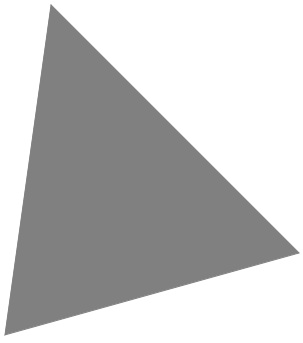
Quick Start



Borland®
Delphi™ 6
for Windows

Borland Software Corporation
100 Enterprise Way, Scotts Valley, CA 95066-3249

Refer to the DEPLOY document located in the root directory of your Delphi 6 product for a complete list of files that you can distribute in accordance with the Delphi License Statement and Limited Warranty.

Borland 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.

Printed in the U.S.A.

HDE1360WW21000 1E0R0501
0102030405-9 8 7 6 5 4 3 2 1
PDF

Contents

Chapter

1

Introduction

This *Quick Start* provides an overview of the Delphi development environment to get you started using the product right away. It also tells you where to look for details about the tools and features available in Delphi.

Chapter 2, "A tour of the desktop" describes the main tools on the Delphi desktop, or integrated desktop environment (IDE). Chapter 3, "Programming with Delphi" explains how you use some of these tools to create an application. Chapter 4, "Creating a text editor—a tutorial" takes you step by step through a tutorial to write a program for a text editor. Chapter 5, "Customizing the desktop" describes how you can customize the Delphi IDE for your development needs.

What is Delphi?

Delphi is an object-oriented, visual programming environment for rapid application development (RAD). Using Delphi, you can create highly efficient applications for Microsoft Windows 2000, Windows 98, and Windows NT with a minimum of manual coding. Delphi also provides a simple cross-platform solution when used in conjunction with Kylix, Borland's RAD tool for Linux. Delphi provides all the tools you need to develop, test, and deploy applications, including a large library of reusable components, a suite of design tools, application and form templates, and programming wizards.

Finding information

You can find information on Delphi in the following ways, described in this chapter:

- Online Help
- Printed documentation
- Borland developer support services and Web site

For information about new features in this release, refer to What's New in the online Help Contents and to the www.borland.com Web site.

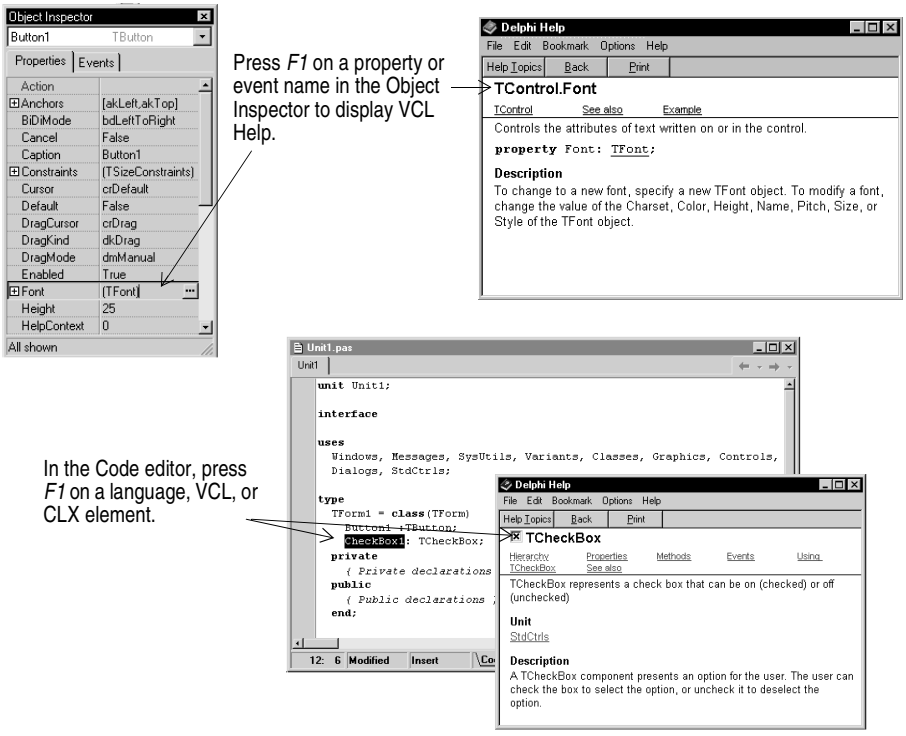
Online Help

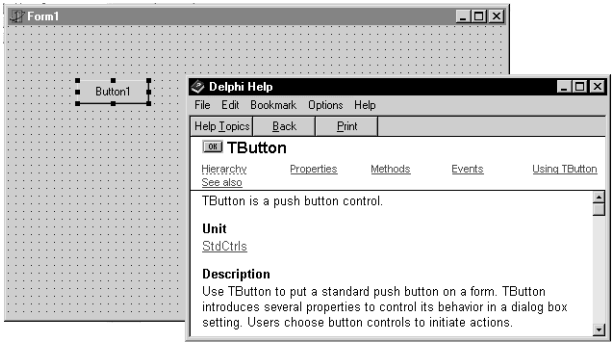
The online Help system provides detailed information about user interface features, language implementation, programming tasks, and the components in the Visual Component Library Reference (VCL) and Borland Component Library for Cross Reference (CLX). It includes all the material in the Delphi *Developer's Guide*, *Object Pascal Language Guide*, and a host of Help files for other features bundled with Delphi.

To view the table of contents, choose Help | Delphi Help and Help | Delphi Tools, and click the Contents tab. To look up VCL or CLX objects or any other topic, click the Index or Find tab and type your request.

F1 Help

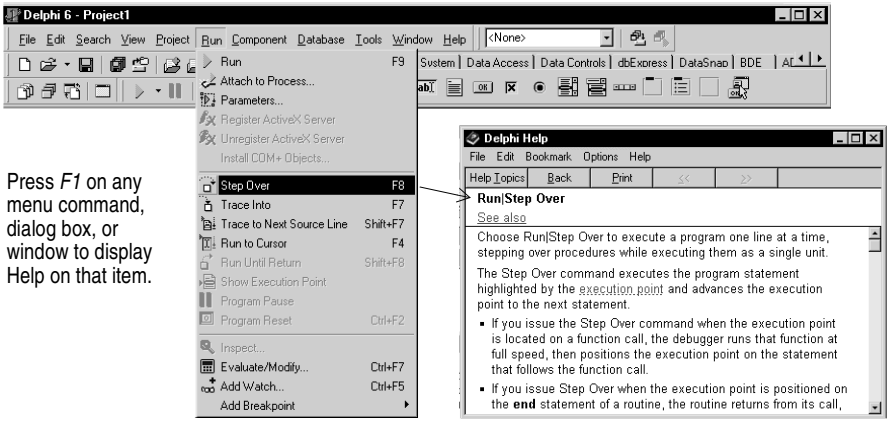
You can get context-sensitive Help on the VCL, CLX, and any part of the development environment, including menu items, dialog boxes, toolbars, and components by selecting the item and pressing *F1*.





Press *F1* on a component on a form.

Pressing the Help button in any dialog box also displays context-sensitive online documentation.



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

Error messages from the compiler and linker appear in a special window below the Code editor. To get Help with compilation errors, select a message from the list and press *F1*.

Printed documentation

This *Quick Start* is an introduction to Delphi. To order additional printed documentation, such as the *Developer's Guide*, refer to shop.borland.com.

Developer support services and Web site

Borland also offers a variety of support options to meet the needs of its diverse developer community. To find out about support, refer to http://www.borland.com/devsupport/.

From the Web site, you can access many newsgroups where Delphi developers exchange information, tips, and techniques. The site also includes a list of books about Delphi, additional Delphi technical documents, and Frequently Asked Questions (FAQs).

Typographic conventions

This manual uses the typefaces described below to indicate special text.

Table 1.1 Typographic conventions

Typeface	Meaning
`Monospace type`	Monospaced type 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.
Italics	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.
Keycaps	This typeface indicates a key on your keyboard. For example, "Press *Esc* to exit a menu."

Chapter

2

A tour of the desktop

This chapter explains how to start Delphi and gives you a quick tour of the main parts and tools of the desktop, or integrated desktop environment (IDE).

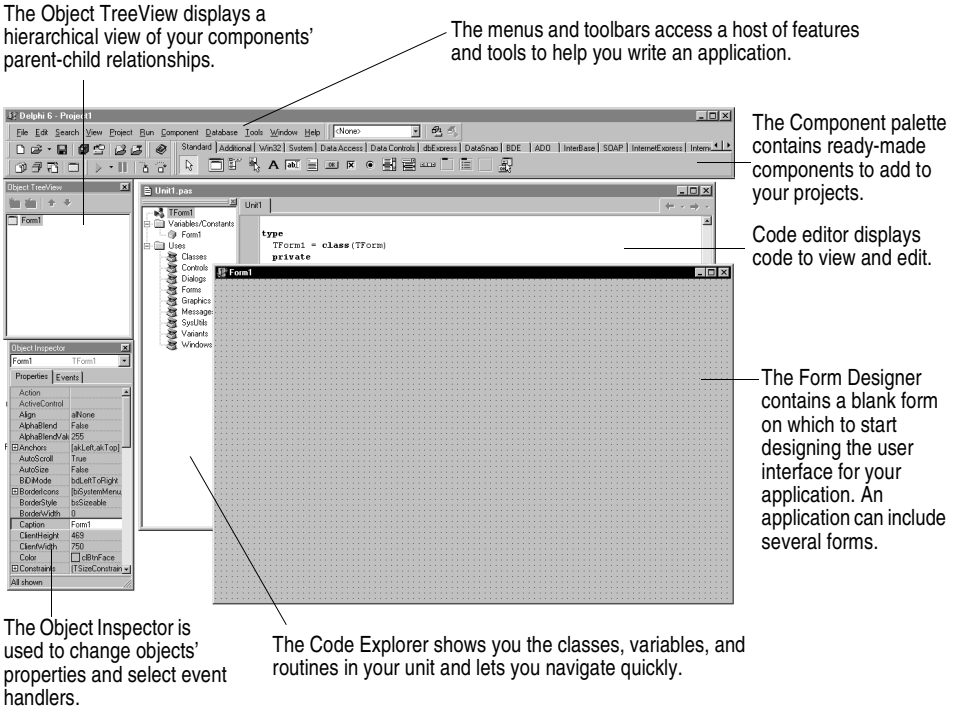
Starting Delphi

You can start Delphi in the following ways:

- Double-click the Delphi icon (if you've created a shortcut).
- Choose Programs | Borland Delphi 6 | Delphi 6 from the Windows Start menu.
- Choose Run from the Windows Start menu, then enter Delphi32.
- Double-click Delphi32.exe in the Delphi\Bin directory.

The IDE

When you first start Delphi, you'll see some of the major tools in the IDE. In Delphi, the IDE includes the menus, toolbars, Component palette, Object Inspector, Object TreeView, Code editor, Code Explorer, Project Manager, and many other tools. The particular features and components available to you will depend on which edition of Delphi you've purchased.



Delphi's development model is based on *two-way* tools. This means that you can move back and forth between visual design tools and text-based code editing. For example, after using the Form Designer to arrange buttons and other elements in a graphical interface, you can immediately view the form file that contains the textual description of your form. You can also manually edit any code generated by Delphi without losing access to the visual programming environment.

From the IDE, all your programming tools are within easy reach. You can design graphical interfaces, browse through class libraries, write code, and compile, test, debug, and manage projects without leaving the IDE.

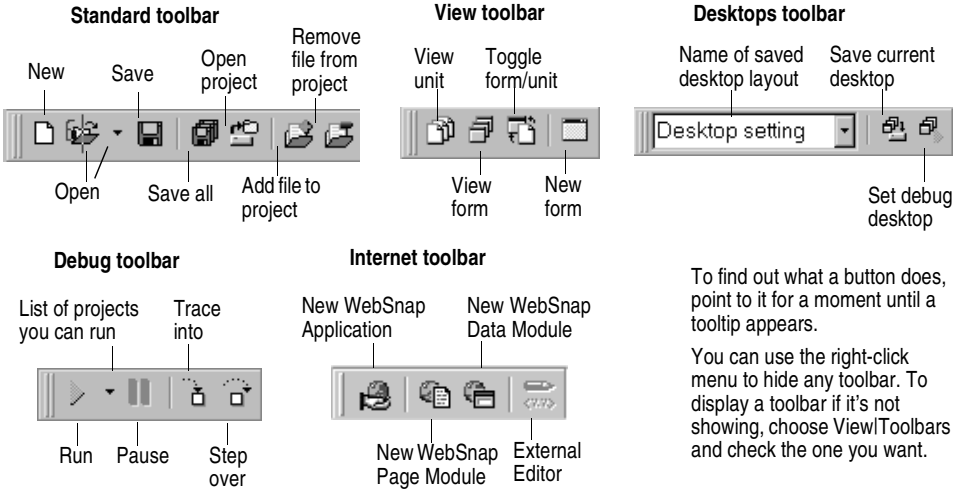
To learn about organizing and configuring the IDE, see Chapter 5, "Customizing the desktop."

The menus and toolbars

The main window, which occupies the top of the screen, contains the main menu, toolbars, and Component palette.

Main window in its default arrangement.

Delphi's toolbars provide quick access to frequently used operations and commands. Most toolbar operations are duplicated in the drop-down menus.



To find out what a button does, point to it for a moment until a tooltip appears.

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

Many operations have keyboard shortcuts as well as toolbar buttons. When a keyboard shortcut is available, it is always shown next to the command on the drop-down menu.

You can right-click on many tools and icons to display a menu of commands appropriate to the object you are working with. These are called *context menus*.

The toolbars are also customizable. You can add commands you want to them or move them to different locations. For more information, see "Arranging menus and toolbars" on page 5-1 and "Saving desktop layouts" on page 5-4.

For more information...

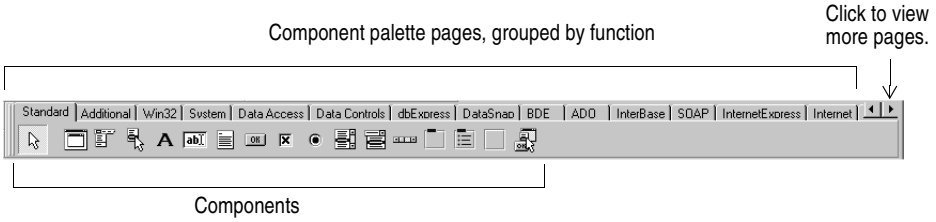
If you need help on any menu option, point to it and press *F1*.

The Component Palette, Form Designer, and Object Inspector

The Component palette, Form Designer, Object Inspector, and Object TreeView work together to help you build a user interface for your application.

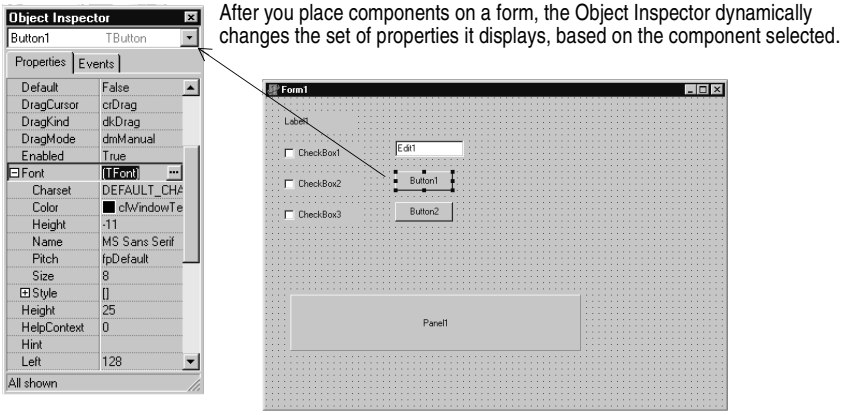
The *Component palette* includes tabbed pages with groups of icons representing visual or nonvisual VCL and CLX components. The pages divide the components into various functional groups. For example, the Standard, Additional, and Win32 pages include windows controls such as an edit box and up/down button; the Dialogs page

includes common dialog boxes to use for file operations such as opening and saving files.



Each component has specific attributes—properties, events, and methods—that enable you to control your application.

After you place components on the form, or *Form Designer*, you can arrange components the way they should look on your user interface. For the components you place on the form, use the *Object Inspector* to set design-time properties, create event handlers, and filter visible properties and events, making the connection between your application's visual appearance and the code that makes your application run. See "Placing components on a form" on page 3-2.



For more information...

See "Component palette" in the online Help index.

The Object TreeView

The Object TreeView displays a component's sibling and parent-child relationships in a hierarchical, or tree diagram. The tree diagram is synchronized with the Object Inspector and the Form Designer so that when you change focus in the Object TreeView, both the Object Inspector and the form change focus.

You can use the Object TreeView to change related components' relationships to each other. For example, if you add a panel and check box component to your form, the

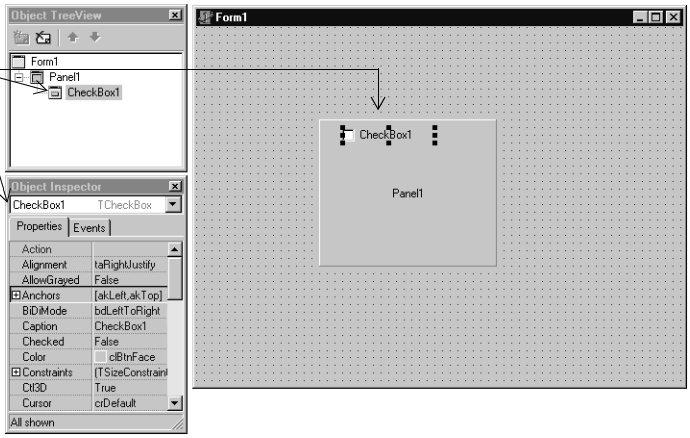
two components are siblings. But in the Object TreeView, if you drag the check box on top of the panel icon, the check box becomes the child of the panel.

If an object's properties have not been completed, the Object TreeView displays a red question mark next to it. You can also double-click any object in the tree diagram to open the Code editor to a place where you can write an event handler.

If the Object TreeView isn't displayed, choose View | Object TreeView.

The Object TreeView, Object Inspector, and the Form Designer work together. When you click an object on your form, it automatically changes the focus in both the Object TreeView and the Object Inspector and vice versa.

Press *Alt-Shift-F11* to focus on the Object TreeView.



The Object TreeView is especially useful for displaying the relationships between database objects.

For more information...

See "Object TreeView" in the online Help index.

The Object Repository

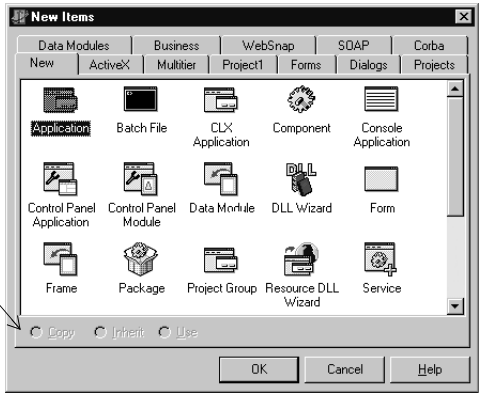
The Object Repository contains forms, dialog boxes, data modules, wizards, DLLs, sample applications, and other items that can simplify development. Choose File | New | Other to display the New Items dialog box when you begin a project. The New

Items dialog box is the same as the Object Repository. Check the Repository to see if it contains an object that resembles one you want to create.

The Repository's tabbed pages include objects like forms, frames, units, and wizards to create specialized items.

When you're creating an item based on one from the Object Repository, you can copy, inherit, or use the item:

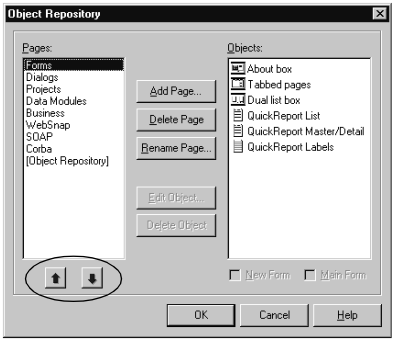
Copy (the default) creates a copy of the item in your project. *Inherit* means changes to the object in the Repository are inherited by the one in your project. *Use* means changes to the object in your project are inherited by the object in the Repository.



To edit or remove objects from the Object Repository, either choose Tools | Repository or right-click in the New Items dialog box and choose Properties.

You can add, remove, or rename tabbed pages from the Object Repository.

Click the arrows to change the order in which a tabbed page appears in the New Items dialog box.



To add project and form templates to the Object Repository, see "Adding templates to the Object Repository" on page 5-10.

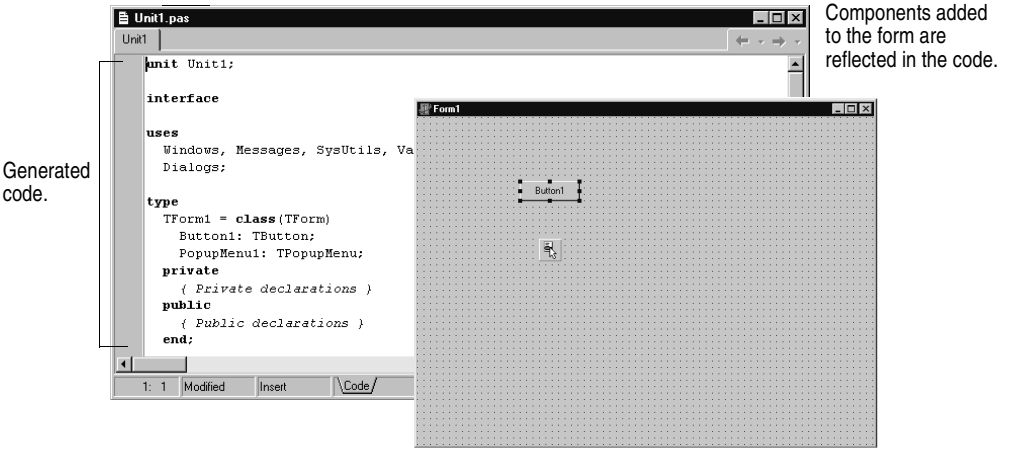
For more information...

See "Object Repository" in the online Help index. The objects available to you will depend on which edition of Delphi you purchased.

The Code Editor

As you design the user interface for your application, Delphi generates the underlying Object Pascal code. When you select and modify the properties of forms and objects, your changes are automatically reflected in the source files. You can add

code to your source files directly using the built-in Code editor, which is a full-featured ASCII editor.



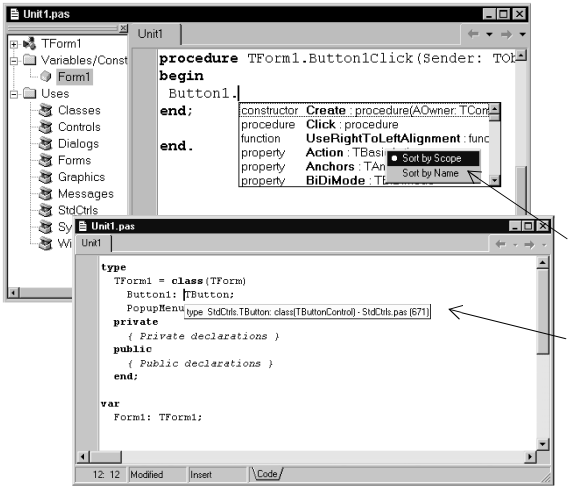
Delphi provides various aids to help you write code, including the Code Insight tools, class completion, and code browsing.

Code Insight

The Code Insight tools display context-sensitive pop-up windows.

Table 2.1 Code Insight tools

Tool	How it works
Code completion	Type a class name followed by a dot (.) to display a list of properties, methods, and events appropriate to the class, select it, and press *Enter*. In the **interface** section of your code you can select more than one item. Type the beginning of an assignment statement and press *Ctrl+space* to display a list of valid values for the variable. Type a procedure, function, or method name to bring up a list of arguments.
Code parameters	Type a method name and an open parenthesis to display the syntax for the method's arguments.
Tooltip expression evaluation	While your program has paused during debugging, point to any variable to display its current value.
Tooltip symbol insight	While editing code, point to any identifier to display its declaration.
Code templates	Press *Ctrl+J* to see a list of common programming statements that you can insert into your code. You can create your own templates in addition to the ones supplied with Delphi.



With code completion, when you type the dot in `Button1.` Delphi displays a list of properties, methods, and events for the class. As you type, the list automatically filters to the selection that pertains to that class. Select an item on the list and press *Enter* to add it to your code.

Procedures and properties are colored as teal and functions as blue.

You can sort this list alphabetically by right-clicking and clicking Sort by Name.

The tooltip symbol insight displays declaration information for any identifier when you pass the mouse over it.

To turn these tools on or off, choose Tools | Editor Options and click the Code Insight tab. Check or uncheck the tools in the Automatic features section.

Class Completion

Class completion generates skeleton code for classes. Place the cursor anywhere within a class declaration of the **interface** section of a unit and press *Ctrl+Shift+C* or right-click and choose Complete Class at Cursor. Delphi automatically adds private **read** and **write** specifiers to the declarations for any properties that require them, then creates skeleton code for all the class's methods. You can also use class completion to fill in class declarations for methods you've already implemented.

To turn on class completion, choose Tools | Environment Options, click the Explorer tab, and make sure Finish incomplete properties is checked.

For more information...

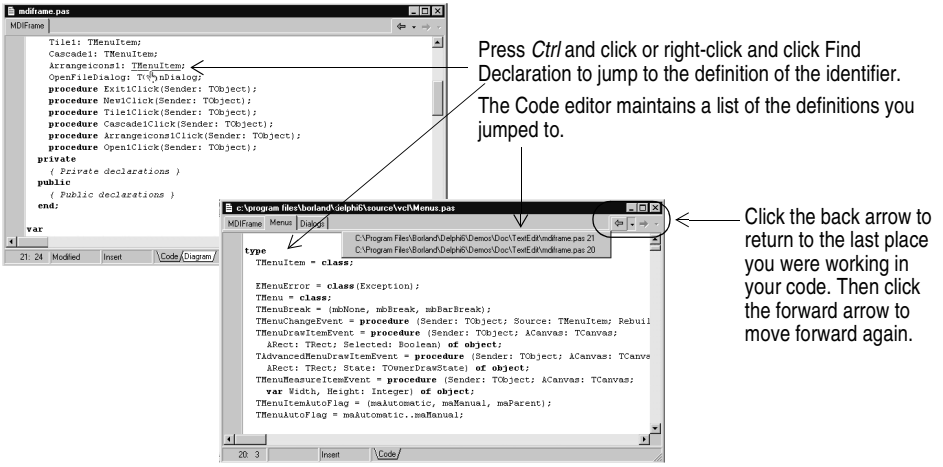
See "Code Insight" and "class completion" in the online Help index.

Code Browsing

While passing the mouse over the name of any class, variable, property, method, or other identifier, the pop-up menu called Tooltip Symbol Insight displays where the identifier is declared. Press *Ctrl* and the cursor turns into a hand, the identifier turns blue and is underlined, and you can click to jump to the definition of the identifier.

The Code editor has forward and back buttons like the ones on Web browsers. As you jump to these definitions, the Code editor keeps track of where you've been in

the code. You can click the drop-down arrows next to the Forward and Back buttons to move forward and backward through a history of these references.



You can also move between the declaration of a procedure and its implementation by pressing *Ctrl+Shift+*↑ or *Ctrl+Shift+*↓.

To customize your code editing environment, see "Customizing the Code Editor" on page 5-12.

For more information...

See "Code editor" in the online Help index.

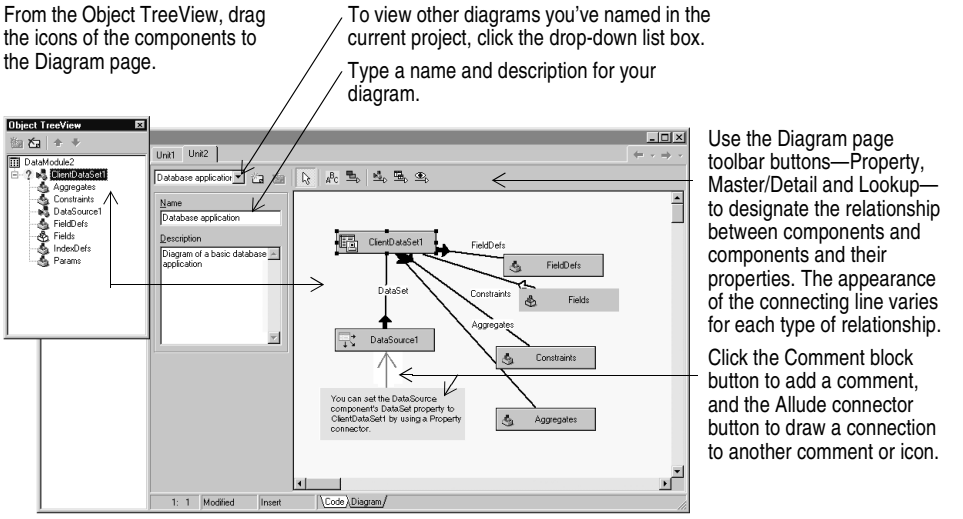
The Diagram page

The bottom of the Code editor may contain one or more tabs, depending on which edition of Delphi you have. The Code page, where you write all your code, appears in the foreground by default. The Diagram page displays icons and connecting lines representing the relationships between the components you place on a form or data module. These relationships include siblings, parent to children, or components to properties.

To create a diagram, click the Diagram page. From the Object TreeView, simply drag one or multiple icons to the Diagram page to arrange them vertically. To arrange them horizontally, press *Shift* while dragging. When you drag icons with parent-children or component-property dependencies onto the page, the lines, or *connectors*, that display the dependent relationships are automatically added. For example, if you add a dataset component to a data module and drag the dataset icon plus its property icons to the Diagram page, the property connector automatically connects the property icons to the dataset icon.

For components that don't have dependent relationships but where you want to show one, use the toolbar buttons at the top of the Diagram page to add one of four

connector types, including allude, property, master/detail, and lookup. You can also add comment blocks that connect to each other or to a relevant icon.



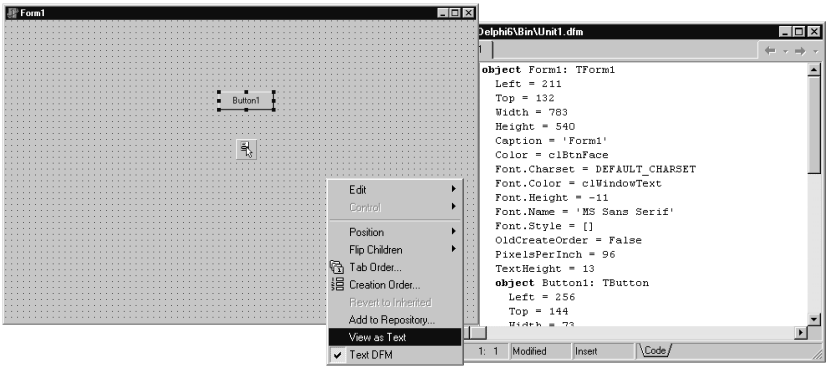
You can type a name and description for your diagram, save the diagram, and print it when you are finished.

For more information...

See "diagram page" in the online Help index.

Viewing form code

Forms are a very visible part of most Delphi projects—they are where you design the user interface of an application. Normally, you design forms using Delphi's visual tools, and Delphi stores the forms in form files. Form files (.dfm, or .xfm for a CLX application) describe each component in your form, including the values of all persistent properties. To view and edit a form file in the Code editor, right-click the form and select View as Text. To return to the graphic view of your form, right-click and choose View as Form.



Use View As Text to view a text description of the form's attributes in the Code editor.

You can save form files in either text (the default) or binary format. Choose Tools | Environment Options, click the Designer page, and check or uncheck the New forms as text check box to designate which format to use for newly created forms.

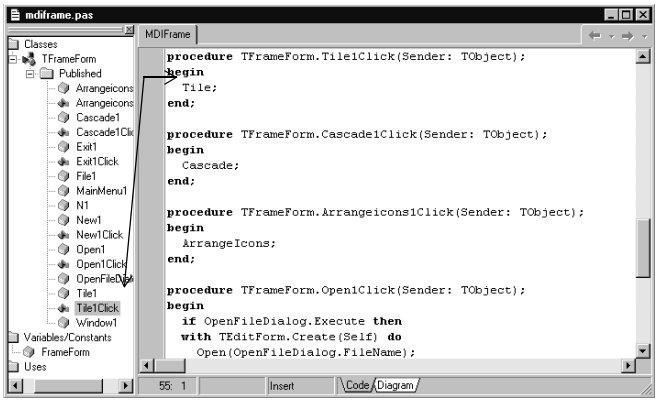
For more information...

See "form files" in the online Help index.

The Code Explorer

When you open Delphi, the Code Explorer is docked to the left of the Code editor window, depending on whether the Code Explorer is available in the edition of Delphi you have. The Code Explorer displays the table of contents as a tree diagram for the source code open in the Code editor, listing the types, classes, properties, methods, global variables, and routines defined in your unit. It also shows the other units listed in the **uses** clause.

You can use the Code Explorer to navigate in the Code editor. For example, if you double-click a method in the Code Explorer, a cursor jumps to the definition in the class declaration in the interface part of the unit in the Code editor.



Double-click an item in the Code Explorer and the cursor moves to that item's implementation in the Code editor. Press *Ctrl+Shift+E* to move the cursor back and forth between the last place you were in the Code Explorer and Code editor.

Each item in the Code Explorer has an icon that designates its type.

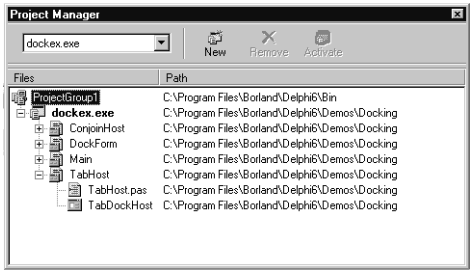
To configure how the Code Explorer displays its contents, choose Tools | Environment Options and click the Explorer tab. See "Customizing the Code Explorer" on page 5-12.

For more information...

See "Code Explorer" in the online Help index.

The Project Manager

When you first start Delphi, it automatically opens a new project, as shown on page 2-2. A project includes several files that make up the application or DLL you are going to develop. You can view and organize these files—such as form, unit, resource, object, and library files—in a project management tool called the Project Manager. To display the Project Manager, choose View | Project Manager.



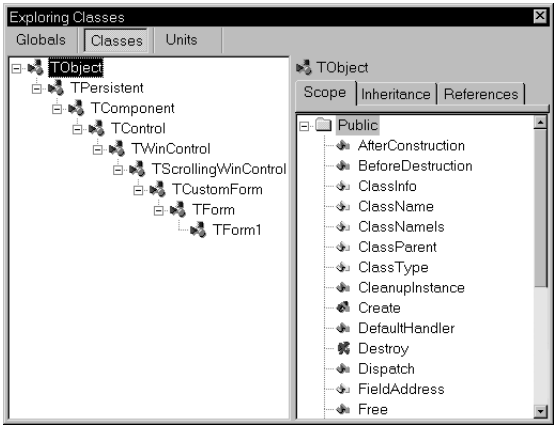
You can use the Project Manager to combine and display information on related projects into a single *project group*. By organizing related projects into a group, such as multiple executables, you can compile them at the same time. To change project options, such as compiling a project, see "Setting project options" on page 5-9.

For more information...

See "Project Manager" in the online Help index.

The Project Browser

The Project Browser examines a project in detail. The Browser displays classes, units, and global symbols (types, properties, methods, variables, and routines) your project declares or uses in a tree diagram. Choose View | Browser to display the Project Browser.



The Project Browser has two resizeable panes: the Inspector pane (on the left) and the Details pane. The Inspector pane has three tabs for globals, classes, and units.

Globals displays classes, types, properties, methods, variables, and routines.

Classes displays classes in a hierarchical diagram.

Units displays units, identifiers declared in each unit, and the other units that use and are used by each unit.

By default, the Project Browser displays the symbols from units in the current project only. You can change the scope to display all symbols available in Delphi. Choose Tools | Environment Options, and on the Explorer page, check All symbols (VCL included).

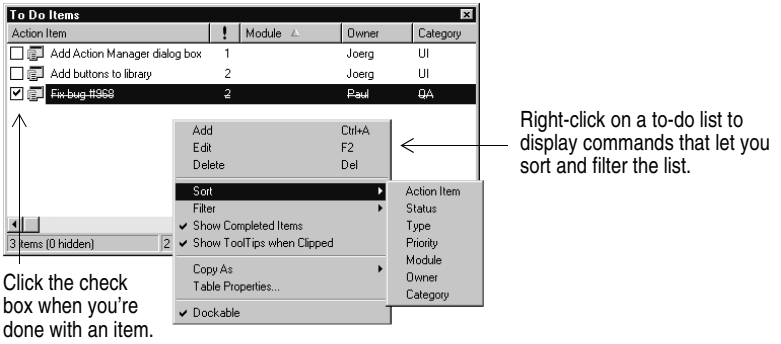
For more information...

See “Project Browser” in the online Help index.

To-do lists

To-do lists record items that need to be completed for a project. You can add project-wide items to a list by adding them directly to the list, or you can add specific items

directly in the source code. Choose View | To-Do List to add or view information associated with a project.



For more information...

See "to-do lists" in the online Help index.

Chapter

3

Programming with Delphi

The following sections provide an overview of software development with Delphi, including creating a project, working with forms, writing code, and compiling, debugging, deploying, and internationalizing applications, and including the types of projects you can develop.

Creating a project

A project is a collection of files that are either created at design time or generated when you compile the project source code. When you first start Delphi, a new project opens. It automatically generates a project file (Project1.dpr), unit file (Unit1.pas), and resource file (Unit1.dfm; Unit1.xfm for CLX applications), among others.

If a project is already open but you want to open a new one, choose either File | New | Application or File | New | Other and double-click the Application icon. File | New | Other opens the Object Repository, which provides additional forms, modules, and frames as well as predesigned templates such as dialog boxes to add to your project. To learn more about the Object Repository, see "The Object Repository" on page 2-5.

When you start a project, you have to know what you want to develop, such as an application or DLL. To read about what types of projects you can develop with Delphi, see "Types of projects" on page 3-8.

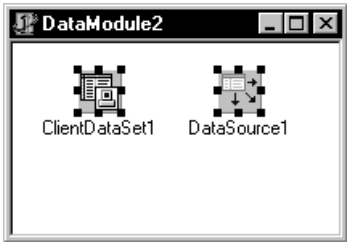
For more information...

See "projects" in the online Help index.

Adding data modules

A data module is a type of form that contains nonvisual components only. Nonvisual components *can* be placed on ordinary forms alongside visual components. But if you plan on reusing groups of database and system objects, or if you want to isolate the parts of your application that handle database connectivity and business rules, data modules provide a convenient organizational tool.

To create a data module, choose File | New | Data Module. Delphi opens an empty data module, which displays an additional unit file for the module in the Code Editor, and adds the module to the current project as a new unit. Add nonvisual components to a data module in the same way as you would to a form.



Double-click a nonvisual component on the Component palette to place the component in the data module.

When you reopen an existing data module, Delphi displays its components.

For more information...
See "data modules" in the online Help index.

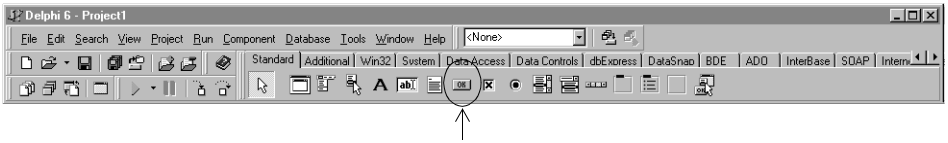
Building the user interface

With Delphi, you first create a user interface (UI) by selecting components from the Component palette and placing them on the main form.

Placing components on a form

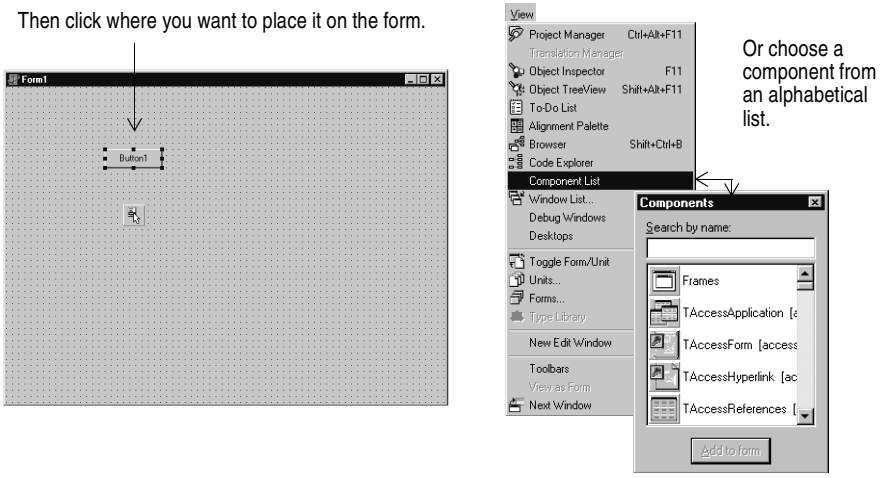
To place components on a form, either:

1. Double-click the component; or
2. Click the component once and then click the form where you want the component to appear.



Click a component on the Component palette.

Select the component and drag it to wherever you want on the form.

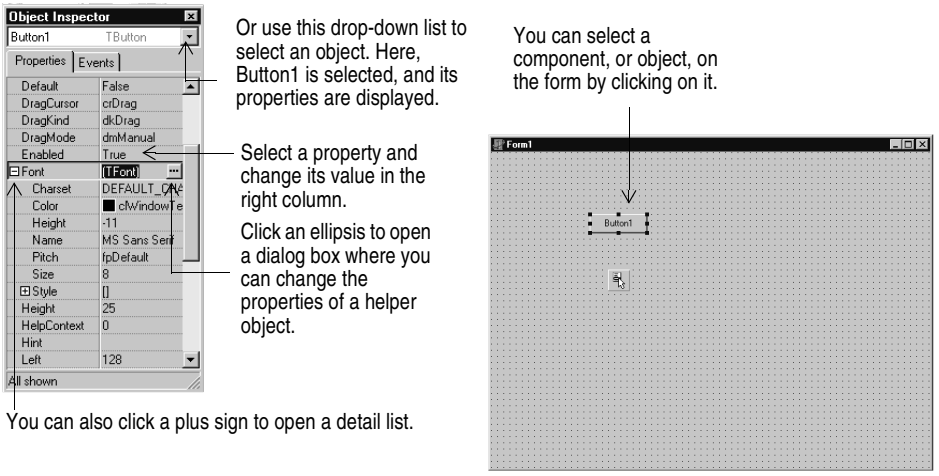


For more information...

See “Component palette” in the online Help index.

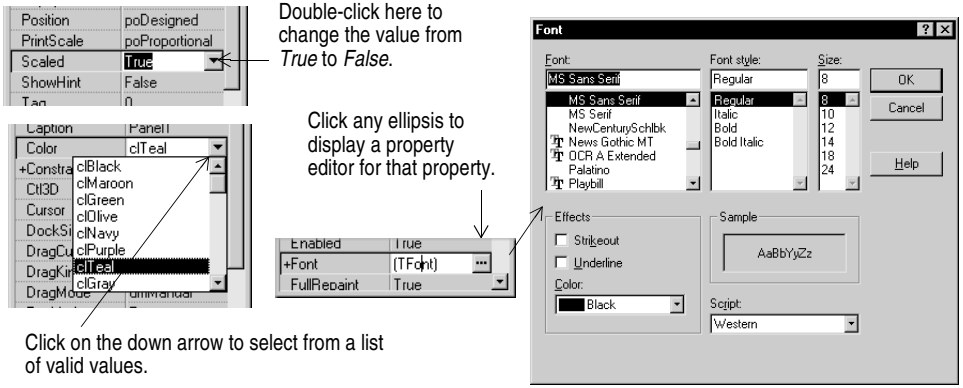
Setting component properties

After you place components on a form, set their properties and code their event handlers. Setting a component’s properties changes the way a component appears and behaves in your application. When a component is selected on a form, its properties and events are displayed in the Object Inspector.



Many properties have simple values—such as names of colors, *True* or *False*, and integers. For Boolean properties, you can double-click the word to toggle between *True* and *False*. Some properties have associated property editors to set more complex

values. When you click on such a property value, you'll see an ellipsis. For some properties, such as size, enter a value.

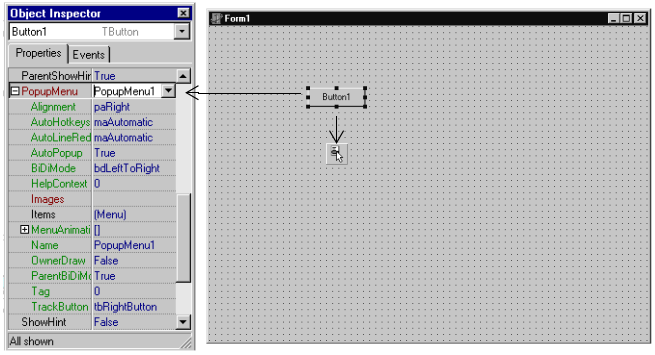


When more than one component is selected in the form, the Object Inspector displays all properties that are shared among the selected components.

The Object Inspector also supports expanded inline component references. This provides access to the properties and events of a referenced component without having to select the referenced component itself. For example, if you add a button and pop-up menu component to your form, when you select the button component, in the Object Inspector you can set the *PopupMenu* property to `PopupMenu1`, which displays all of the pop-up menu's properties.

Set the Button component's *PopupMenu* property to `PopupMenu1`, and all of the popup menu's properties appear when you click the plus sign (+).

Inline component references are colored red, and their subproperties are colored green.



For more information...

See "Object Inspector" in the online Help index.

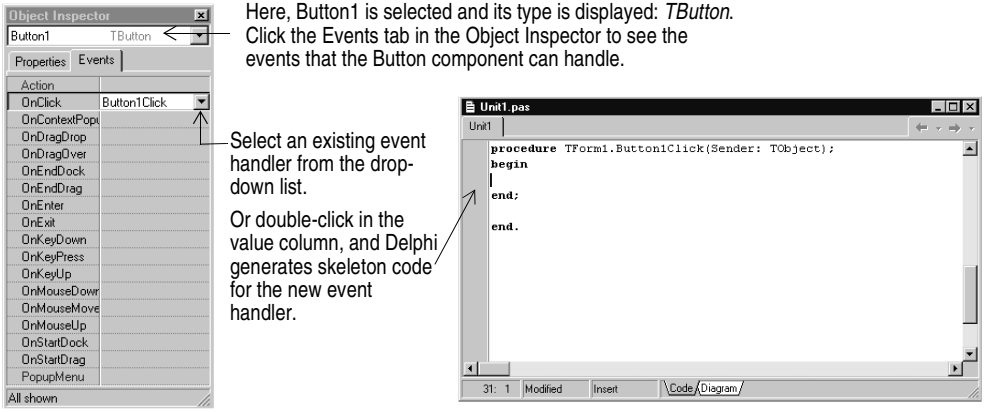
Writing code

An integral part of any application is the code behind each component. While Delphi's RAD environment provides most of the building blocks for you, such as preinstalled visual and nonvisual components, you will usually need to write event handlers, methods, and perhaps some of your own classes. To help you with this task, you can choose from thousands of objects in Delphi's VCL and CLX class libraries. To work with your source code, see "The Code Editor" on page 2-6.

Writing event handlers

Your code may need to respond to events that might occur to a component at runtime. An event is a link between an occurrence in the system, such as clicking a button, and a piece of code that responds to that occurrence. The responding code is an event handler. This code modifies property values and calls methods.

To view predefined event handlers for a component on your form, select the component and, on the Object Inspector, click the Events tab.



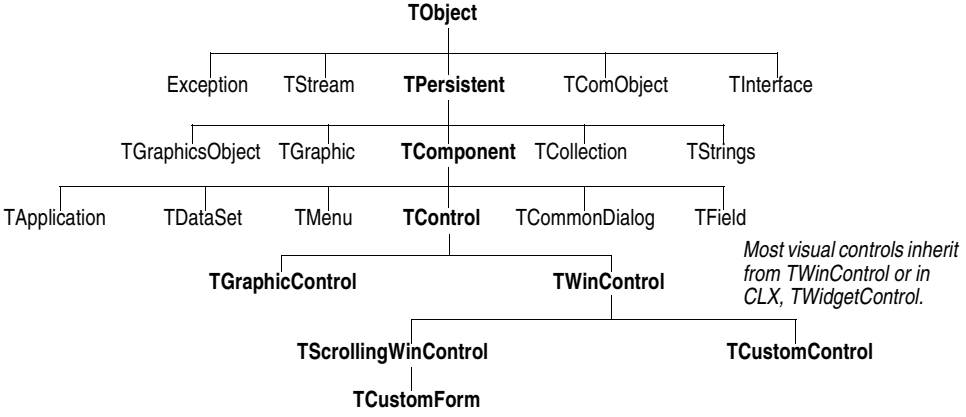
For more information...

See "events" in the online Help index.

Using the VCL and CLX libraries

Delphi comes with two class libraries made up of objects, some of which are also components or controls, that you use when writing code. You can use the Visual Component Library (VCL) for Windows applications and Borland Component Library for Cross Platform (CLX) for Linux applications. These libraries include objects that are visible at runtime—such as edit controls, buttons, and other user interface elements—as well as nonvisual controls like datasets and timers. The

following diagram below shows some of the principal classes that make up the VCL The CLX hierarchy is similar.



Objects descended from *TComponent* have properties and methods that allow them to be installed on the Component palette and added to Delphi forms and data modules. Because VCL and CLX components are hooked into the IDE, you can use tools like the Form Designer to develop applications quickly.

Components are highly encapsulated. For example, buttons are preprogrammed to respond to mouse clicks by firing *OnClick* events. If you use a VCL or CLX button control, you don't have to write code to handle generated events when the button is clicked; you are responsible only for the application logic that executes in response to the click itself.

Most editions of Delphi come with VCL and CLX source code and examples of Object Pascal programming techniques.

For more information...

See "Visual Component Library Reference" and "CLX Reference" in the Help contents and "VCL" in the online Help index. See http://www.borland.com/delphi for open source and licensing options on CLX.

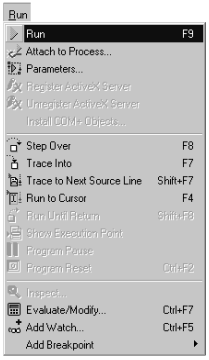
Compiling and debugging projects

After you have written your code, you will need to compile and debug your project. With Delphi, you can either compile your project first and then separately debug it, or you can compile and debug in one step using the integrated debugger. To compile your program with debug information, choose Project | Options, click the Compiler page, and make sure Debug information is checked.

Delphi uses an integrated debugger so that you can control program execution, watch variables, and modify data values. You can step through your code line by line, examining the state of the program at each breakpoint. To use the integrated

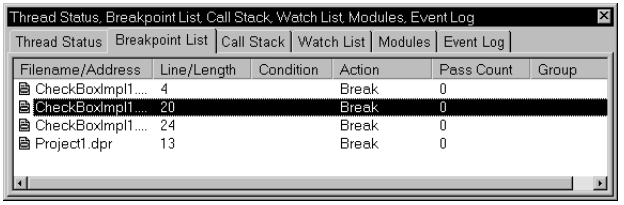
debugger, choose Tools | Debugger Options, click the General page, and make sure Integrated debugging is checked.

You can begin a debugging session in the IDE by clicking the Run button on the Debug toolbar, choosing Run | Run, or pressing F9.



Choose any of the debugging commands from the Run menu. Some commands are also available on the toolbar.

With the integrated debugger, many debugging windows are available, including Breakpoints, Call Stack, Watches, Local Variables, Threads, Modules, CPU, and Event Log. Display them by choosing View | Debug Windows. Not all debugger views are available in all editions of Delphi.



You can combine several debugging windows for easier use.

To learn how to combine debugging windows for more convenient use, see “Docking tool windows” on page 5-2.

Once you set up your desktop as you like it for debugging, you can save the settings as the debugging or runtime desktop. This desktop layout will be used whenever you are debugging any application. For details, see “Saving desktop layouts” on page 5-4.

For more information...

See “debugging” and “integrated debugger” in the online Help index.

Deploying applications

You can make your application available for others to install and run by deploying it. When you deploy an application, you will need all the required and supporting files, such as the executables, DLLs, package files, and helper applications. Delphi comes bundled with a setup toolkit called InstallShield Express that helps you create an installation program with these files. To install InstallShield Express, from the Delphi setup screen, choose InstallShield Express Custom Edition for Delphi.

For more information...

See "deploying, applications" in the online Help index.

Internationalizing applications

Delphi offers several features for internationalizing and localizing applications. The IDE and the VCL support input method editors (IMEs) and extended character sets to internationalize your project. Delphi includes a translation suite, not available in all editions of Delphi, for software localization and simultaneous development for different locales. With the translation suite, you can manage multiple localized versions of an application as part of a single project.

The translation suite includes three integrated tools:

- Resource DLL wizard, a DLL wizard that generates and manage resource DLLs.
- Translation Manager, a table for viewing and editing translated resources.
- Translation Repository, a shared database to store translations.

To open the Resource DLL wizard, choose File | New | Other and double-click the Resource DLL Wizard icon. To configure the translation tools, choose Tools | Translation Tools Options.

For more information...

See "international applications" in the online Help index.

Types of projects

All editions of Delphi support general-purpose 32-bit Windows programming, DLLs, packages, custom components, multithreading, COM (Component Object Model) and automation controllers, and multiprocess debugging. Some editions support server applications such as Web server applications, database applications, COM servers, multi-tiered applications, CORBA, and decision-support systems.

For more information...

To see what tools your edition supports, refer to the feature list on www.borland.com/delphi.

CLX applications

With Delphi, you can develop a cross-platform application that can be ported to Kylix, where you compile, debug, and deploy your project to run on Linux. To develop a CLX application, choose File | New | CLX Application. The IDE is similar to that of a regular Delphi application, except that only the components and items you can use in a CLX application appear on the Component palette and in the Object Repository. Windows-specific features supported on Delphi will not port directly to Linux environments.

For more information...

To see which components are available for developing cross-platform applications, see "CLX Reference" in the online Help contents.

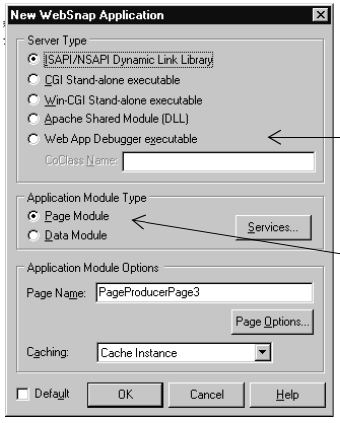
Web server applications

A Web server application works with a Web server by processing a client's request and returning an HTTP message in the form of a Web page. To publish data for the Web, Delphi includes two different technologies, depending on what edition of Delphi you have.

To develop a basic Web server application, you create a Web module to dispatch requests, define actions, create HTML pages, and write event handlers for both Windows and Linux applications. To create a WebBroker Web server application, choose File | New | Other and double-click the Web Server Application icon. You can add components to your Web module from the Internet and InternetExpress Component palette pages.

WebSnap adds to this functionality with adapters, additional dispatchers, additional page producers, session support, and Web page modules. To create a new WebSnap server application, select File | New | Other, click the WebSnap page, and double-click the Web Server Application icon. You can add WebSnap components from the WebSnap Component palette page.

You can also access the WebSnap Application data module by choosing View| Toolbars|Internet, and clicking the New WebSnap Application icon.



You can create an application to run on various Web server application types, including a test server to help you debug your Web server application.

Choose whether you want a data module or a page module, which displays your HTML page as you work.

For more information...
See “Web applications” in the online Help index.

Database applications

Delphi offers a variety of database and connectivity tools to simplify the development of database applications.

To create a database application, first design your interface on a form using the Data Controls page components. Second, add a data source to a data module using the Data Access page. Third, to connect to various database servers, add a dataset and data connection component to the data module from the previous or corresponding pages of the following connectivity tools:

- dbExpress is a collection of database drivers for cross-platform applications that provide fast access to SQL database servers, including DB2, InterBase, MySQL, and Oracle. With a dbExpress driver, you can access databases using unidirectional datasets.
- The Borland Database Engine (BDE) is a collection of drivers that support many popular database formats, including dBASE, Paradox, FoxPro, Microsoft Access, and any ODBC data source. SQL Links drivers, available with some versions of Delphi, support servers such as Oracle, Sybase, Informix, DB2, SQL Server, and InterBase.
- ActiveX Data Objects (ADO) is Microsoft's high-level interface to any data source, including relational and nonrelational databases, e-mail and file systems, text and graphics, and custom business objects.
- InterBase Express (IBX) components are based on the custom data access Delphi component architectures. IBX applications provide access to advanced InterBase features and offer the highest performance component interface for InterBase 5.5 and later. IBX is compatible with Delphi’s library of data-aware components.

Certain database connectivity tools are not available in all editions of Delphi.

For more information...
See “database applications” in the online Help index.

BDE Administrator

Use the BDE Administrator (BDEAdmin.exe) to configure BDE drivers and set up the aliases used by data-aware VCL controls to connect to databases.

For more information...
From the Windows Start menu, choose Programs | Borland Delphi 6 | BDE Administrator. Then choose Help | Contents.

SQL Explorer (Database Explorer)

The SQL Explorer (DBExplor.exe) lets you browse and edit databases. You can use it to create database aliases, view schema information, execute SQL queries, and maintain data dictionaries and attribute sets.

For more information...

From the Delphi main menu, choose Database | Explore. Then choose Help | Contents. Or see "Database Explorer" in the online Help index.

Database Desktop

The Database Desktop (DBD32.exe) lets you create, view, and edit Paradox and dBase database tables in a variety of formats.

For more information...

From the Windows Start menu, choose Programs | Borland Delphi 6 | Database Desktop. Then choose Help | User's Guide Contents.

Data Dictionary

When you use the BDE, 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. The Data Dictionary can reside on a remote server to share additional information.

For more information...

Choose Help | Delphi Tools to see "Data Dictionary."

Custom components

The components that come with Delphi are preinstalled on the Component palette and offer a range of functionality that should be sufficient for most of your development needs. You could program with Delphi for years without installing a new component, but you may sometimes want to solve special problems or display particular kinds of behavior that require custom components. Custom components promote code reuse and consistency across applications.

You can either install custom components from third-party vendors or create your own. To create a new component, choose Component | New Component to display the New Component wizard. To install components provided by a third party, see "Installing component packages" on page 5-7.

For more information...

See Part V, "Creating custom components," in the *Developer's Guide* and "components, creating" in the online Help index.

DLLs

Dynamic-link libraries (DLLs) are compiled modules containing routines that can be called by applications and by other DLLs. A DLL contains code or resources typically used by more than one application. Choose File | New | Other and double-click the DLL Wizard icon to create a template for a DLL.

For more information...
See "DLLs" in the online Help index.

COM and ActiveX

Delphi supports Microsoft's COM standard and provides wizards for creating ActiveX controls. Choose File | New | Other and click the ActiveX tab to access the wizards. Sample ActiveX controls are installed on the ActiveX page of the Component palette. Numerous COM server components are provided on the Servers tab of the Component palette. You can use these components as if they were VCL components. For example, you can place one of the Microsoft Word components onto a form to bring up an instance of Microsoft Word within an application interface.

For more information...
See "COM" and "ActiveX" in the online Help index.

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. By including a type library with your COM application or ActiveX library, you make information about these entities available to other applications and programming tools. Delphi provides a Type Library editor for creating and maintaining type libraries.

For more information...
See "type libraries" in the online Help index.

Chapter

4

Creating a text editor—a tutorial

This tutorial takes you through the creation of a text editor complete with menus, a toolbar, and a status bar.

Note This tutorial is for all editions of Delphi and is for the Windows platform only.

Starting a new application

Before beginning a new application, create a directory to hold the source files:

1. Create a directory called TextEditor in your C:\Program Files\Borland\Delphi6\Projects directory.
2. Open a new project.

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

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

   - *Project1.dpr*: a source-code file associated with the project. This is called a *project file*.
   - *Unit1.pas*: a source-code file associated with the main project form. This is called a *unit file*.
   - *Unit1.dfm*: a resource file that stores information about the main project form. This is called a *form file*.

   Each form has its own unit (*Unit1.pas*) and form (*Unit1.dfm*) files. If you create a second form, a second unit (*Unit2.pas*) and form (*Unit2.dfm*) file are automatically created.

3 Choose File | Save All to save your files to disk. When the Save dialog box appears:

- Navigate to your TextEditor folder.
- Save Unit1 using the default name Unit1.pas.
- Save the project using the name TextEditor.dpr. (The executable will be named the same as the project name with an .exe extension.)

Later, you can resave your work by choosing File | Save All.

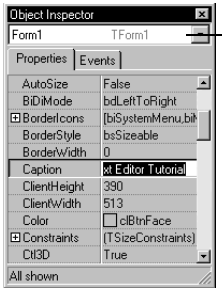
When you save your project, Delphi creates additional files in your project directory. These files include TextEditor.dof, which is the Delphi Options file, TextEditor.cfg, which is the configuration file, and TextEditor.res, which is the Windows resource file. You don't need to worry about these files but don't delete them.

Setting property values

When you open a new project, Delphi displays the project's main form, named *Form1* by default. You'll create the user interface and other parts of your application by placing components on this form.

Next to the form, you'll see the Object Inspector, which you can use to set property values for the form and the components you place on it. When you set properties, Delphi maintains your source code for you. The values you set in the Object Inspector are called *design-time* settings.

1 Find the form's *Caption* property in the Object Inspector and type `Text Editor Tutorial` replacing the default caption `Form1`. Notice that the caption in the heading of the form changes as you type.



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

When a component is selected, the Object Inspector displays its properties.

2 Run the form now by pressing *F9*, even though there are no components on it.

Without any components on it, the runtime view of the form looks similar to the design-time view, complete with the Minimize, Maximize, and Close buttons.

3 To return to the design-time view of Form1, do one of the following:

- Click the **X** in the upper right corner of the title bar of your application (the runtime view of the form);
- Click the Exit application button in the upper left corner of the title bar and click Close;
- Choose View | Forms, select Form1, and click OK; or
- Choose Run | Program Reset.

Adding components to the form

Before you start adding components to the form, you need to think about the best way to create the user interface (UI) for your application. The UI is what allows the user of your application to interact with it and should be designed for ease of use.

Delphi includes many components that represent parts of an application. For example, there are components (derived from *objects*) on the Component palette that make it easy to program menus, toolbars, dialog boxes, and many other visual and nonvisual program elements.

The text editor application requires an editing area, a status bar for displaying information such as the name of the file being edited, menus, and perhaps a toolbar with buttons for easy access to commands. The beauty of designing the interface using Delphi is that you can experiment with different components and see the results right away. This way, you can quickly prototype an application interface.

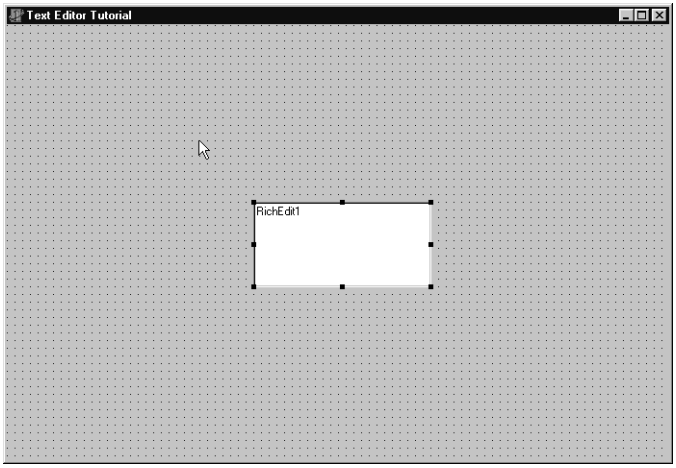
To start designing the text editor, add a *RichEdit* and a *StatusBar* component to the form:

1 To create a text area, first add a *RichEdit* component. To find the *RichEdit* component, on the Win32 page of the Component palette, point to an icon on the

palette for a moment; Delphi displays a Help tooltip showing the name of the component.

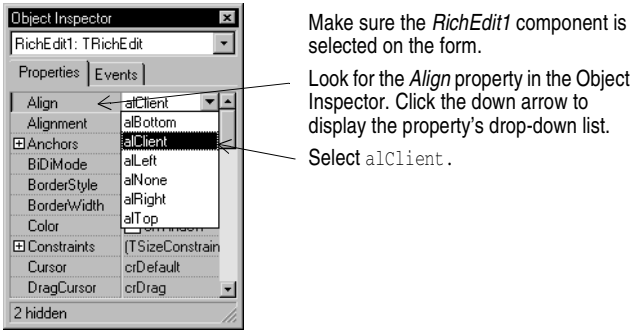
When you find the *RichEdit* component, either:

- Select the component on the palette and then click on the form where you want to place the component; or
- Double-click the component to place it in the middle of the form.



Each Delphi component is a *class*; placing a component on a form creates an *instance* of that class. Once the component is on the form, Delphi generates the code necessary to construct an instance of the object when your application is running.

2 With the *RichEdit* component selected, in the Object Inspector, click the drop-down arrow of the *Align* property and set it to `alClient`.



The *RichEdit* component now fills the entire form so you have a large text editing area. By choosing the `alClient` value for the *Align* property, the size of the *RichEdit* control will vary to fill whatever size window is displayed even if the form is resized.

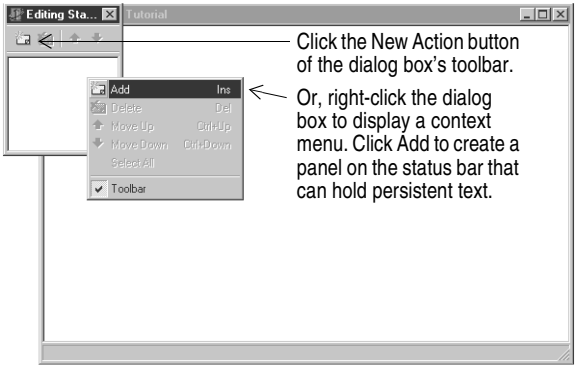
3 Double-click the *StatusBar* component on the Win32 page of the Component palette. This adds a status bar to the bottom of the form.

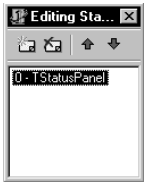
4 To create one panel on the status bar to display the path and file name of the file being edited by your text editor:

- Make sure the status bar is selected.
- After the *SimpleText* property, type `untitled.txt`. When you use the text editor, if the file being edited is not yet saved, the file name will be untitled.txt.

- Click the *(TStatusPanel)* ellipse of the *Panels* property to open the Editing StatusBar1.Panels dialog box.
- Click the New Action button on the toolbar of the dialog box to add a panel to the status bar.

Tip You can also access the Editing StatusBar1.Panels dialog box by double-clicking the status bar on your form.





The *Panels* property is a zero-based array so that you can access each panel you create based on its unique index value. By default, the first panel has a value of 0.

Each time you click Add, you add an additional panel to the status bar.

5 Click the **X** to close the Editing StatusBar1.Panels dialog box.

Now the main editing area of the user interface for the text editor is set up.

Adding support for a menu and a toolbar

For the application to do anything, it needs a menu, commands, and, for convenience, a toolbar. Though you can code the commands separately, Delphi provides an *action manager* to help centralize the code and an *image list* to centralize the images to add to the commands on your menus and toolbar.

By convention, the actions that are connected to menu commands are named with the name of the top-level menu and the command name. For example, the FileExit action refers to the Exit command on the File menu.

Following are the kinds of actions our sample text editor application needs:

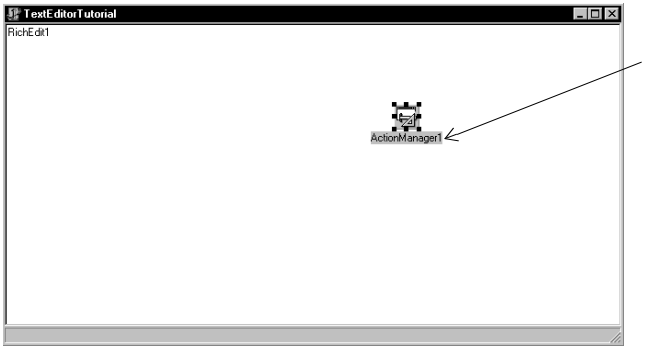
Table 4.1 Planning Text Editor commands

Menu	Command	On Toolbar?	Description
File	New	Yes	Creates a new file.
File	Open	Yes	Opens an existing file for editing.
File	Save	Yes	Saves the current file to disk.
File	Save As	No	Saves a file using a new name (also lets you save a new file using a specified name).
File	Exit	Yes	Quits the editor program.
Edit	Cut	Yes	Deletes text and stores it in the clipboard.
Edit	Copy	Yes	Copies text and stores it in the clipboard.
Edit	Paste	Yes	Inserts text from the clipboard.
Help	Contents	No	Displays the Help contents screen from which you can access Help topics.
Help	Index	No	Displays the Help index screen.
Help	About	No	Displays information about the application in a box.

To centralize both the code and images in an action manager, you need to add the Action Manager editor to your project:

1 On the Additional page of the Component palette, double-click the *ActionManager* component to drop it onto the form. Because it is nonvisual, you can place it anywhere on the form.

2 To display the captions for nonvisual components you drop on the form, choose Tools | Environment Options, click the Designer page, and select Show component captions, and click OK.



To display the captions for the components you place on a form, choose Tools|Environment Options|Designer and click Show component captions.

Because the *ActionManager* component is nonvisual, you cannot see it when the application is running.

Adding actions to the action manager

First you'll add the actions to the action manager and set their properties. By convention, you'll name actions that are connected to menu commands with the name of the top-level menu and the command name. For example, the FileExit action refers to the Exit command on the File menu.

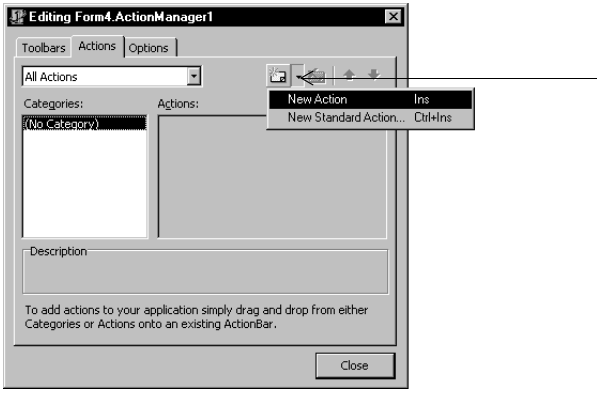
You will add both actions for which you set all the properties, and standard actions, which have their properties automatically set.

1 Double-click the *ActionManager* component to open it.

The Editing Form1.ActionManager1 dialog box, or Action Manager editor, appears.

2 Make sure the Actions tab is displayed. Click the drop-down arrow next to the New Action button and click New Action.

Tip You can also right-click the Action Manager editor and choose New Action.



Click the drop-down arrow next to the New Action button to create new actions for the action manager.

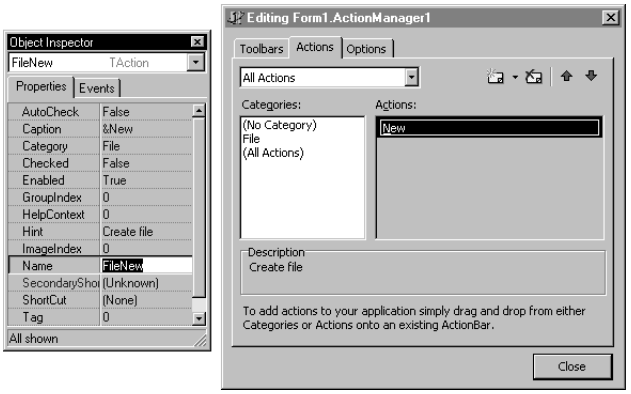
When the Delete button is activated, you can remove existing actions from the actions list.

3 With No Category selected, in the Actions list, click Action1. In the Object Inspector, set the following properties:

- After *Caption*, type `&New`. Note that typing an ampersand before one of the letters makes that letter a shortcut to accessing the command.
- After *Category*, type `File` (this organizes the File commands in one place).
- After *Hint*, type `Create file` (this will be the Help tooltip).
- After *ImageIndex*, type `6` (this will associate image number 6 in your ImageList with this action).
- After *Name*, type `FileNew` (for the File | New command) and press *Enter* to save the change.

With Action1 selected in the Action Manager editor, change its properties in the Object Inspector.

Caption is the name of the action, *Category* is the type of action, *Hint* is a Help tooltip, *ImageIndex* lets you refer to an image in the image list, and *Name* is what the action called in the code.



4 Click the drop-down arrow next to the New Action button and click New Action.

5 With No Category selected, click Action1. In the Object Inspector, set the following properties:

- After *Caption*, type `&Save`.
- Click the drop-down arrow after *Category* and click `File`.
- After *Hint*, type `Save file`.
- After *ImageIndex*, type `8`.
- After *Name*, enter `FileSave` (for the File | Save command).

6 Click the drop-down arrow next to the New Action button and click New Action.

7 With No Category selected, click Action1. In the Object Inspector, set the following properties:

- After *Caption*, type `&Index`.
- After *Category*, type `Help`.
- After *Name*, enter `HelpIndex` (for the Help | Index command).

8 Click the drop-down arrow next to the New Action button and click New Action.

9 Next to (No Category), select Action1. In the Object Inspector, set the following properties:

- After *Caption*, type `&About`.
- Make sure *Category* says `Help`.
- After *Name*, enter `HelpAbout` (for the Help | About command).

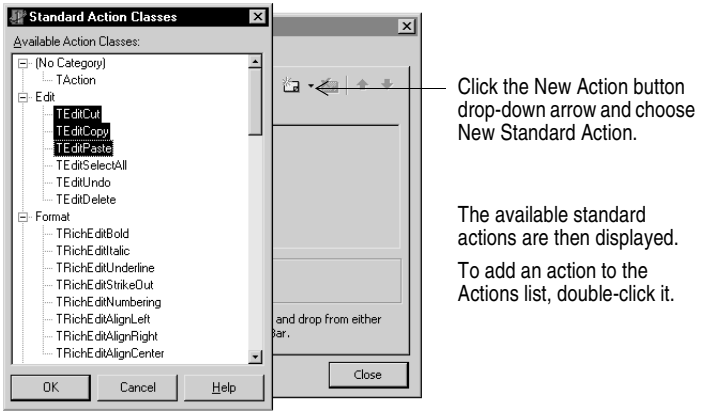
10 Keep the Action Manager editor on the screen.

Adding standard actions to the action manager

Next you'll add the standard actions (open, save as, exit, cut, copy, paste, and help contents) to the action manager.

1. The Action Manager editor should still be displayed. If it's not, double-click the *ActionManager* component to open it.
2. Click the drop-down arrow next to the New Action button and click New Standard Action.

   The Standard Actions Classes dialog box appears.
3. In the Standard Actions Classes dialog box, scroll to the Edit category and select the *TEditCut*, *TEditCopy*, and *TEditPaste*. Click OK to add these actions to a new Edit category in the Categories list of the Editing Form1.ActionManager1 dialog box

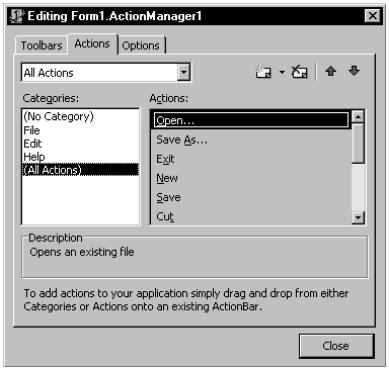


4. Click the drop-down arrow next to the New Action button and click New Standard Action.
5. Scroll to the File category and select the *TFileOpen*, *TFileSaveAs*, and *TFileExit* actions. Click OK to add these actions to the File category.
6. Click the drop-down arrow next to the New Action button and click New Standard Action.
7. In the Standard Actions Classes dialog box, scroll to the Help category and select the *THelpContents*. Click OK to add this action to the Help category.

Note Adding a custom Help | Contents command will display a Help file with a Help Contents tab. The standard Help | Contents command brings up the last tabbed page that was displayed, either Contents, Index, or Find.

Now you've added all the standard actions you need for your application. The standard actions have their properties set automatically, including the image index.

8 Click (All Actions) to display both nonstandard and standard actions that you just added.



Clicking All Actions displays the actions you just added for every category.

9 Click the Close button to close the Action Manager editor.

10 Click File | Save All to save your changes.

Adding images to the image list

In this section, you'll add images to the action manager for use on the menus and toolbar.

The standard actions are associated with preassigned images from a built-in image list that comes with Delphi. For example, the image index for the Edit | Cut action is 0. All of the images you will use for your text editor commands are in this file.

To add the image list:

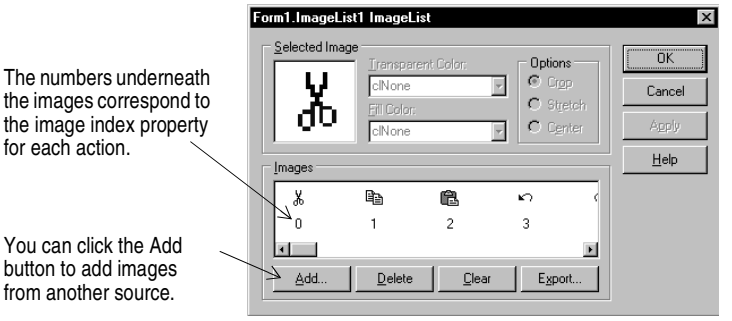
1 If you installed Delphi to the default directory, open C:\Program Files\Borland\Delphi6\Source\Vcl\ActnRes.pas. The StandardsActions window opens.

2 Select the *ImageList1* component and copy and paste it to your form. It is a nonvisual component, so it doesn't matter where you paste it. The ActnRes.pas unit is added to the Code editor.

- To copy *ImageList1*, right-click the component, and click Edit | Copy. On your form, right-click, and choose Edit | Paste.

3 Close the StandardActions window.

4 Double-click *ImageList1* to display all the possible images you can use.



Following are the image index numbers that are used for each command:

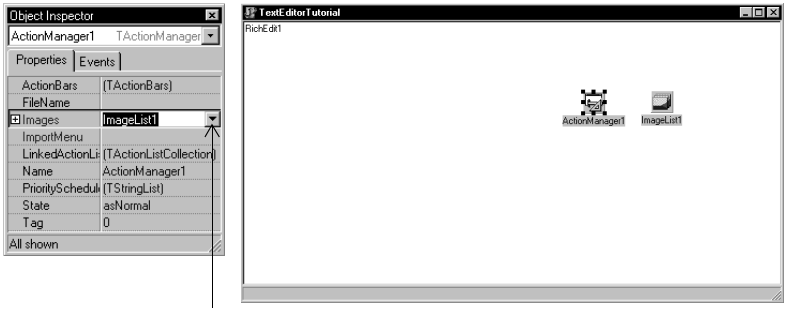
Command	ImageIndex property
Edit	Cut	0
Edit	Copy	1
Edit	Paste	2
File	New	6
File	Open	7
File	Save	8
File	SaveAs	30
File	Exit	43
Help	Contents	40

Note You can add images from an entirely different list. In the Form1.ImageList dialog box, click the Add button and navigate to the Buttons directory provided with the product. The default location is C:\Program Files\Common Files\Borland Shared\Images\Buttons.

For the File | Open command, for example, double-click fileopen.bmp. When a message asks if you want to separate the bitmap into two separate ones, click Yes. Each of the images includes an active and a grayed out version of the image. You'll see both images. Delete the grayed out (second) image. Then make sure the image index in the Object Inspector matches the new number assigned to this image in the image list.

5 Click OK to close the ImageList dialog box.

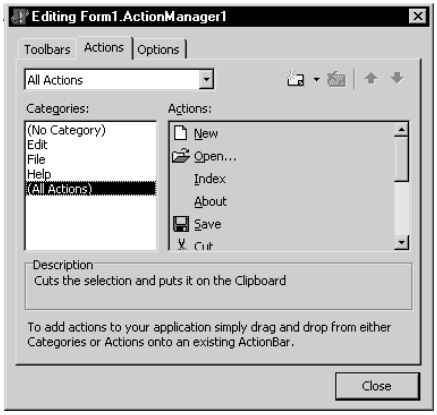
6 Select the *ActionManager* component and set its *Images* property to `ImageList1`.



Click the down arrow next to the *Images* property. Select ImageList1. This associates the images that you'll add to the image list with the actions in the action manager.

Because you already set an image index for all of your actions, the images are added to the correct action automatically. You've associated 8 images with your actions.

7 To see the associated images in the action manager, open the *ActionManager* component, make sure the Actions tab is selected, and click the All Actions category.



When you display the Action Manager editor now, you'll see the images associated with the actions.

8 Choose File | Save All to save your changes.

9 Keep the Action Manager editor open.

Now you're ready to add the menu and toolbar.

Adding a menu

In the next two sections, you'll add a customizable menu bar and toolbar, called *action bands*.

The main menu bar includes three drop-down menus—File, Edit, and Help—and their menu commands. With the Action Manager editor, you can drag each menu category and its commands onto the menu bar in one step.

1 From the Additional page of the Component palette, double-click a *ActionMainMenuBar* component to add it to the form.

   A blank menu bar appears at the top of the form.

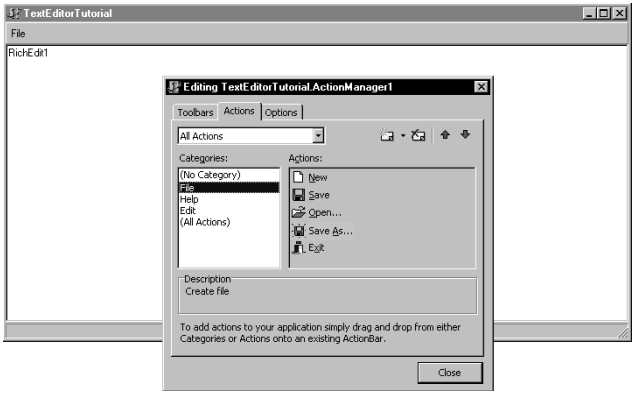
2 Open the Action Manager editor if it isn't already and select File in the Categories list. The submenu commands are not in the exact order that you want them, but you can easily change this by using the Move Up and Move Down buttons, or *Ctrl+*↑ and *Ctrl+*↓.

   - Select the Open action and click the Move Up button on the Action Manager editor toolbar, so that the File commands are listed in the following order: New, Open, Save, Save As, and Exit.

3 Drag File to the menu bar. The File menu and its submenu commands appear on the menu bar.

Tip You can also reposition menu commands after you've dragged the menu category to the menu bar. For example, click File on the menu bar so its submenu commands appear, and drag Open above New and then back again.

When you select the File category from the Action Manager editor and drag it to the menu bar, you drag all its submenu commands with it.



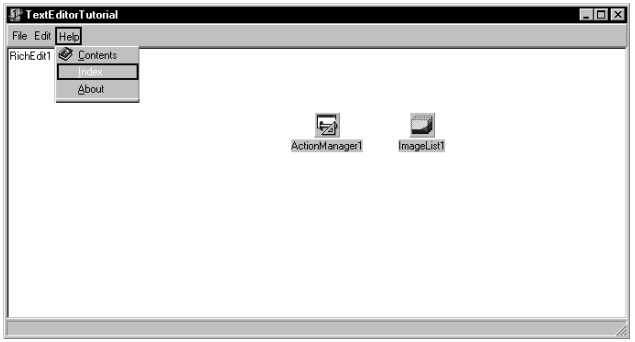
4 From the Categories list of the Action Manager editor, drag Edit to the right of File on the menu bar.

5 From the Categories list of the Action Manager editor, drag Help to the right of the Edit on the menu bar.

6 Click the Help menu to view its submenu commands. Drag the Contents command to above the Index command.

You can change the position of submenu commands in two ways:

You can select an action in the Action Manager editor and click the Move Up or Move Down button. Or, after you drag the Help category to the menu bar, drag Contents above About.



7 Press *Esc* or click the Help menu again to close it.

8 Choose File | Save All to save your changes.

Now you'll want to add a toolbar to provide easy access to the commands.

Adding a toolbar

Since you've set up actions in an action manager, you can add some of the same actions that were used on the menus to an action band toolbar, which will resemble a Microsoft Office 2000 toolbar when you're finished with it.

1 On the Additional page of the Component palette, double-click the *ActionToolBar* component to add it to the form.

A blank toolbar appears under the menu bar.

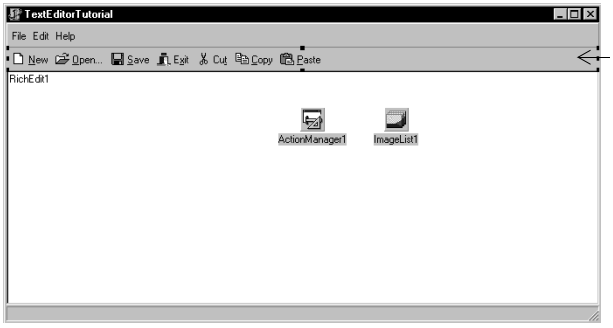
Tip You can also add an action band toolbar by opening the Action Manager editor, clicking the Toolbars tab, and clicking the New button.

2 If the Action Manager editor isn't displayed, open it and select File in the Categories list.

3 In the Actions list, select New, Open, Save, and Exit and drag these items to the toolbar. They automatically appear as buttons with each assigned image.

4 In the Action Manager editor, select Edit in the Categories list.

- In the Actions list, select Cut, Copy, and Paste and drag these items to the toolbar.



The *ActionToolBar* component is added under the menu bar by default.

You can move the menu toolbar above the menu bar and vice versa by dragging it.

You can drag buttons on and off the toolbar.

If you drag the wrong command onto the toolbar, you can drag it off again. Or you can also select the item in the Object TreeView and click the delete key. You can reposition the buttons simply by dragging them to the right or left of each other.

5 Choose File | Save All to save your changes.

6 Press *F9* to compile and run the project.

Tip You can also run the project by clicking the Run button on the Debug toolbar or choosing Run | Run.

When you run your project, Delphi opens the program in a runtime window like the one you designed. The menus and toolbar buttons work although some of the commands are grayed out.

Your text editor already has lots of functionality. You can type in the text area. If you select text in the text area, the Cut, Copy, and Paste buttons should work. However, there's still more to do to activate the commands.

7 To return to design mode, click **X** in the upper right corner.

Clearing the text area (optional)

When you ran your program, the name *RichEdit1* appeared in the text area. You can remove that text using the Strings List Editor. If you don't clear the text now, the text should be removed when initializing the main form in the last step.

To clear the text area:

1 On the main form, click the *RichEdit1* component.

2 In the Object Inspector, next to the *Lines* property, double-click the value *(TStrings)* to display the Filter editor.

3 Select and delete the text (*RichEdit1*) you want to remove in the Filter editor and click OK.

4 Save your changes and trying running the program again.

The text editing area is now cleared when the main form is displayed.

Writing event handlers

Up to this point, you've developed your application without writing a single line of code. By using the Object Inspector to set property values at design time, you've taken full advantage of Delphi's RAD environment. In this section, you'll write procedures called *event handlers* that respond to user input while the application is running. You'll connect the event handlers to the items on the menus and toolbar, so that when an item is selected your application executes the code in the handler.

For the nonstandard actions, you must create an event handler. For the standard actions, such as the File | Exit and Edit | Paste commands, the events are included in the code. However, for some of the standard actions, such as the File | Save As command, you will want to write your own event handler to customize the command.

Because all the menu items and toolbar actions are consolidated in the Action Manager editor, you can create the event handlers from there.

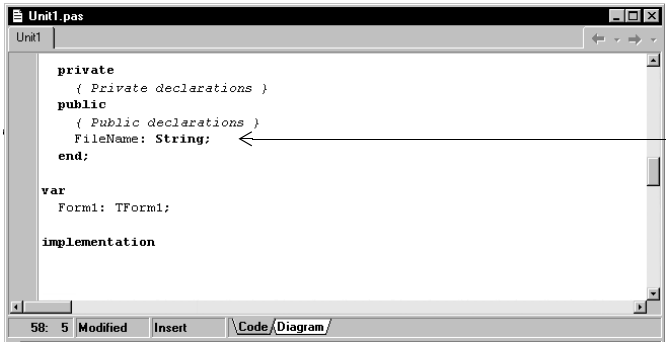
Creating an event handler for the New command

To create an event handler for the New command:

1. Choose View | Units and select Unit1 to display the code associated with Form1.
2. You need to declare a file name that will be used in the event handler, adding a custom property for the file name to make it globally accessible. Early in the Unit1.pas file, locate the public declarations section for the class TForm1 and on the line after `{ Public declarations }`, type:

   ```
   FileName: String;
   ```

   Your screen should look like this:



3. Press *F12* to go back to the main form.

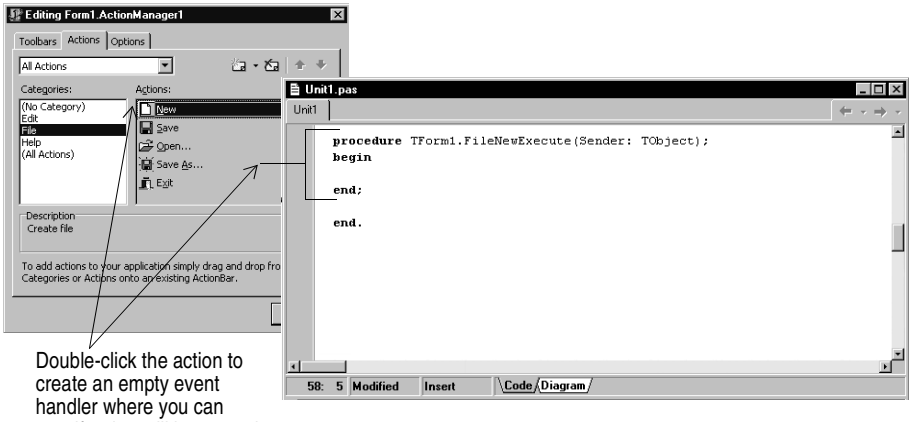
Tip *F12* is a toggle that takes you back and forth from the form to the associated code.

4. Double-click the *ActionManager* to open it.

5 In the Action Manager editor, select the File category and then double-click the New action.

Tip You can also double-click the File | FileNew action in the Object TreeView.

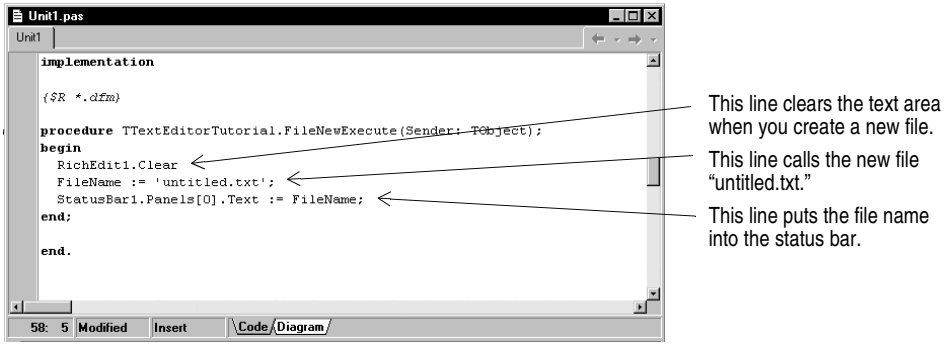
The Code editor opens with the cursor inside the event handler.



6 Right where the cursor is positioned in the Code editor (between `begin` and `end`), type the following lines:

```
RichEdit1.Clear;
FileName := 'untitled.txt';
StatusBar1.Panels[0].Text := FileName;
```

Your event handler should look like this when you're done:



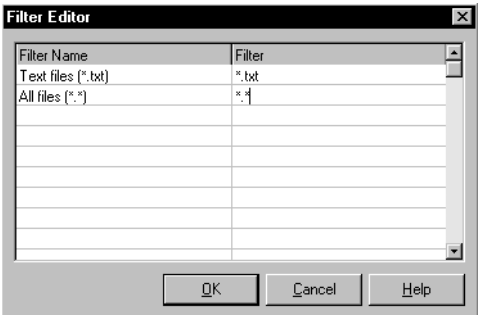
Save your work and that's it for the File | New command.

Note You can resize the code portion of the window to reduce horizontal scrolling.

Creating an event handler for the Open command

To open a file in the text editor, you want a standard Windows Open dialog box to appear. You've already added a standard File | Open command to the Action Manager editor, which automatically includes the dialog box. However, you still need to customize the event handler for the command.

1. Press *F12* to locate the main form (or select View | Forms and choose Form1).
2. Double-click the Action Manager editor to open it. Select the File | Open action.
3. In the Object Inspector, click the plus sign to the left of the *Dialog* property to expand its properties. Delphi names the dialog box *FileOpen1.OpenDialog* by default. When *OpenDialog1*'s *Execute* method is called, it invokes the standard dialog box for opening files.
4. Set the following properties of *FileOpen1.Dialog*:
   - Set *DefaultExt* to `txt`.
   - Double-click the text area next to *Filter* to display the Filter editor. In the first row under the Filter Name column, type `Text files (*.txt)`. In the Filter column, type `*.txt`. In the second row under the Filter Name column, type `All files (*.*)` and in the Filter column, type `*.*`. Click OK.

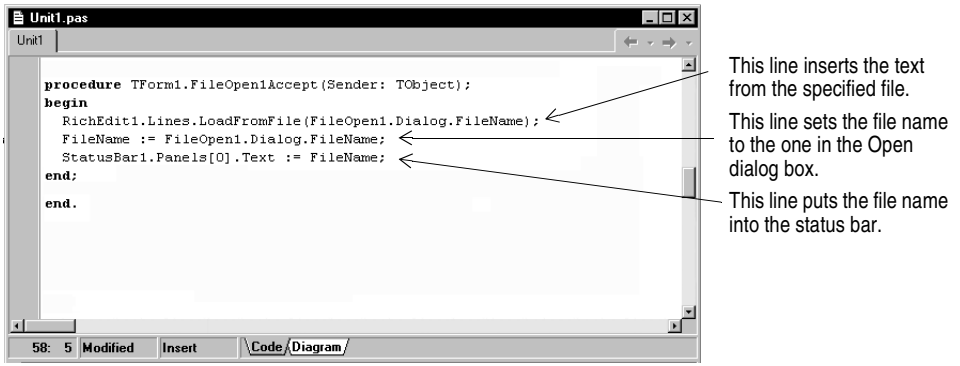


Use the Filter editor to define filters for the *FileOpen1.Dialog* and *FileSaveAs1.Dialog* actions.

   - After *Title*, type `Open file`. These words will appear at the top of the Open dialog box.
5. Click the Events tab. Double-click the *OnAccept* event so that `FileOpen1Accept` appears.
6. The Code editor opens with the cursor inside the event handler.
7. Right where the cursor is positioned in the Code editor (between **`begin`** and **`end`**), type the following lines:

```
RichEdit1.Lines.LoadFromFile(FileOpen1.Dialog.FileName);
FileName := FileOpen1.Dialog.FileName;
StatusBar1.Panels[0].Text := FileName;
```

Your FileOpen event handler should look like this when you're done:



That's it for the File | Open command and the Open dialog box.

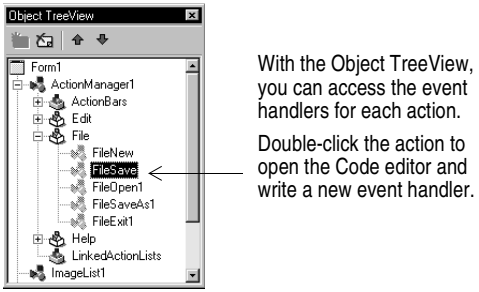
Creating an event handler for the Save command

To create an event handler for the Save command:

1. Press F12 to display the form. Double-click the *ActionManager* component to open it.
2. Double-click the File | Save action.

   The Code editor opens with the cursor inside the event handler.

Tip You can also double-click the File | FileSave action in the Object TreeView.

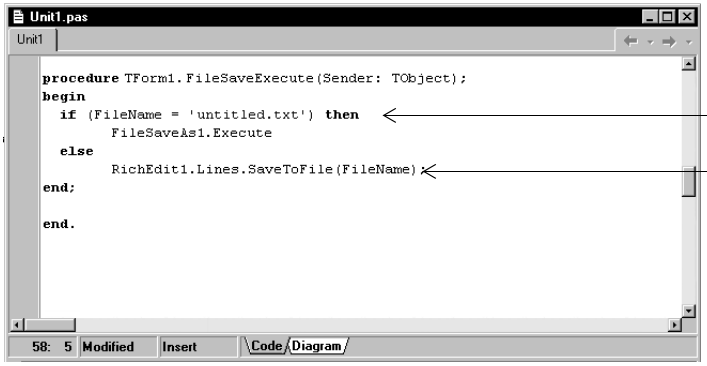


3. Right where the cursor is positioned in the Code editor (between `begin` and `end`), type the following lines:

```
if (FileName = 'untitled.txt') then
   FileSaveAs1.Execute
else
   RichEdit1.Lines.SaveToFile(FileName);
```

This code tells the text editor to display the SaveAs dialog box if the file isn't named yet so the user can assign a name to it. Otherwise, save the file using its current name. The SaveAs dialog box is defined in the event handler for the Save As command on page 4-20. *FileSaveAs1BeforeExecute* is the automatically generated name for the Save As command.

Your event handler should look like this when you're done:



That's it for the File | Save command.

Creating an event handler for the Save As command

When *SaveDialog*'s *Execute* method is called, it invokes the standard Windows Save As dialog box for saving files. To create an event handler for the Save As command:

1. Press F12 to display the form. Double-click the *ActionManager* component to open it.
2. Select the File | SaveAs action.
3. In the Object Inspector, click the Properties tab, and set the following properties for the *FileSaveAs1* dialog box. Delphi names it *FileSaveAs1.Dialog* by default.
4. Click the plus sign to the left of the *Dialog* property and set the following properties:
   - Set *DefaultExt* to `txt`.
   - Double-click the text area next to *Filter* to display the Filter editor. In the Filter editor, specify filters for file types as in the Open dialog box. In the first row under the Filter Name column, type `Text files (*.txt)`. In the Filter column, type `*.txt`. In the second row under the Filter Name column, type `All files (*.*)` and in the Filter column, type `*.*`. Click OK.
   - Set *Title* to `Save as`.
5. In the Object Inspector, click the Events tab. Double-click the text area next to *BeforeExecute* so that `FileSaveAs1BeforeExecute` appears. The Code editor opens with the cursor inside the Code editor.
6. Right where the cursor is positioned in the Code editor, type the following line:

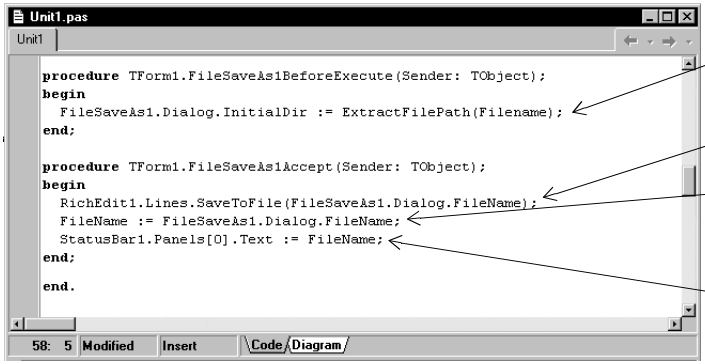
   ```
   FileSaveAs1.Dialog.InitialDir := ExtractFilePath(Filename);
   ```

7. In the Object Inspector, the Events tab should still be displayed. Double-click the text area next to the *OnAccept* event so that `FileSaveAs1Accept` appears.

8 The Code editor opens with the cursor inside the event handler. Type the following lines.

```
RichEdit1.Lines.SaveToFile(FileSaveAs1.Dialog.FileName);
FileName := FileSaveAs1.Dialog.FileName;
StatusBar1.Panels[0].Text := FileName;
```

Your FileSaveAs event handler should look like this when you're done:



This line sets the default directory to the last one accessed.

This line saves the text to the specified file.

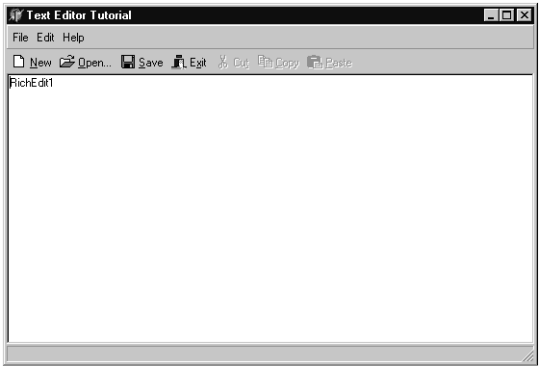
This sets the main form's FileName to the name specified in the SaveAs dialog box.

This puts the file name in the status bar.

That's it for the File | SaveAs command.

9 Choose File | Save All to save your project.

10 To see what it looks like so far, run the application by pressing *F9*.



Your application has full functionality. The images appear next to commands with which you associated an image index.

Notice that the nonvisual components aren't there.The menus, toolbar, text area, and status bar all appear on the form.

Most of the buttons and toolbar buttons work but you're not finished yet.

If you receive any error messages at the bottom of the Code editor, click them to go right to the place in the code where the error occurred. Make sure you've followed the steps as described in the tutorial.

11 To return to design mode, click **X** in the upper right corner.

Creating a Help file

It's a good idea to create a Help file that explains how to use your application. Delphi provides Microsoft Help Workshop in the C:\Project Files\Borland | Delphi6\Help\Tools directory which includes information on designing and compiling a Windows Help file. In the sample text editor application, users can choose Help | Contents or Help | Index to access a Help file with either the contents or index displayed.

Earlier, you created HelpContents and HelpIndex actions in the action manager for displaying the Contents tab or Index tab of a compiled Help file. You need to assign constant values to the Help parameters and create event handlers that display what you want.

To use the Help commands, you'll have to create and compile a Windows Help file. Creating Help files is beyond the scope of this tutorial. However, you can download a sample rtf file (TextEditor.rtf), Help file (TextEditor.hlp) and contents file (TextEditor.cnt):

1. From your C:\Project Files\Borland\Delphi6\Help directory, open D6X1.zip.
2. Extract and save the .hlp and .cnt files in your Text Editor directory; by default, C:\Project Files\Borland\Delphi6\Projects\TextEditor.

Note You can use any HLP or CNT file (such as one of the Delphi Help files and its associated CNT file) in your project. You will have to rename them as TextEditor.hlp and TextEditor.cnt for the application to find them.

Creating an event handler for the Help Contents command

To create an event handler for the Help Contents command:

1. Double-click the *ActionManager* component to open it.
2. On the Action Manager editor, select the Help category, then double-click the HelpContents action.

   The Code editor opens with the cursor inside the event handler.
3. Right before where the cursor is positioned in the text editor, that is, right before **begin**, type the following lines:

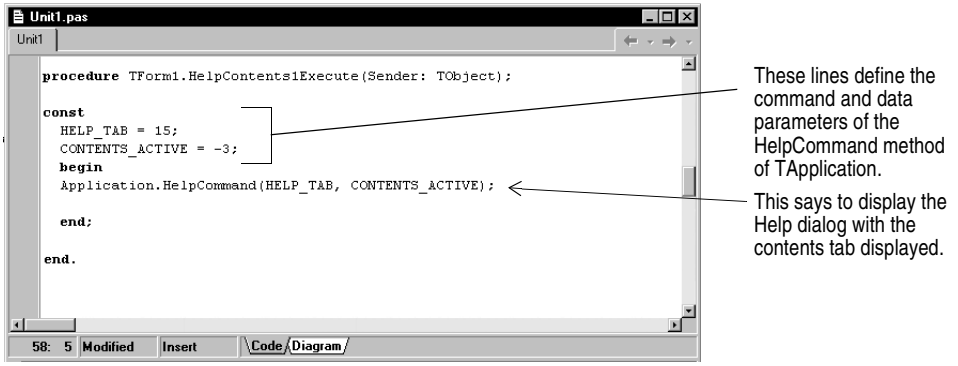
```
const
 HELP_TAB = 15;
 CONTENTS_ACTIVE = -3;
```

   Right after **begin**, type:

```
Application.HelpCommand(HELP_TAB, CONTENTS_ACTIVE);
```

   This code assigns constant values to the HelpCommand parameters. Setting HELP_TAB to 15 displays the Help dialog and setting CONTENTS_ACTIVE to -3 displays the Contents tab.

Your event handler should look like this when you're done:



Note To get Help on the HelpCommand event, put the cursor next to HelpCommand in the editor and press *F1*.

That's it for the Help | Contents command.

Creating an event handler for the Help Index command

To create an event handler for the Help Index command:

1. The Action Manager editor should still be displayed. If it's not, double-click the *ActionManager* component on the form.
2. In the Action Manager editor, select the Help category and then double-click the HelpIndex action.

   The Code editor opens with the cursor inside the event handler.
3. Right before where the cursor is positioned in the text editor, that is right before begin, type the following lines:

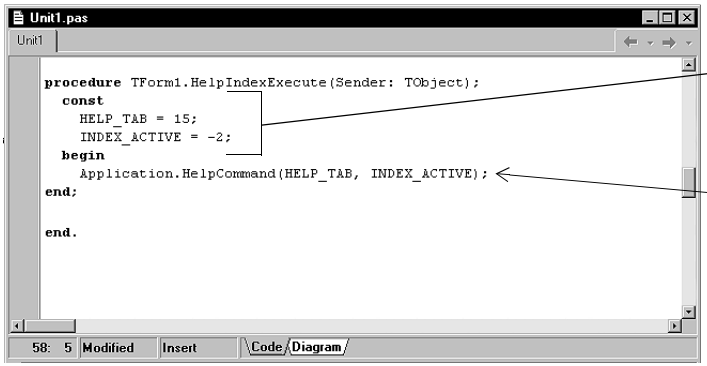
```
  const
HELP_TAB = 15;
INDEX_ACTIVE = -2;
```

   Right after **begin**, type

```
Application.HelpCommand(HELP_TAB, INDEX_ACTIVE);
```

   This code assigns constant values to the HelpCommand parameters. Setting HELP_TAB to 15 again displays the Help dialog box and setting INDEX_ACTIVE to -2 displays the Index tab.

Your event handler should look like this when you're done:



That's it for the Help | Index command.

Creating an About box

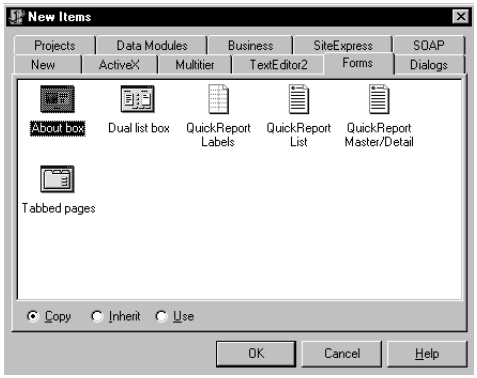
Many applications include an About box which displays information on the product such as the name, version, logos, and may include other legal information including copyright information.

You've already set up a Help About command on the action manager.

To add an About box:

1 Choose File | New | Other to display the New Items dialog box and click the Forms tab.

2 On the Forms tab, double-click About Box.



A new form is created that simplifies creation of an About box.

3 Select the form itself (click the grid portion) and in the Object Inspector, change its *Caption* property to `About Text Editor`.

4 In the Object Inspector, click the Properties tab and change the *Caption* properties for the following *TLabel* items:

- Change `Product Name` to `Text Editor`.
- Add `1.0` after `Version`.
- Add the year after `Copyright`.

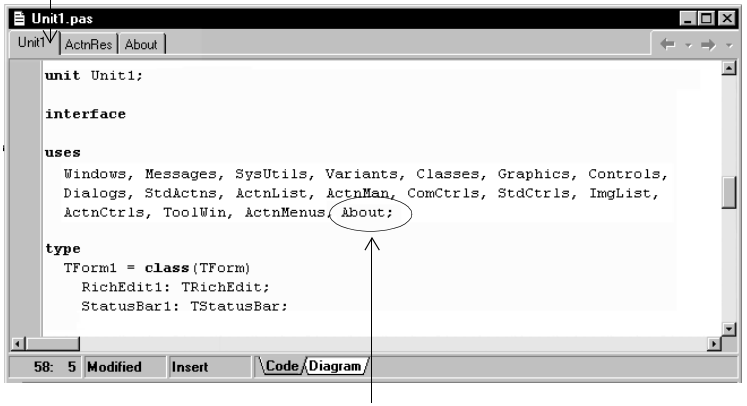
The Object Repository contains a standard About box that you can modify as you like to describe your application.

5 Save the About box form by choosing File | SaveAs and saving it as About.pas.

6 In the Delphi Code editor, you should have three unit files displayed: Unit1, ActnRes, and About. Click the Unit1 tab to display Unit1.pas. You don't need the ActnRes unit but you can leave it there.

7 Click the Unit1 tab, and add the new About unit by typing the word About to the list of included units in the **uses** clause.

Click the tab to display a file associated with a unit. If you open other files while working on a project, additional tabs appear on the Code editor.



When you create a new form for your application, you need to add it to the **uses** clause of the main form. Here you're adding the About box.

8 Press *F12* to return to design mode. Double-click the *ActionManager* component to open it.

9 Double-click the HelpAbout action to create an event handler. Right where the cursor is positioned in the Code editor, type the following line:

```
AboutBox.ShowModal;
```

This code opens the About box when the user clicks Help | About. `ShowModal` opens the form in a modal state, a runtime state when the user can't do anything until the form is closed.

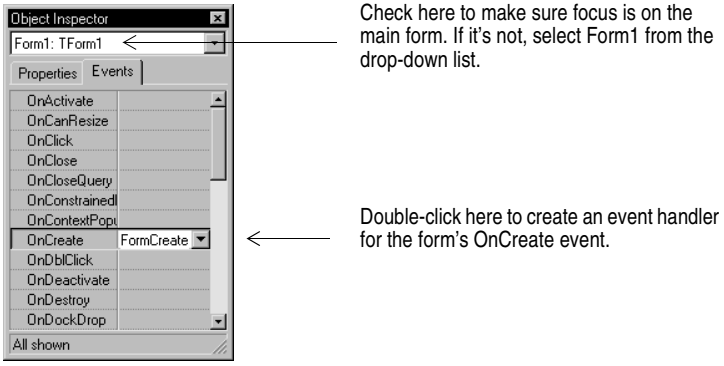
Completing your application

The application is almost complete. However, you still have to specify some items on the main form. To complete the application:

1 Press *F12* to locate the main form.

2 Check that focus is on the form itself, not any of its components. The list box at the top of the Object Inspector should say `Form1: TForm1`. (If it doesn't, select `Form1` from the drop-down list.)

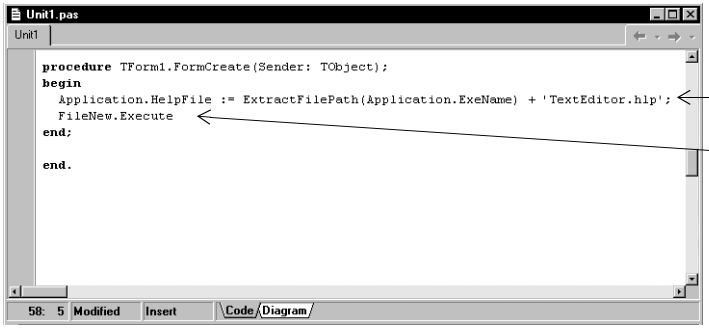
3 Click the Events tab, and next to the *OnCreate* event, choose `FormCreate` from the drop-down list to create an event handler that describes what happens when the form is created (that is, when you open the application).



4 Right where the cursor is positioned in the Code editor, type the following lines:

```
Application.HelpFile := ExtractFilePath(Application.ExeName) + 'TextEditor.hlp';
FileNew.Execute
```

This code initializes the application by application by associating a Help file, setting the value of *FileName* to `untitled.txt`, putting the file name into the status bar, and clearing out the text editing area.



5 Choose File | SaveAll to save your changes.

6 Press *F9* to run the application.

Congratulations! You're done.

Chapter

5

Customizing the desktop

This chapter explains some of the ways you can customize the tools in Delphi's IDE.

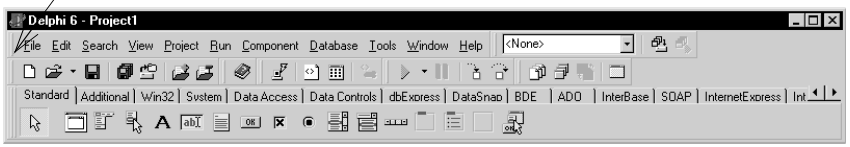
Organizing your work area

The IDE provides many tools to support development, so you'll want to reorganize your work area for maximum convenience, including rearranging your menus and toolbars, combining tool windows, and saving a new way your desktop looks.

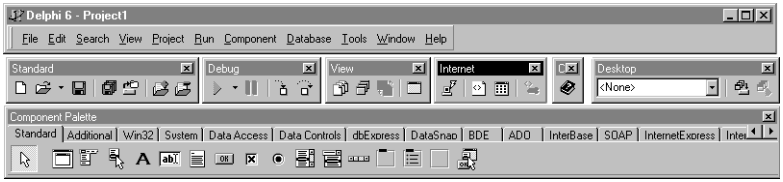
Arranging menus and toolbars

In the main window, you can reorganize the menu, toolbars, and Component palette by clicking the grabber on the left-hand side of each one and dragging it to another location.

You can move menus and toolbars within the main window. Drag the grabber (the double bar on the left) of an individual toolbar to move it.

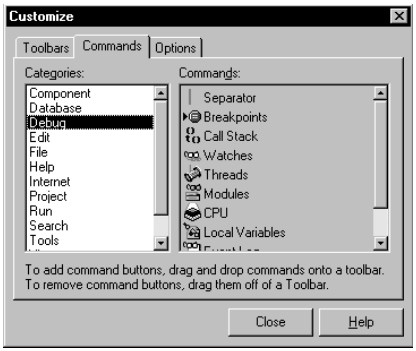


You can separate parts from the main window and place them elsewhere on the screen or remove them from the desktop altogether. This is useful if you have a dual monitor setup.



Main window organized differently.

You can add or delete tools from the toolbars by choosing View | Toolbars | Customize. Click the Commands page, select a category, select a command, and drag it to the toolbar where you want to place it.



On the Commands page, select any command and drag it onto any toolbar.

On the Options page, click Show tooltips to make sure the hints for components and toolbar icons appear.

For more information...

See “toolbars, customizing” in the online Help index.

Docking tool windows

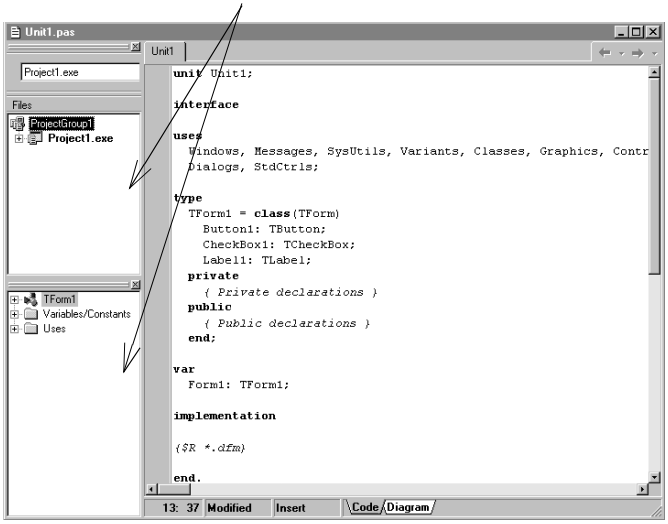
You can open and close individual tool windows and arrange them on the desktop as you wish. Many windows can also be *docked* to one another for easy management. Docking—which means attaching windows to each other so that they move together—helps you use screen space efficiently while maintaining fast access to tools.

From the View menu, you can bring up any tool window and then dock it directly to another. For example, when you first open Delphi in its default configuration, the

Code Explorer is docked to the left of the Code editor. You can add the Project Manager to the first two to create three docked windows.

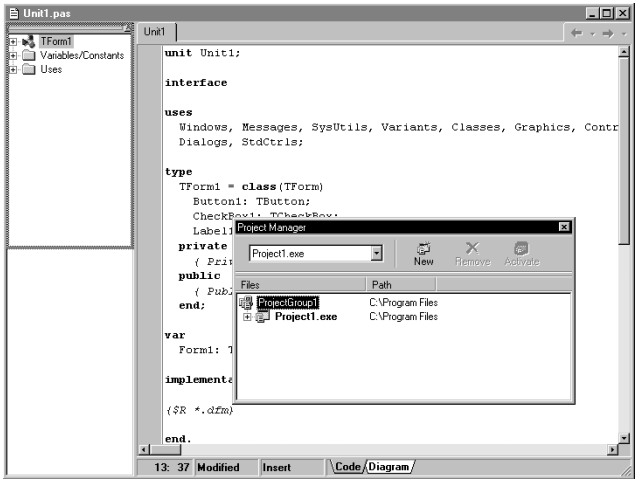
Here the Project Manager and Code Explorer are docked to the Code editor.

You can combine, or "dock" windows with either grabbers, as on the right, or tabs, as on page 5-4.

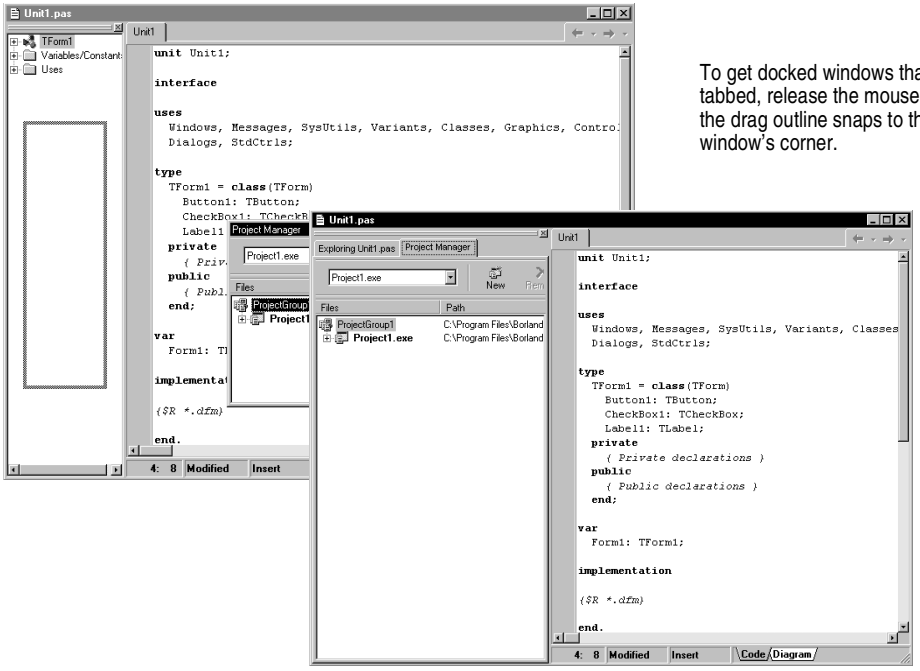


To dock a window, click its title bar and drag it over the other window. When the drag outline narrows into a rectangle and it snaps into a corner, release the mouse. The two windows snap together.

To get docked windows with grabbers, release the mouse when the drag outline snaps to the window's corner.



You can also dock tools to form tabbed windows.



To get docked windows that are tabbed, release the mouse *before* the drag outline snaps to the other window's corner.

To undock a window, double-click its grabber or tab.

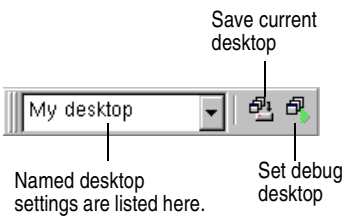
To turn off automatic docking, either press the *Ctrl* key while moving windows around the screen, or choose Tools | Environment Options, click the Designer page, and uncheck the Auto drag docking check box.

For more information...

See "docking" in the online Help index.

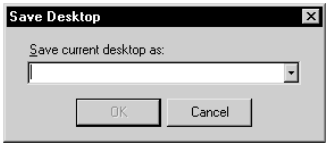
Saving desktop layouts

You can customize and save your desktop layout. The Desktops toolbar in the IDE includes a pick list of the available desktop layouts and two icons to make it easy to customize the desktop.



Arrange the desktop as you want, including displaying, sizing, and docking particular windows.

On the Desktops toolbar, click the Save current desktop icon or choose View | Desktops | Save Desktop, and enter a name for your new layout.



Enter a name for the desktop layout you want to save and click OK.

For more information...

See "desktop layout" in the online Help index.

Customizing the Component palette

In its default configuration, the Component palette displays many useful VCL or CLX objects organized functionally onto tabbed pages. You can customize the Component palette by:

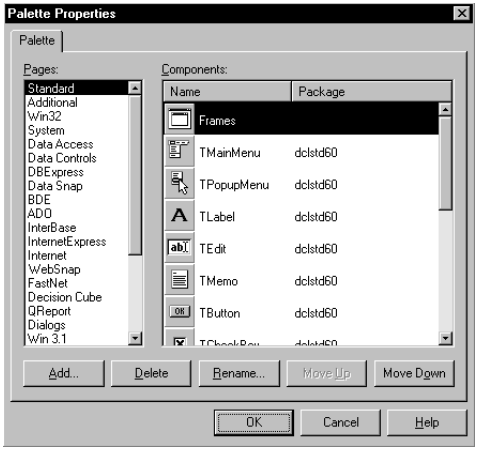
- Hiding or rearranging components.
- Adding, removing, rearranging, or renaming pages.
- Creating component templates and adding them to the palette.
- Installing new components.

Arranging the Component palette

To add, delete, rearrange, or rename pages, or to hide or rearrange components, use the Palette Properties dialog box. You can open this dialog box in several ways:

- Choose Component | Configure Palette.
- Choose Tools | Environment Options and click the Palette tab.

- Right-click the Component palette and choose Properties.



You can rearrange the palette and add new pages.

For more information...

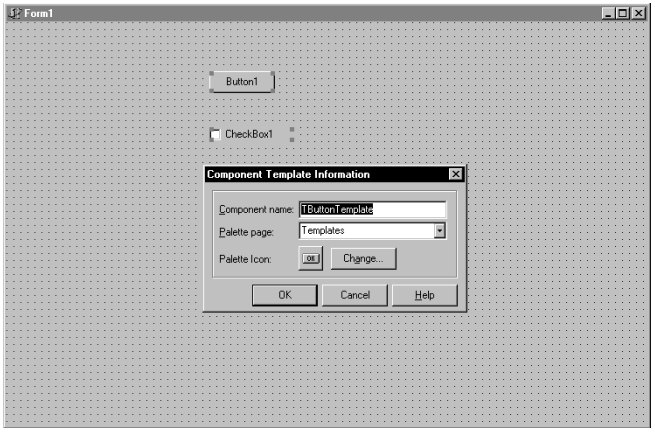
Click the Help button in the Palette Properties dialog box.

Creating component templates

Component templates are groups of components that you add to a form in a single operation. Templates allow you to configure components on one form, then save their arrangement, default properties, and event handlers on the Component palette to reuse on other forms.

To create a component template, simply arrange one or more components on a form and set their properties in the Object Inspector, and select all of the components by dragging the mouse over them. Then choose Component | Create Component Template. When the Component Template Information dialog box opens, select a name for the template, the palette page on which you want it to appear, and an icon to represent the template on the palette.

After placing a 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...

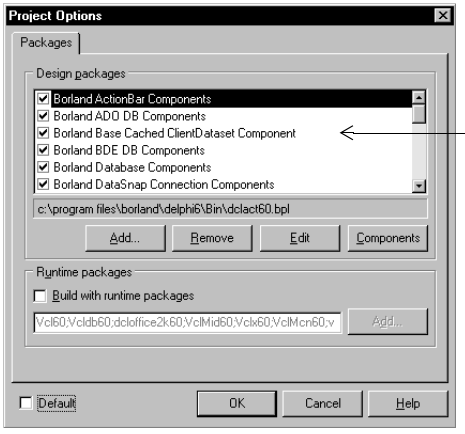
See "templates, component" in the online Help index.

Installing component packages

Whether you write custom components or obtain them from a vendor, the components must be compiled into a *package* before you can install them on the Component palette.

A package is a special DLL containing code that can be shared among Delphi applications, the IDE, or both. *Runtime packages* provide functionality when a user runs an application. *Design-time packages* are used to install components in the IDE. Delphi packages have a .bpl extension.

If a third-party vendor's components are already compiled into a package, either follow the vendor's instructions or choose Component | Install Packages.



These components come preinstalled in Delphi. When you install new components from third-party vendors, their package appears in this list.

Click Components to see what components the package contains.

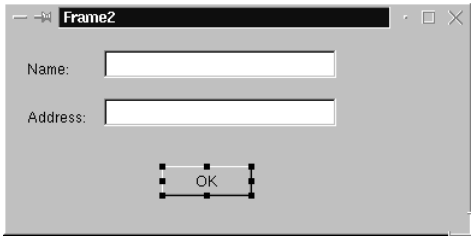
For more information...

See "installing components" and "packages" in the online Help index.

Using frames

A frame (*TFrame*), like a form, is a container for components that you want to reuse. A frame is more like a customized component than a form. Frames can be saved on the Component palette for easy reuse and they can be nested within forms, other frames, or other container objects. After a frame is created and saved, it continues to function as a unit and to inherit changes from the components (including other frames) it contains. When a frame is embedded in another frame or form, it continues to inherit changes made to the frame from which it derives.

To open a new frame, choose File | New | Frame.



You can add whatever visual or nonvisual components you need to the frame. A new unit is automatically added to the Code editor.

For more information...

See "frames" and "TFrame" in the Help index.

Adding ActiveX controls

You can add ActiveX controls to the Component palette and use them in your Delphi projects. Choose Component | Import ActiveX Control to open the Import ActiveX dialog box. From here you can register new ActiveX controls or select an already registered control for installation in the IDE. When you install an ActiveX control, Delphi creates and compiles a "wrapper" unit file for it.

For more information...

Choose Component | Import ActiveX Control and click the Help button.

Setting project options

If you need to manage project directories and to specify form, application, compiler, and linker options for your project, choose Project | Options. When you make changes in the Project Options dialog box, your changes affect only the current project; but you can also save your selections as the default settings for new projects.

Setting default project options

To save your selections as the default settings for all new projects, in the lower-left corner of the Project Options dialog box, check Default. Checking Default writes the current settings from the dialog box to the options file Defproj.dof, located in the Delphi6\Bin directory. To restore Delphi's original default settings, delete or rename the Defproj.dof file.

For more information...

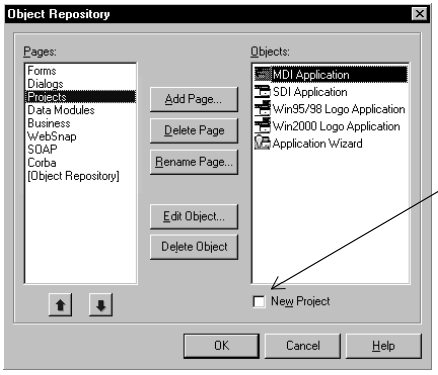
See "Project Options dialog box" in the online Help index.

Specifying project and form templates as the default

When you choose File | New | Application, Delphi creates a standard new application with an empty form, unless you specify a project *template* as your *default* project. You can save your own project as a template in the Object Repository on the Projects page by choosing Project | Add to Repository (see "Adding templates to the Object Repository" on page 5-10). Or you can choose from one of Delphi's existing project templates from the Object Repository (see "The Object Repository" on page 2-5).

To specify a project template as the default, choose Tools | Repository. In the Object Repository dialog box, under Pages, select Projects. If you've saved a project as a

template on the Projects page, it appears in the Objects list. Select the template name, check New Project, and click OK.



The Object Repository's pages contain project templates only, form templates only, or a combination of both.

To set a project template as the default, select an item in the Objects list and check New Project.

To set a form template as the default, select an item in the Objects list and check New Form or Main Form.

Once you've specified a project template as the default, Delphi opens it automatically whenever you choose File | New | Application.

In the same way that you specify a default project, you can specify a *default new form* and a *default main form* from a list of existing form templates in the Object Repository. The default new form is the form created when you choose File | New | Form to add an additional form to an open project. The default main form is the form created when you open a new application. If you haven't specified a default form, Delphi uses a blank form.

You can override your default project or form temporarily by choosing File | New | Other and selecting a different template from the New Items dialog box.

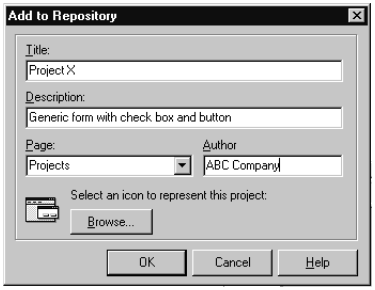
For more information...

See "templates, adding to Object Repository," "projects, specifying default," and "forms, specifying default" in the online Help index.

Adding templates to the Object Repository

You can add your own objects to the Object Repository as *templates* to reuse and share with other developers over a network. Reusing objects lets you build families of applications with common user interfaces and functionality that reduces development time and improves quality.

For example, to add a project to the Repository as a template, first save the project and choose Project | Add To Repository. Complete the Add to Repository dialog box.



Enter a title, description, and author. In the Page list box, choose Projects so that your project will appear on the Repository's Projects tabbed page.

The next time you open the New Items dialog box, your project template will appear on the Projects page (or the page to which you had saved it). To make your template the default every time you open Delphi, see "Specifying project and form templates as the default" on page 5-9.

For more information...

See "templates, adding to Object Repository" in the online Help index.

Setting tool preferences

You can control many aspects of the appearance and behavior of the IDE, such as the Form Designer, Object Inspector, and Code Explorer. These settings affect not just the current project, but projects that you open and compile later. To change global IDE settings for all projects, choose Tools | Environment Options.

For more information...

See "Environment Options dialog box" in the online Help index, or click the Help button on any page in the Environment Options dialog box.

Customizing the Form Designer

The Designer page of the Tools | Environment Options dialog box has settings that affect the Form Designer. For example, you can enable or disable the "snap to grid" feature, which aligns components with the nearest grid line; you can also display or hide the names, or *captions*, of nonvisual components you place on your form.

For more information...

In the Environment Options dialog box, click the Designer page and click the Help button.

Customizing the Code Editor

One tool you may want to customize right away is the Code editor. Several pages in the Tools | Editor Options dialog box have settings for how you edit your code. For example, you can choose keystroke mappings, fonts, margin widths, colors, syntax highlighting, tabs, and indentation styles.

You can also configure the Code Insight tools that you can use within the editor on the Code Insight page of Editor Options. To learn about these tools, see "Code Insight" on page 2-7.

For more information...

In the Editor Options dialog box, click the Help button on the General, Display, Key Mappings, Color, and Code Insight pages.

Customizing the Code Explorer

When you start Delphi, the Code Explorer (described in "The Code Explorer" on page 2-11) opens automatically. If you don't want Code Explorer to open automatically, choose Tools | Environment Options, click the Explorer tab, and uncheck Automatically show Explorer.

You can change the way the Code Explorer's contents are grouped within the Code Explorer by right-clicking in the Code Explorer, choosing Properties, and, under Explorer categories, checking and unchecking the check boxes. If a category is checked, elements in that category are grouped under a single node. If a category is unchecked, each element in that category is displayed independently on the diagram's trunk. For example, if you uncheck the Published category, the Published folder disappears but not the items in it.

In the Code Explorer, you can sort all source elements alphabetically or in the order in which they are declared in the source file.

To display the folder for each type of source element in the Code Explorer, check an Explorer category.

For more information...

See "Code Explorer, Environment options" in the online Help index.

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X