORBs) in a distributed object computing environment. The ORB connects a client application with the objects it wants to use. The ORB handles the details of locating the object, routing the request, and returning the result.

Delphi can be used to develop CORBA objects and clients. The design of a CORBA application is much like any other object-oriented application, except that it includes an additional layer for handling network communication when an object resides on a different machine. This additional layer is handled by special objects called stubs and skeletons. You use the type library editor to define the CORBA interfaces and automatically create the skeleton and stub objects that handle low-level CORBA communication.

For more information...

See “Developing Distributed Applications” in the Help contents or search for “CORBA” in the Help index. Refer to the VisiBroker documentation for information on using VisiBroker.

Developing distributed applications using COM and MTS

COM (Component Object Model) is a client/server object-based model designed by Microsoft that enables interaction between software components and applications. This technology is also referred to as ActiveX, which is a consolidation of OLE and OCX implementations.

COM provides object interoperability using predefined routines called interfaces. COM has a library containing a set of standard interfaces that define the core functionality of a COM object, and a small set of API functions designed for the

purpose of creating and managing COM objects. COM objects can be transparently extended, modified, and updated because unique identifiers are used to create them and to access their interfaces.

Using Delphi to create COM-based applications offers many business solutions, from improving software design by using interfaces internally in an application, to creating objects that can interact with other COM-based API objects on the system, such as the Win95 Shell extensions, Web page support, and DirectX multimedia support.

For more information...

See “Developing Distributed Applications Using COM and MTS” in the Help contents.

Creating COM applications with wizards

Delphi provides wizards and classes to make it easy to implement applications based on COM. With these wizards, you can create simple COM-compatible classes to use within a single application or you can create fully functional Automation servers or ActiveX controls.

Delphi provides wizards to create

- Simple COM objects
- Automation servers
- Automation controllers
- ActiveX servers or ActiveForms
- Microsoft Transaction Server (MTS) objects

When you use Delphi wizards and VCL objects in your application, you are using Delphi’s implementation of the COM specification. Delphi also provides wrappers for additional COM services such as Active Documents.

For more information...

See “Developing Distributed Applications Using COM and MTS” in the Help contents. Search for “ActiveX,” “COM,” and “COM interfaces.”

Using MTS with Delphi

You can use Delphi to create a COM object that can work within the Microsoft Transaction Server (MTS) environment. MTS is a component-based transaction processing system for building, deploying, and managing large intranet and Internet server applications. Even though MTS is not architecturally part of COM, it is designed to extend the capabilities of COM in a large, distributed environment. It provides support for resource pooling, transaction processing, and security.

MTS and MIDAS are complementary technologies. MTS acts as a transaction server that manages transactions for objects. These objects are written by Delphi developers and managed by MTS. The components may have database connections by

indicating they are transactional, or they may not. If these components contain ODBC connections for Microsoft SQL Server, these connections become part of the transaction, and MTS can talk directly with the ODBC Driver. MTS developers using MIDAS gain thin client briefcase mode, automatic constraint and business rule updates, data updates, and support for database engines other than Microsoft SQL Server

For more information...

See “Developing Distributed Applications Using COM and MTS” in the Help contents.

Creating and editing type libraries

Type libraries are files that include information about data types, interfaces, member functions, and object classes exposed by an ActiveX control or server. Type libraries identify what types of objects and interfaces exist on your ActiveX server.

By including a type library with your COM application or ActiveX library, you make information about the objects in your application available to other applications and programming tools. Delphi includes a Type Library editor that enables you to create and edit type libraries. The Type Library editor is a graphical tool that lets developers examine and create type information for ActiveX controls, COM, and CORBA objects and clients.

For more information...

See “Developing Distributed Applications Using COM and MTS” in the Help contents. Search for “Type Library editor,” “COM,” and “CORBA” in the Help index.

Deploying applications

Delphi includes add-on tools to help with application deployment. For example, InstallShield Express helps you to create an installation package for your application that includes all of the files needed for running a distributed application. PVCS Version Manager software is also available for tracking application updates.

If you are writing applications that you plan to distribute internationally, Delphi also has tools and guidelines for internationalizing and localizing applications. For example, you can use the string table editor to edit binary RES files created while using the Resource DLL wizard.

For more information...

Search for “deploying applications” and “international applications” in the Help index.

Building custom components

You may find that you need to solve a specific application programming need by building your own components. When you create a component, you add to the VCL by deriving a new class from one of the existing class types in the hierarchy.

There are four primary reasons to build custom Delphi components:

- To provide additional functionality
- To support reusability
- To increase productivity
- To promote consistency

You can create components in Delphi using the same development environment as you use to develop applications.

For more information...

See “Creating Custom Components” in the Help contents.

Index

A

- accessing
 - databases 3-5
- action list 5-4
- activating property editors 3-13
- ActiveX 5-12
 - controls 4-10
 - servers 5-14
- adding
 - components 3-3
 - overview 2-2
 - memos 3-12
 - objects 3-3 to 3-4
- adding components 4-10
- applications
 - client/server 5-10
 - CORBA 5-12
 - creating 2-11, 3-1
 - database 5-8
 - deploying 5-14
 - designing 3-2, 3-3, 5-2
 - distributed 5-11
 - fixing errors 2-9
 - tiered architecture 5-10
- Automation servers 5-13

B

- background colors 3-3
- BDE Administration utility 5-10
- BIOLIFE.DB 3-6
- Borland Database Engine (BDE) 5-8
- buttons
 - events 3-14

C

- canvases 3-2
- captions
 - removing 3-11
- changing
 - source files 2-4
- child windows 3-2
- click events 3-14
- client/server development 5-10
- code 3-13
 - editing 2-4
 - syntax highlighting 2-8, 4-8
- code completion 2-7

- Code editor 2-4, 2-5
 - browser 2-5
 - context-sensitive help 2-15
 - customizing 4-8
 - illustrated 2-4
 - overview 2-4
- Code Explorer 2-6
- Code Insight 2-7
- code templates 2-7
- coding
 - Help while 2-7
- Color property 3-3
- colors 3-3
- COM (Component Object Model) 5-12
- commands 5-4
- component packages 4-10
- Component palette 3-3 to 3-4
 - adding components 4-10
 - customizing 4-8
 - overview 2-2
- component templates 4-11
- components 2-2, 3-2, 3-3, 5-3, 5-4
 - adding 3-3, 4-10
 - building custom 5-15
 - customizing 2-3
 - defined 2-2
 - overview 2-2
 - predefined 3-3
- connections 2-11
- context menus 2-14, 3-2
- context-sensitive help 2-15
- controls 3-2
- CORBA applications 5-12
- creating
 - applications 3-1
- customizing
 - Code editor 4-8
 - Component palette 4-8
 - components 2-3
 - Delphi 4-1
 - online Help 4-11
 - project options 4-5
 - tool preferences 4-7

D

- Data Dictionary 5-9
- data grids 3-5, 3-6
- data sources 3-5

- data-aware controls 3-6
- database applications 5-8
- Database Desktop (DBD) 5-9
- Database Explorer 2-11, 5-8
- databases 2-11, 3-5
 - accessing 2-11, 3-5
 - architecture 5-10
 - configuring 5-10
 - getting values 3-5
 - sample 3-6
 - saving data 3-14
- DataSource component 3-5
- DBGrid component 3-5, 3-6
- debugging 2-9
- defaults
 - changing project 4-5
 - event handler 3-14
 - package containers 4-10
- DEFPROJ.DOF 4-6
- deleting
 - captions 3-11
- Delphi
 - customizing 4-1
 - development
 - environment 2-1
 - documentation 1-2
 - overview 1-1
 - product versions 5-2
 - programming
 - environment 5-1
 - starting 2-1
 - technical support 1-4
 - toolbars 2-14
- deploying applications 5-14
- designing applications 3-2, 3-3, 5-2
- design-time properties 3-3, 3-13
 - viewing 3-2
- dialog boxes 3-2
 - context-sensitive help 2-15
- displaying
 - event handlers 2-4
 - property values 3-2
- distributed applications 5-11
- docking windows 2-8, 4-3
- DPL file 5-7

E

- editing 3-13
 - code 2-4
- editor 2-5

- ellipse buttons 3-13
- environment options 4-7
- Environment Options dialog box 4-8
- errors 5-7
- event handlers 3-14
 - default 3-14
 - viewing 2-4
- events 2-4
 - navigating 2-4
- Events page (Object Inspector) 2-4
- exception handling 5-7

F

- fields 3-5
- File Save dialog box 3-14
- files
 - saving 3-14
- folders 3-1
- Font editor 3-13
- fonts
 - customizing 3-13
- form wizards 4-7
- forms 3-2
 - adding components 2-2
 - adding objects 3-3 to 3-4
 - specifying default 4-7

G

- grid 3-5

H

- Help 2-15
 - customizing 4-11
 - list of Help files 1-2

I

- IDE 2-1, 5-1
 - adding components 4-10
- initializing
 - objects 3-3
- InstallShield Express 5-14
- instantiation
 - objects 3-4
- interfaces 3-2
 - designing 3-3
- international applications 5-14

M

- memos
 - creating 3-12

- Menu editor 3-13
- menus 3-2
- Microsoft Transaction Server (MTS) 5-13
- MIDAS 5-10
- MTS 5-13
- multi-tiered application 5-10
- Multi-tiered Distributed Application Services Suite (MIDAS) 5-10

N

- New Application command 3-1

O

- object hierarchy 5-2
- Object Inspector 3-2
 - creating event handlers 3-14
 - overview 2-3
 - setting properties 3-3, 3-13
 - viewing properties 3-2
- Object Repository 2-11, 4-6
 - default forms 4-7
- Object Request Broker (ORB) 5-12
- objects 2-11, 3-2
 - adding 3-3 to 3-4
 - initializing 3-3
 - instantiating 3-4
 - setting properties 3-3, 3-13
 - storing as templates 2-11
- OCX 5-12
- ODBC 5-14
- OLE 5-12
- OLEnterprise 5-10
- OnClick event 3-14
- online help 2-15
- OpenHelp 4-11
- Oracle8 5-8

P

- packages 5-7
 - default container 4-10
 - installing 4-10
- palette pages 3-3
- panels
 - removing captions 3-11
- parent windows 3-2
- Picture editor 3-13
- predefined components 3-3
- project files 2-13
- Project Manager
 - overview 2-13

- Project Options dialog box 4-5
- project templates
 - specifying default 4-6
- projects 2-13, 3-1
 - defaults 4-6
 - options 4-5
 - restoring defaults 4-6
- properties
 - initial values 3-3
 - setting 3-3, 3-13
 - viewing 3-2
- property editors 3-13
 - activating 3-13
 - initializing objects and 3-3
- PVCS Version Manager 5-14

R

- remote connections 2-11
- Remote Data Broker 5-10
- remote debugging 2-10
- repository 2-11
- restoring project defaults 4-6
- reusing objects 2-11
- right-click menus 2-14
- runtime packages 5-7

S

- sample database 3-6
- SaveDialog component 3-14
- saving
 - files 3-14
- setting property values
 - tutorial 3-3, 3-13
- single-tiered application 5-10
- source code 2-5
 - default
 - event handlers 3-14
- source files 3-1
 - changing 2-4
- SQL Explorer 2-11, 5-9
- SQL Links 5-10
- support 1-4

T

- templates 2-11
 - creating component 4-11
 - specifying default 4-6
- text
 - adding to forms 3-12
 - controls 3-12
 - setting fonts 3-13
- tool preferences 4-7
- tool windows 2-8

toolbars 2-14

tools

 organizing 4-1

tooltip expression

 evaluation 2-7

type libraries 5-14

U

user interface 2-2, 5-4

V

VCL30 5-7

viewing

 event handlers 2-4

 property values 3-2

VisiBroker 5-12

Visual Component Library
 (VCL) 2-2, 5-2

W

Web site

 Delphi 1-4

windows

 context-sensitive help 2-15

 docking 2-8

wizards

 COM 5-13

 CORBA 5-12

 Object Repository 2-11

 specifying default 4-7

