Quick Start

Borland®
Delphi™ 5
for Windows 98, Windows 95, & Windows NT

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

Refer to the file DEPLOY.TXT located in the root directory of your Delphi 5 product for a complete list of files that you can distribute in accordance with the Delphi 5 License Statement and Limited Warranty.

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

Printed in the U.S.A.

HDE1350WW21000 2E1R799
9900010203-9 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.

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 95, Windows 98, and Windows NT with a minimum of manual coding. Delphi provides all the tools you need to develop, test, debug, and deploy applications, including a large library of reusable components, a suite of design tools, application and form templates, and programming wizards. These tools simplify prototyping and shorten development time.

Where to find information

Information on Delphi is available in a variety of forms:

- Online Help
- Printed documentation
- Inprise developer support services
- Inprise and borland.com Web sites

For information about new features in this release, refer to What's New in Delphi? in the online Help and to the 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 (VCL). It includes the core Help files listed in Table 1.1.

Table 1.1 Online Help files

Help file	Contents	Audience
What's New in Delphi? (Del5new.hlp)	Introduces new features and enhancements to Delphi for the current release and includes links to detailed information.	Developers who upgraded to this release
Using Delphi (Delphi5.hlp)	Introduces the Delphi development environment and explains how to work with forms projects, and packages. Discusses basic concepts of component-based object-oriented programming.	New Delphi developers, people with questions about the IDE
Visual Component Library Reference (Del5vcl.hlp)	Presents detailed reference on VCL classes, global routines, types, and variables. Entries show the unit where each class is declared; its position in the hierarchy; a list of available properties, methods, and events; and code examples.	All Delphi developers
Programming with Delphi (Del5prog.hlp)	Provides details about using the VCL components and illustrates common programming tasks such as handling exceptions, creating toolbars and drag-and-drop controls, and using graphics.	All Delphi developers
Developing Database Applications (Del5dbd.hlp)	Explains design of single- and multi-tiered database applications, including database architecture, datasets, fields, tables, queries, and decision support.	Database developers
Developing Distributed Applications (Del5dap.hlp)	Explains how to create distributed applications. Includes information on CORBA, DCOM, MTS, HTTP, and sockets.	Developers writing client/server applications
Creating Custom Components (Del5cw.hlp)	Provides information on writing custom Delphi components. Explains how to design, build, test, and install a component.	Developers writing Delphi components
Developing COM-based Applications (Del5com.hlp)	Explains how to build distributed applications using COM. Topics include COM objects, MTS components, Automation servers and controllers, ActiveX controls, and type libraries. Explains how to modify generated type libraries using Delphi's Type Library Editor	Developers writing client/server applications
Object Pascal Reference (Del5op.hlp)	Provides a formal definition of the Object Pascal language and includes topics on file I/O, string manipulation, program control, data types, and language extensions.	Developers who need Object Pascal language details
Customizing Help (OpenHelp.hlp)	Explains how to configure the Delphi Help system. The OpenHelp utility lets you add or remove any Windows Help (.HLP) file.	Developers wanting to customize the Delphi Help system

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

- Integrated Translation Environment (ITE) Help
- Borland Database Engine (BDE) Help
- BDE Administrator Help
- Database Explorer Help
- Local SQL, SQL Builder, and SQL Monitor Help
- Package Collection Editor Help
- Help Author's Guide (Help Workshop)
- QuickReport Help
- TeeChart Help
- InterBase and InterBase Express Help
- CORBA Component Library Reference Help
- Help for miscellaneous components (FastNet Time, DayTime, Echo, Finger, HTTP, NNTP, POP3, Powersock, SMTP, UDP, URL Encode/Decode, UUprocessor, Stream and Msg components)

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

For information on customizing your Help system, see "Customizing the Help system" on page 4-9.

Printed documentation

This *Quick Start* is an introduction to Delphi. To order additional printed documentation, refer to the Web site at shop.borland.com.

Developer support services

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

Additional Delphi Technical Information documents and answers to Frequently Asked Questions (FAQs) are also available at this Web site.

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.

For information about year 2000 issues and our products, see the following URL: http://www.inprise.com/devsupport/y2000/.

Typographic conventions

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

Table 1.2 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

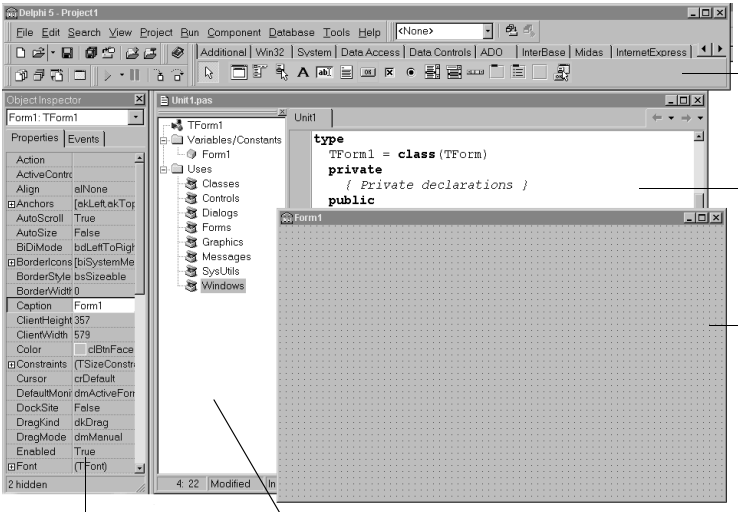
A tour of the environment

Starting Delphi

You can start Delphi in several ways:

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

Right away, you'll see some of the major tools in Delphi's integrated development environment (IDE).



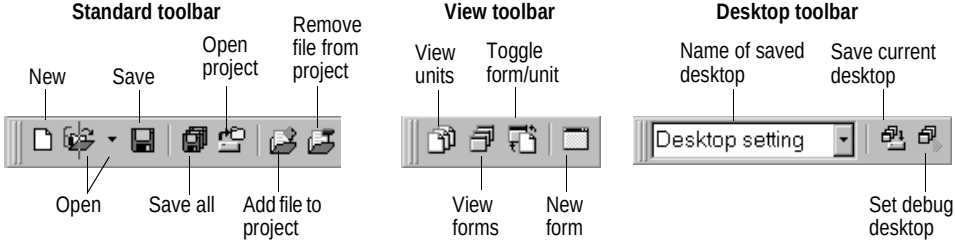
Delphi's development model is based on *two-way* tools. This means that you can move back and forth between visual design tools and text-based editing. For example, after using the Form Designer to arrange buttons and other elements in a graphical interface, you can immediately view the .DFM file that 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.

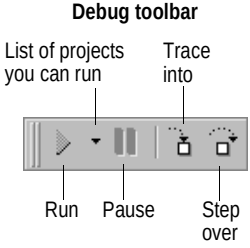
From the IDE, all your programming tools are within easy reach. You can manage projects, design graphical interfaces, write code, search databases, compile, test, debug, and browse through class libraries without leaving the IDE.

To learn about organizing and configuring the IDE, see Chapter 4, "Customizing the environment."

Using toolbars, menus, and keyboard shortcuts

Delphi's toolbars, located in the main window, provide quick access to frequently used operations and commands. All toolbar operations are duplicated in the drop-down menus.





To find out what a button does, point to it for a moment and a hint is displayed. The button's keyboard shortcut, if it has one, is displayed as well.

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*.

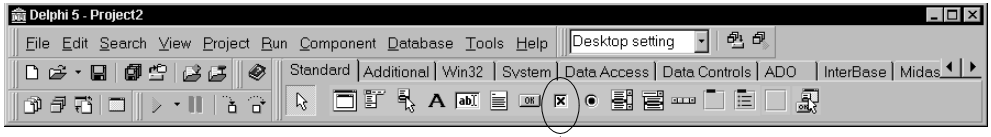
The toolbar is also customizable. You can add commands you want to it or move the parts of the toolbar to different locations. For more information, see "Arranging menus and toolbars" on page 4-4.

You can name and save desktop arrangements using the Desktop toolbar.

Placing components on a form

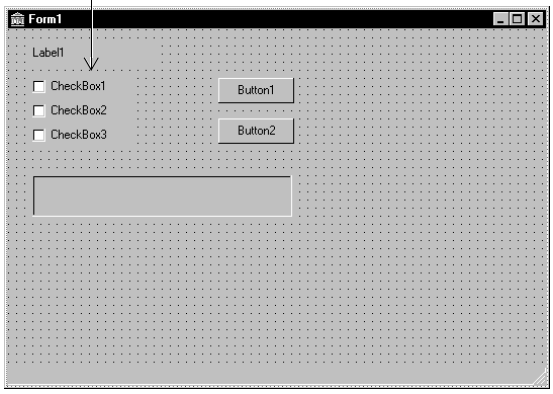
To build an application interface, you place components on a form, set their properties, and code their event handlers.

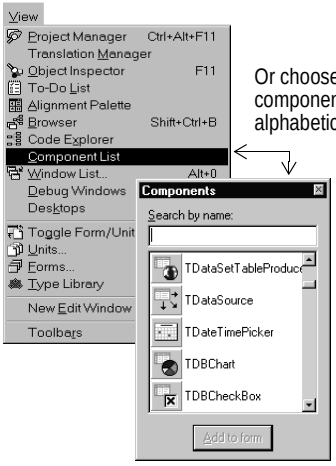
Many components are provided on the Component palette, grouped by function.



Click a component on the Component palette.

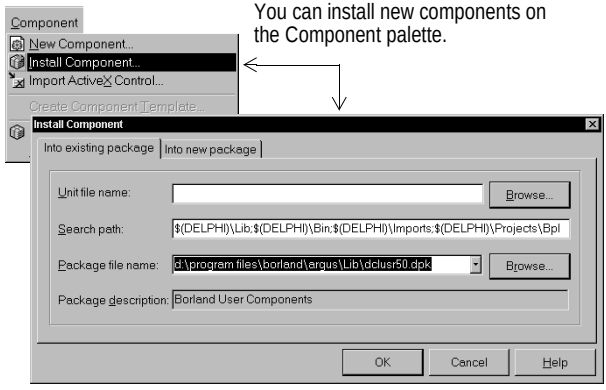
Then click where you want to place it on the form.



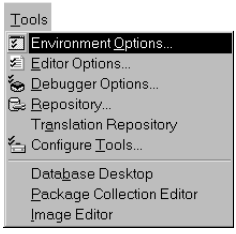


Or choose a component from an alphabetical list.

You can install new components on the Component palette.

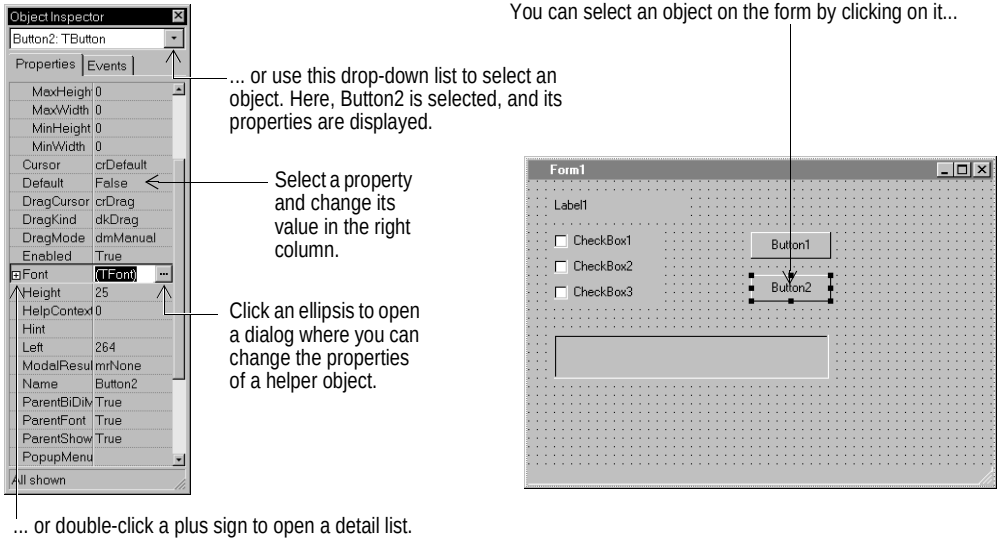


You can also rearrange the palette and add new pages. Choose Environment Options, then the Palette page.

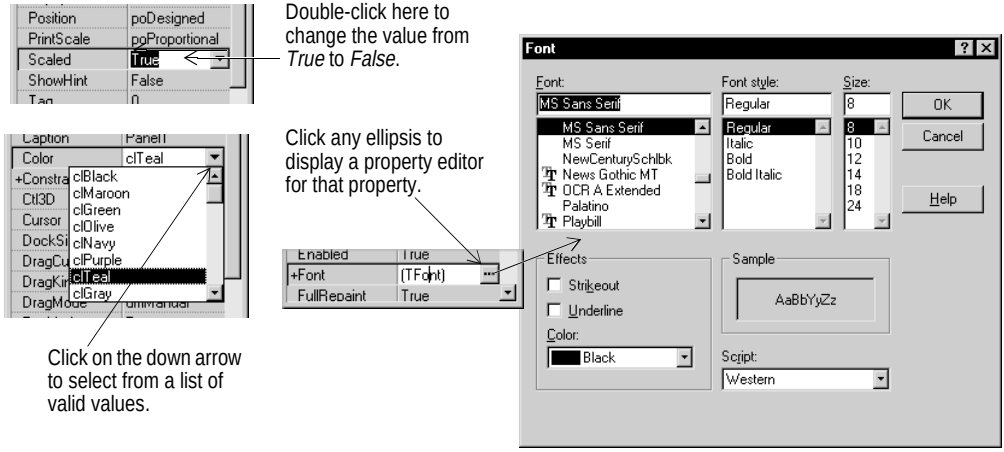


Changing component appearance and behavior

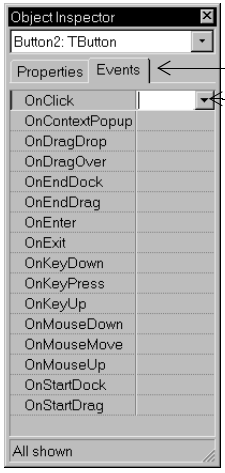
You can change the way a component appears and behaves in your application by using the Object Inspector. When a component is selected on a form, its properties and events are displayed in the Object Inspector.



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.



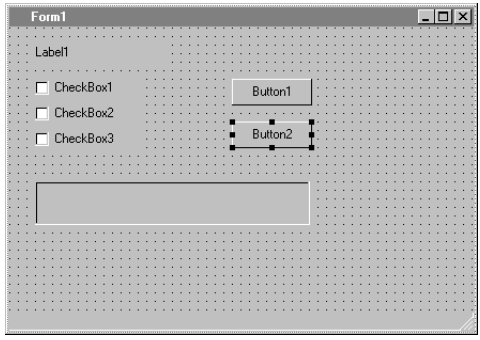
Working with events



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

Select an existing event handler from the drop-down list.

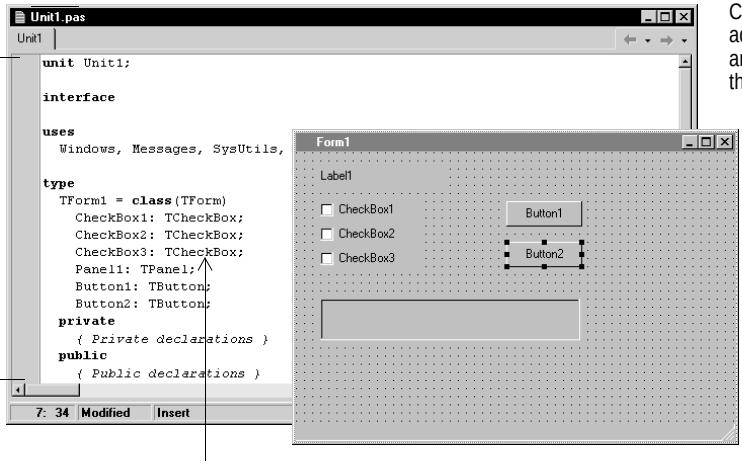
Or double-click in the value column, and Delphi generates skeleton code for a new event handler.



Viewing and editing code

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

You can also add code to your source files directly using the built-in Code editor. The Code editor is a full-featured ASCII editor.



Components added to the form are reflected in the code.

Generated code.

Click on any language keyword or VCL element, and press *F1* to get Help.

Choose Tools | Editor Options to customize your editing environment. You can set options such as tabbing, key mapping, color, and automatic features.

Viewing form files

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 them forms in form files. Form files (extension .DFM) describe each component in your form, including the values of all persistent properties.

To view a form (.DFM) file in the editor, right-click on the form and select View as Text. Form files can be edited. To return to the pictorial view of your form, right-click and choose View as Form.

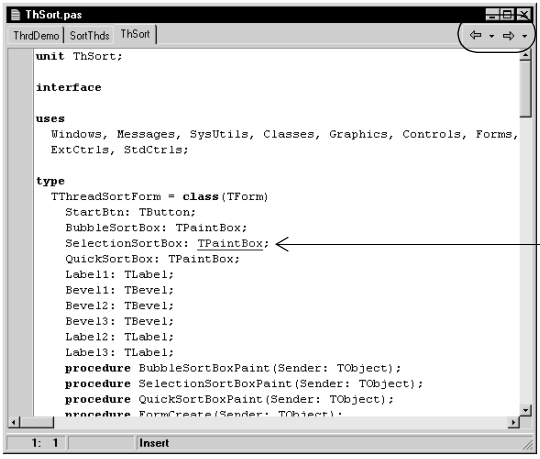
You can save form files in either text (the default) or binary format. The Environment Options dialog lets you indicate which format to use for newly created forms.

For more information...

Search for "form files" in the Help index.

Browsing with the editor

The Code editor has Forward and Back buttons like the ones you've seen on Web browsers. You can use them to navigate through source code. Click the left arrow to return to the last place you were working in your code. Then click the right arrow to move forward again.



Use the editor like a Web browser.

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

Click to jump to the definition of the identifier.

After navigating, click the Back arrow to return to your previous location.

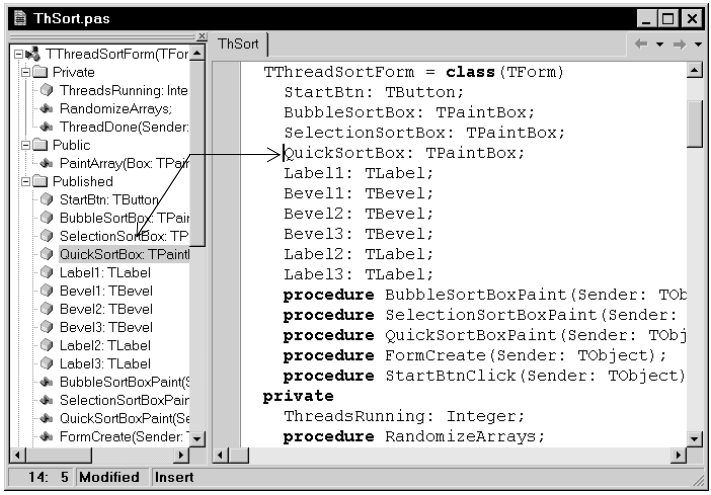
Within the Code editor, you can also move between the declaration of a procedure and its implementation by typing *Ctrl+Shift+*↑ or *Ctrl+Shift+*↓.

For more information...

Search for "Code editor" in the Help index.

Exploring code

When a source file is open in the Code editor, you can use the Code Explorer to see a structured table of contents for the code. The Code Explorer contains a tree diagram showing the types, classes, properties, methods, global variables, and routines defined in your unit. It also shows the other units listed in the **uses** clause.



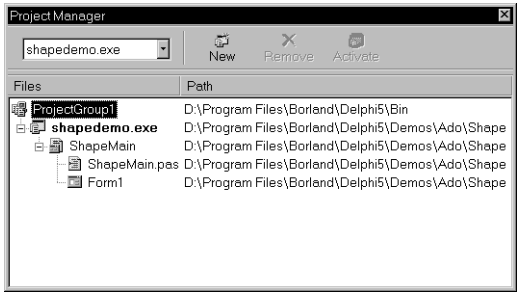
To search for a class, property, method, variable, or routine, just type its name.

For more information...

Search for "Code Explorer" in the Help index.

Managing projects

Use the Project Manager to organize the form and unit files that make up an application. To display the Project Manager, choose View | Project Manager.



The Project Manager shows you the form, unit, resource, object, library, and other files contained in a project. You can use the Project Manager to add and remove files, and you can open any file by double-clicking it.

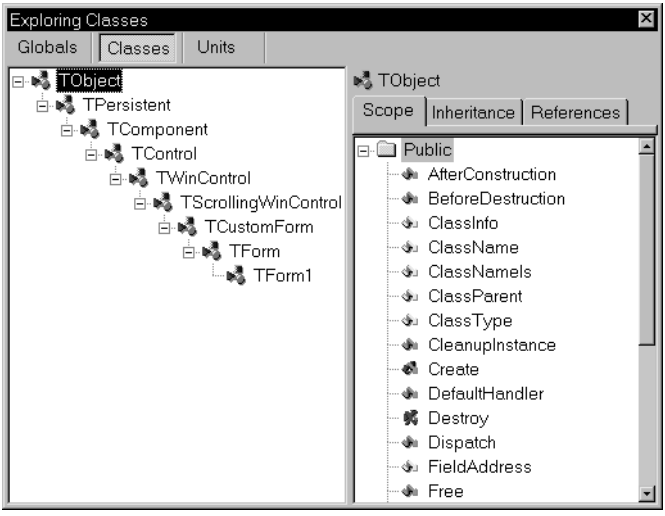
You can combine related projects into a single *project group*. For example, you might use project groups to organize a multi-tiered application or to keep DLLs with executables that use them.

For more information...

Search for "Project Manager" in the Help index.

Browsing project elements and structure

As mentioned earlier, the Code Explorer lets you examine a unit in detail. For a broader view of what's available to you in your project, you can use the Project Browser. It displays the object hierarchies, units, and global symbols within your entire project. Choose View | Browser to display the Project Browser.



You can also expand the scope of the Project Browser to include all symbols available in Delphi's VCL object hierarchy. Choose Tools | Environment Options and check All symbols (VCL included) on the Explorer page.

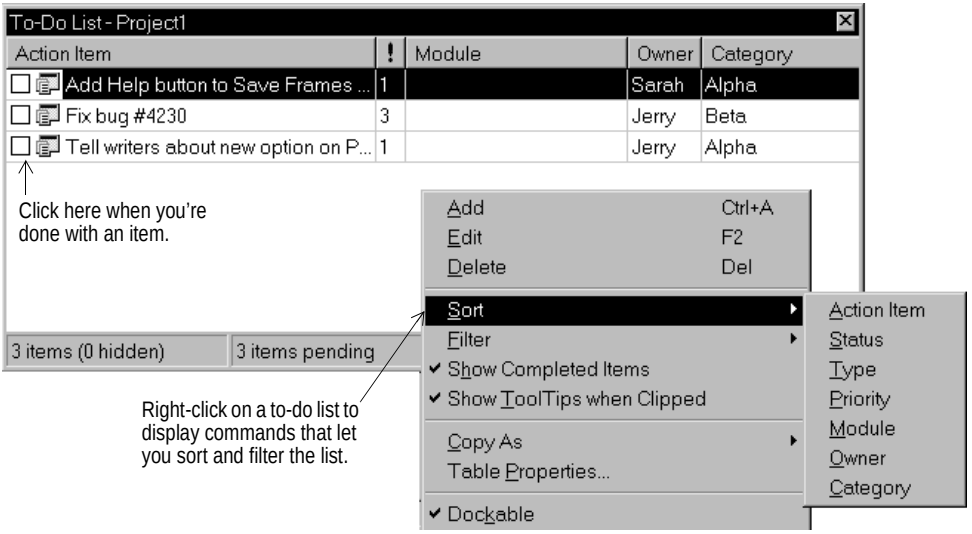
The Project Browser has three tabs that display classes, units, and globals. On the Explorer page of Tools | Environment Options, you can set the scope of the Project Browser and control how source elements are grouped.

For more information...

Search for "Project Browser" in the Help index.

Creating 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 do add or view information associated with a project.



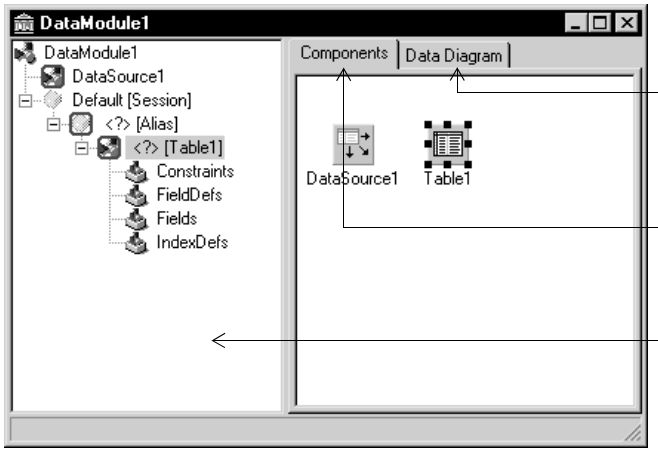
For more information...

Search for "To-Do Lists" in the Help index.

Designing data modules

A data module is a special form that contains nonvisual components. All the components in a data module *could* be placed on ordinary forms alongside visual controls. 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.

The Data Module Designer makes it easy to create data modules. To create a data module, choose File | New and double-click on Data Module.



Delphi opens an empty data module in the Data Module Designer, displays the unit file for the new module in the Code editor, and adds the module to the current project. When you reopen an existing data module, Delphi displays its components in the Data Module Designer.

For more information...

Search for "Data Module Designer" or "data module" in the Help index.

Setting project and environment options

The Project Options dialog, accessed by choosing Project | Options, controls compiler and linker switches, some search paths and output directories, project version information, and other settings that are maintained separately for each application. When you make changes in the Project Options dialog, your changes affect only the current project; but if the Default check box is selected, your selections are also saved as the default settings for new projects. (See "Setting default project options" on page 4-5.)

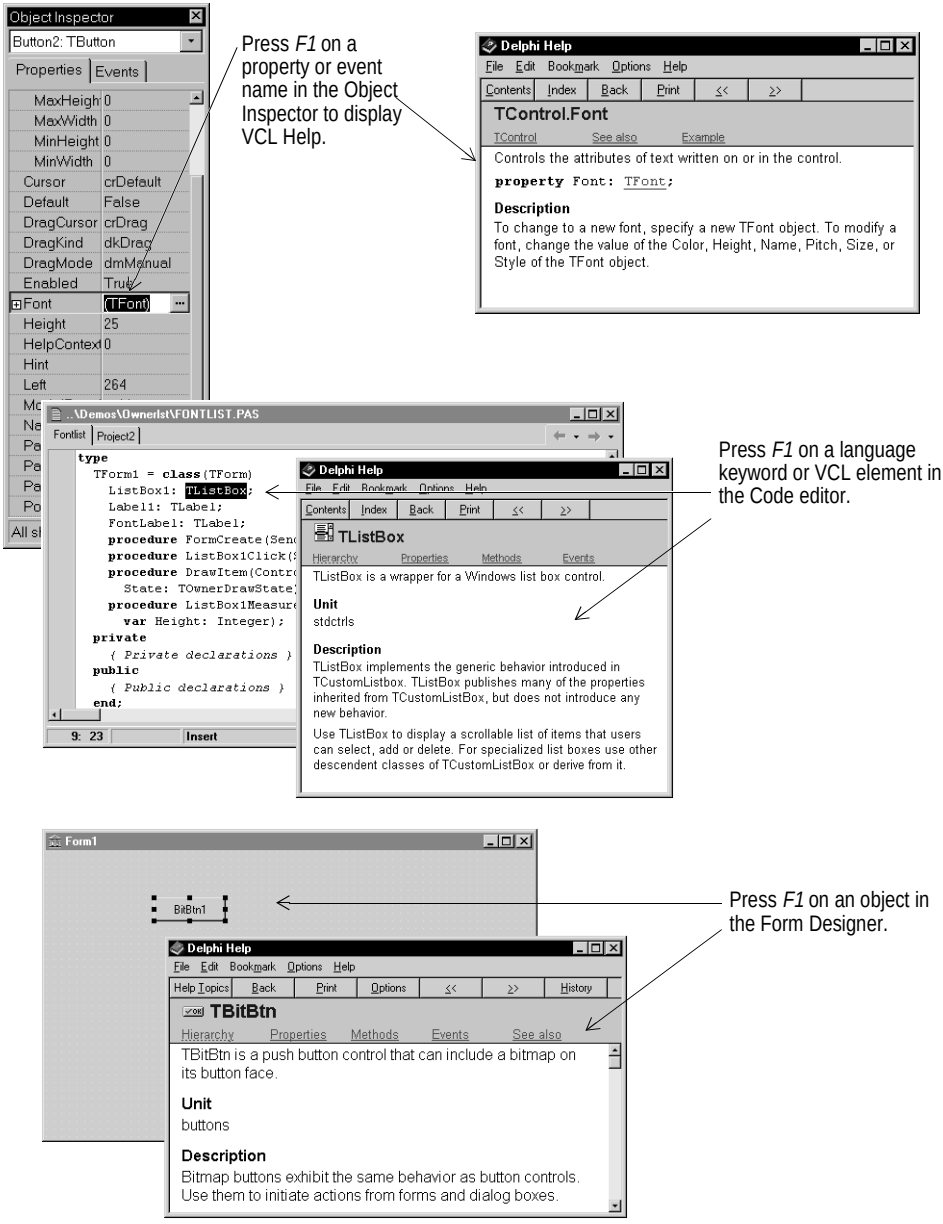
The Environment Options dialog, accessed by choosing Tools | Environment Options, controls global IDE settings for all projects. These include many settings that affect the appearance and behavior of the IDE, as well as some search paths and output directories. You'll find more information about some of these options in "Setting tool preferences" on page 4-6.

For more information...

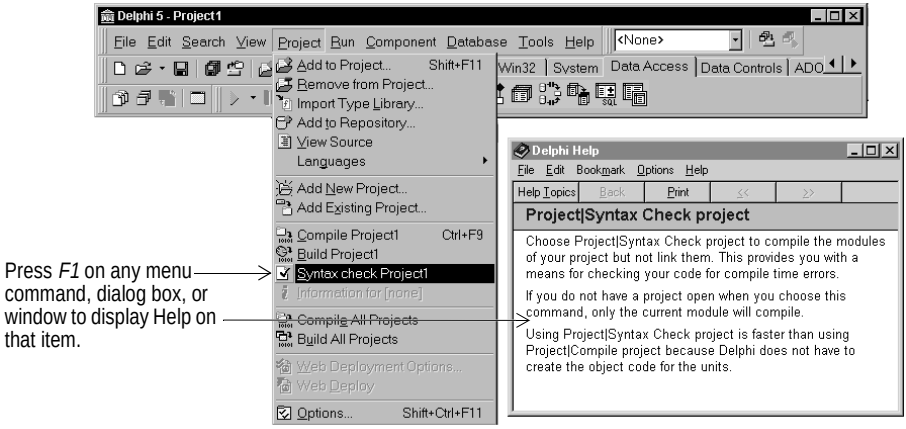
For details about the options on any page of the Project Options or Environment Options dialog, click the Help button on that page. Or search for "Project Options dialog box" or "Environment Options dialog box" in the Help index.

Getting help

The online Help system provides extensive documentation on the VCL and other parts of Delphi. Here are some of the ways you can display Help:



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



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

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*.

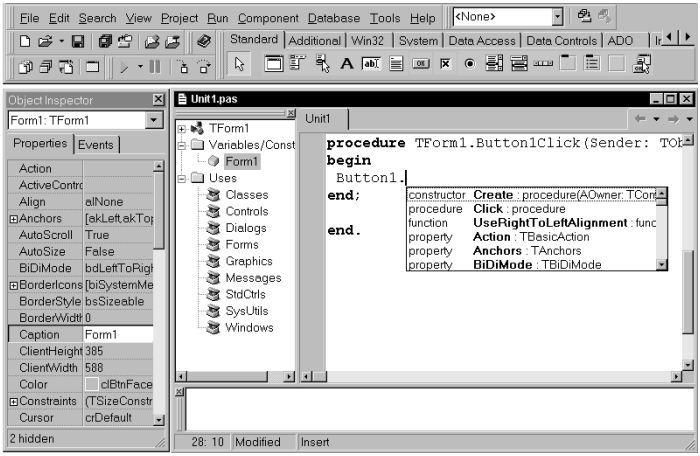
Help with coding

Delphi provides various aids to help you write code. The Code Insight tools display context-sensitive pop-up windows in the Code editor.

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. 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.
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.
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.

To configure these tools, choose Tools | Environment Options and click the Code Insight tab.



When you type the dot in `Button1.`, Delphi displays a list of properties, methods, and events for the class.

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

Class Completion

Class Completion generates skeleton code for classes. Place the cursor anywhere within a class declaration; then press *Ctrl+Shift+C*, or right-click and select 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.

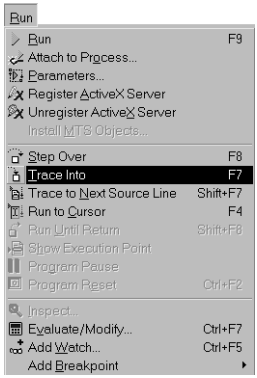
To configure Class Completion, choose Tools | Environment Options and click the Explorer tab.

For more information...

Search for "Code Insight" and "Class Completion" in the Help index.

Debugging applications

The IDE includes an integrated debugger that helps you locate and fix errors in your code. The debugger lets you control program execution, watch variables, and modify data values while your application is running. You can step through your code line by line, examining the state of the program at each breakpoint

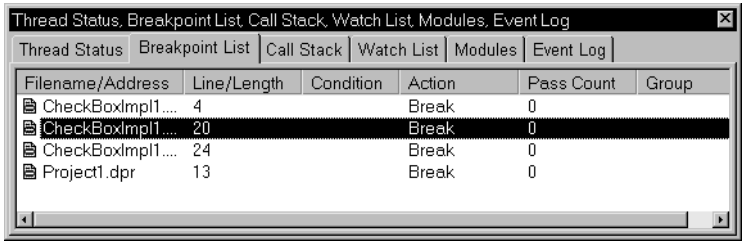


Choose any of the debugging commands from the Run menu.

Some commands are also available on the toolbar.

To use the debugger, you must compile your program with debug information. Choose Project | Options, select the Compiler page, and check Debug Information. Then you can begin a debugging session by running the program from the IDE. To set debugger options, choose Tools | Debugger Options.

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. To learn how to combine debugging windows for more convenient use, see "Docking tool windows" on page 4-1.



You can attach or overlay several debugging windows for easier use.

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 "Customizing desktop settings" on page 4-5.

Some versions of Delphi support multiprocess and remote debugging of distributed applications from either the client or the server. To turn on remote debugging, choose Run | Parameters, click the Remote tab, and check "Debug Project on remote machine"; then choose Project | Options, click the Linker tab, and check "Include remote debug symbols".

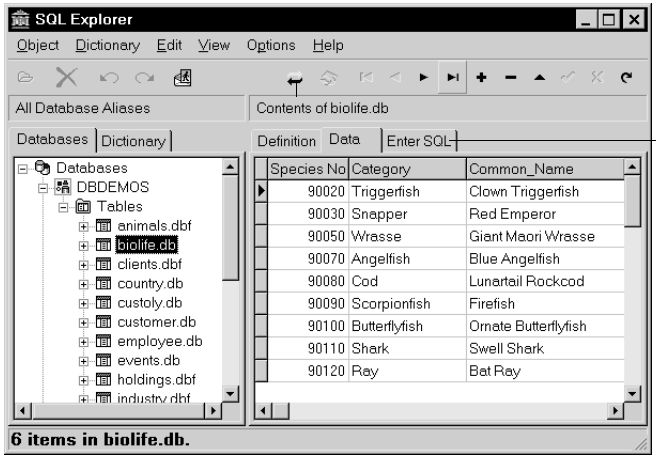
For more information...

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

Exploring databases

The SQL Explorer (or Database Explorer in some editions of Delphi) lets you work directly with a remote database server during application development. For example,

you can create, delete, or restructure tables, and you can import constraints while you are developing a database application.



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

And you can query a database directly.

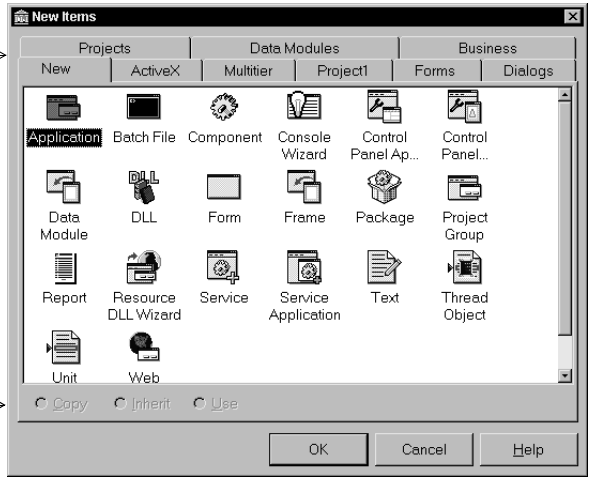
For more information...

Choose Database | Explore to open the Explorer; then press *F1*. Or search for "Database Explorer" in the main Help index.

Templates and 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 to display the New Items dialog when you begin a project. Check the Repository to see if it contains an object that resembles one you want to create.

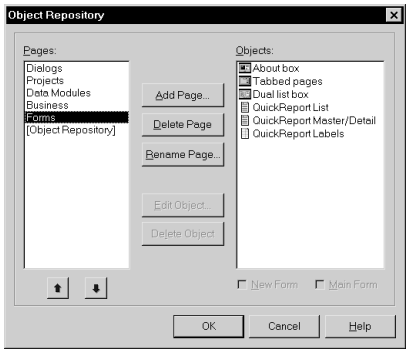
The Object Repository contains many tabbed pages, which include objects like forms, frames, units, and batch files, and wizards to create specialized items.



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

You can add your own objects to the Repository to facilitate reusing them and sharing them with other developers. Reusing objects lets you build families of applications with common user interfaces and functionality; building on an existing foundation also reduces development time and improves quality. The Object Repository provides a central location for tools that members of a development team can access over a network.

To add objects to the Repository, right-click in the New Items dialog and choose Properties, or choose Tools | Repository from the main menu.



For more information...

See “Using Delphi” in the Help contents or search for “Object Repository” in the Help index. Also choose File | New and browse in the Object Repository to see the kinds of templates and wizards you can use as starting points for your applications. The objects available to you will depend on the version of Delphi you purchased.

Chapter

3

Your first application—a brief tutorial

The quickest way to introduce yourself to Delphi is to write an application. This tutorial guides you through the creation of a program that navigates a marine-life database. After setting up access to the database, you'll write an event handler that opens a standard Save As dialog box, allowing you to write information from the database to a file.

Starting a new application

Before beginning a new application, create a folder to hold the source files.

1 Create a folder called Marine in the Projects directory off the main Delphi directory.

2 Open a new project.

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

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 and form files.

3 Choose File | Save All to save your files to disk. When the Save dialog appears, navigate to your Marine folder and save each file using its default name.

Later on, you can save your work at any time by choosing File | Save All.

When you save your project, Delphi creates additional files in your project directory. You don't need to worry about them but don't delete them.

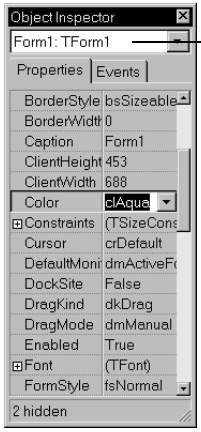
When you open a new project, Delphi displays the project's main form, named *Form1* by default. You'll create the user interface and other parts of your application by placing components on this form.

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

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

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

Next to the form, you'll see the Object Inspector, which you can use to set property values for the form and components you place on it.



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

When an object is selected, the Object Inspector shows its properties.

Setting property values

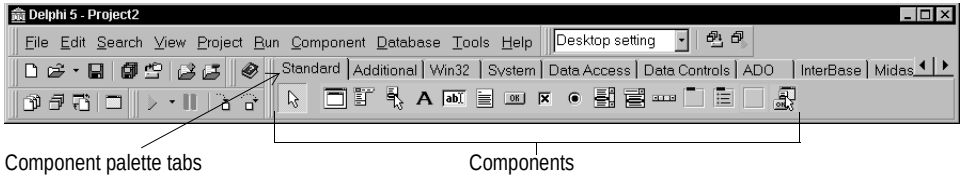
When you use the Object Inspector to set properties, Delphi maintains your source code for you. The values you set in the Object Inspector are called *design-time* settings.

- Set the background color of *Form1* to Aqua.

  Find the form's *Color* property in the Object Inspector and click the drop-down list displayed to the right of the property. Choose *clAqua* from the list.

Adding objects to the form

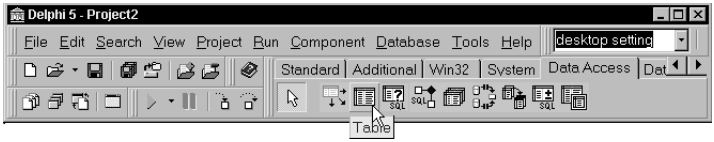
The Component palette represents components by icons grouped onto tabbed pages. Add a component to a form by selecting the component on the palette, then clicking on the form where you want to place it. You can also double-click a component to place it in the middle of the form.



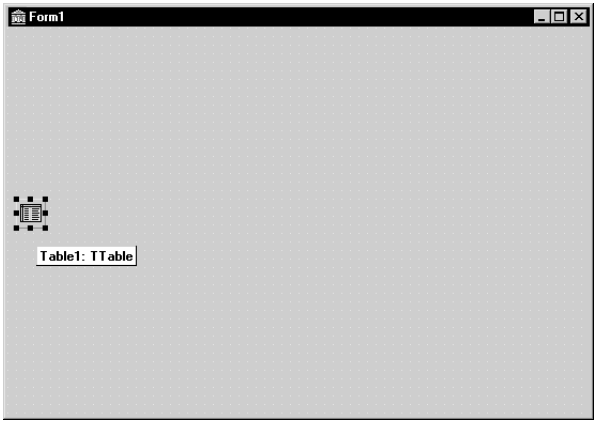
Add a *Table* and a *StatusBar* to the form:

1 Drop a *Table* component onto the form.

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



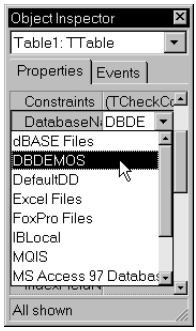
When you find the *Table* component, click it once to select it, then click on the form to place the component. The *Table* component is nonvisual, so it doesn't matter where you put it. Delphi names the object *Table1* by default. (When you point to the component on the form, Delphi displays its name—*Table1*—and the type of object it is—*TTable*.)



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 object when your application is running.

2 Set the *DatabaseName* property of *Table1* to *DBDEMOS*. (*DBDEMOS* is an alias to the sample database that you're going to use.)

Select *Table1* on the form, then choose the *DatabaseName* property in the Object Inspector. Select *DBDEMOS* from the drop-down list.

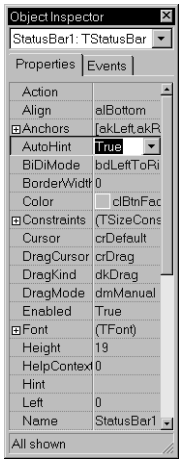


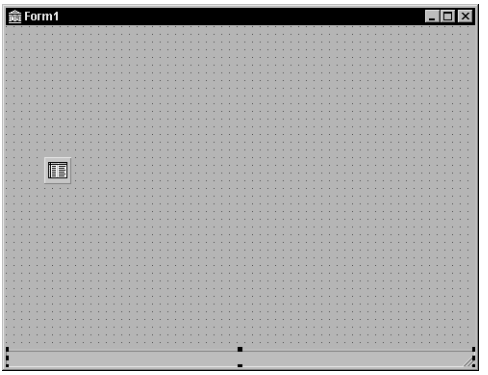
Click the down arrow to display the property drop-down list.

Select *DBDEMOS*.

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

4 Set the *AutoHint* property of the status bar to *True*. The easiest way to do this is to double-click on *False* next to *AutoHint* in the Object Inspector. (Setting *AutoHint* to *True* allows Help hints to appear in the status bar at runtime.)



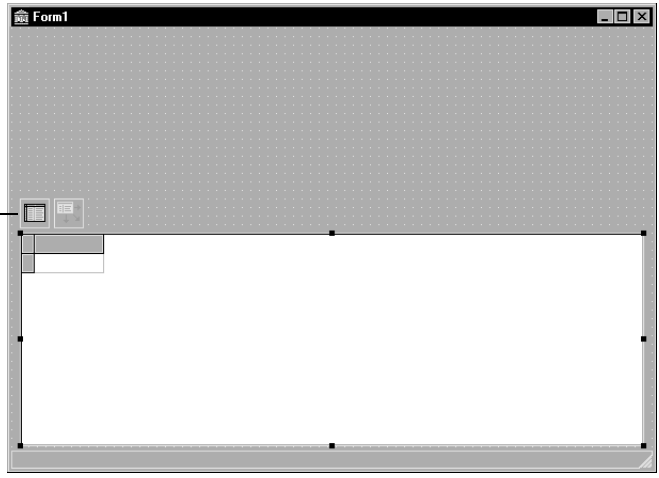


Connecting to a database

The next step is to add database controls and a *DataSource* to your form.

1. From the Data Access page of the Component palette, drop a *DataSource* component onto the form. The *DataSource* component is nonvisual, so it doesn't matter where you put it on the form. Set its *DataSet* property to *Table1*.
2. From the Data Controls page, choose the *DBGrid* component and drop it onto your form. Position it in the lower left corner of the form above the status bar, then expand it by dragging its upper right corner.

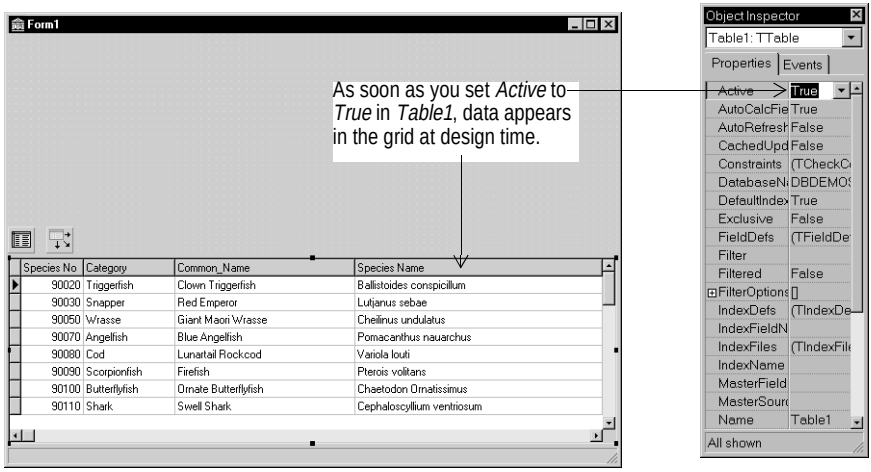
   If necessary, you can enlarge the form by dragging its lower right corner. Your form should now resemble the following figure.



3. Set *DBGrid* properties to align the grid with the form. Double-click *Anchors* in the Object Inspector to display *akLeft*, *akTop*, *akRight*, and *akBottom*; set them all to *True*.
4. Set the *DataSource* property of *DBGrid* to *DataSource1* (the default name of the *DataSource* component you just added to the form).

   Now you can finish setting up the *Table1* object you placed on the form earlier.
5. Select the *Table1* object on the form, then set its *TableName* property to *BIOLIFE.DB*. (*Name* is still *Table1*.) Next, set the *Active* property to *True*.

   When you set *Active* to *True*, the grid fills with data from the BIOLIFE.DB database table. If the grid doesn't display data, make sure you've correctly set the properties of all the objects on the form, as explained in the instructions above. (Also verify that you copied the sample database files into your ...\Borland Shared\Data directory when you installed Delphi.)



The *DBGrid* control displays data at design time, while you are working in the IDE. This allows you to verify that you've connected to the database correctly. You cannot, however, edit the data at design time; to edit the data in the table, you'll have to run the application.

6 Press *F9* to compile and run the project. (You can also run the project by clicking the Run button on the Debug toolbar, or by choosing Run from the Run menu.)

In connecting our application to a database, we've used three components and several levels of indirection. A data-aware control (in this case, a *DBGrid*) points to a *DataSource* object, which in turn points to a dataset object (in this case, a *Table*). Finally, the dataset (*Table1*) points to an actual database table (BIOLIFE), which is accessed through the BDE alias *DBDEMOS*. (BDE aliases are configured through the BDE Administrator.)

data-aware control (Grid) ⟶ DataSource ⟶ dataset (Table) ⟶ BDE ⟶ database

This architecture may seem complicated at first, but in the long run it simplifies development and maintenance. For more information, see "Developing database applications" in the *Developer's Guide* or online Help.

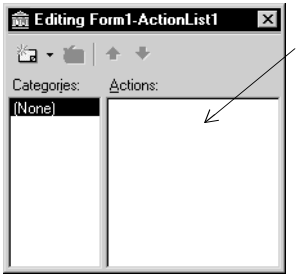
Adding support for a menu and a toolbar

When you run your project, Delphi opens the program in a window like the one you designed on the form. The program is a full-fledged Windows application, complete with Minimize, Maximize, and Close buttons and a Control menu. You can scroll through the BIOLIFE data in the grid.

Though your program already has a great deal of functionality, it still lacks many features usually found in Windows applications. For example, most Windows applications implement menus and toolbars to make them easy to use.

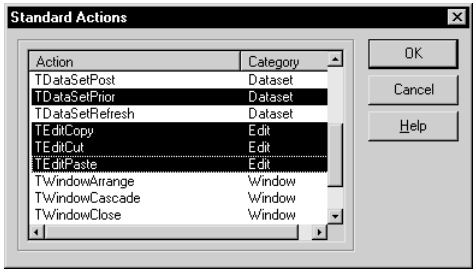
In this section, you'll prepare your application for additional graphical-interface elements by setting up an *ActionList* component. While you can create menus, toolbars, and buttons without using action lists, action lists simplify development and maintenance by centralizing responses to user commands.

1. Click the **X** in the upper right corner to close the application and return to the design-time view of the form.
2. From the Win32 page of the Component palette, drop an *ImageList* onto the form. This is a nonvisual component, so it doesn't matter where you place it. The *ImageList* will contain icons that represent standard actions like *Cut* and *Paste*.
3. From the Standard page of the Component palette, drop an *ActionList* onto the form. This is another nonvisual component.
4. Set the action list's *Images* property to *ImageList1*.
5. Double-click the action list to display the Action List editor.

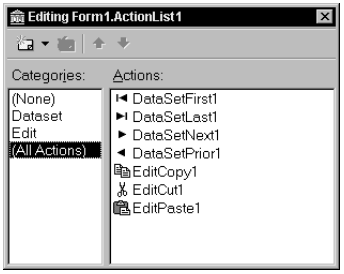


Right-click in the editor and choose New Standard Action to display the Standard Actions list box.

Select the actions you want and click OK. Press *Ctrl* to select multiple actions.



6. Right-click on the Action List editor and choose New Standard Action. The Standard Actions list box is displayed.
7. Select the following actions: *TDataSetFirst*, *TDataSetLast*, *TDataSetNext*, *TDataSetPrior*, *TEditCopy*, *TEditCut*, and *TEditPaste*. (Use the *Ctrl* key to select multiple items.) Then click OK.



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

You'll use these on a toolbar and menu.

8. Click on the **X** to close the Action List editor.

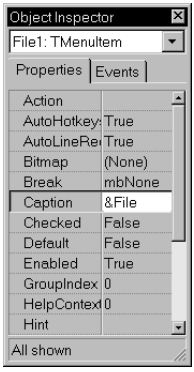
You've added standard actions. Now you're ready to add the menu and toolbar.

Adding a menu

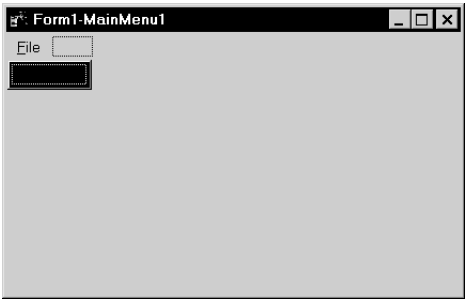
In this section, you'll add a main menu bar with three drop-down menus—File, Edit, and Record—and you'll add menu items to each one using the standard actions in the action list.

1. From the Standard page of the Component palette, drop a *MainMenu* component onto the form. It doesn't matter where you place it.
2. Set the main menu's *Images* property to *ImageList1*.
3. Double-click the menu component to display the Menu Designer.

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



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



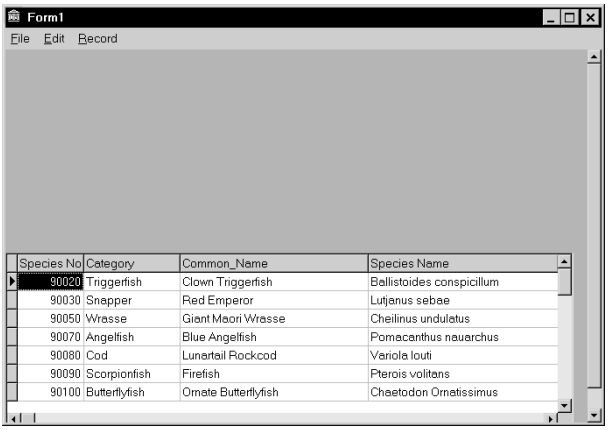
5. Type `&Save` and press *Enter* to create a Save menu item under File.
6. Type a hyphen in the next item under the File menu and press *Enter* to create a separator bar on the menu.
7. Type `E&xit` and press *Enter* to create an Exit menu item under File.
8. Click on the second top-level menu item (to the right of File), type `&Edit`, and press *Enter*. The first menu item under Edit is selected.
   - In the Object Inspector, set its *Action* to *EditCut1* and press *Enter*. The item's caption is automatically set to *Cut*.
   - Select the next menu item (under *Cut*) and set its *Action* to *EditCopy1*.
   - Select the next menu item and set its *Action* to *EditPaste1*.

9 Click on the third top-level menu item (to the right of Edit), type `&Record` as its caption, and press *Enter*. The menu item under Record is selected.

- In the Object Inspector, set its *Action* to *DataSetFirst1*.
- Select the next menu item and set its *Action* to *DataSetPrior1*.
- Select the next menu item and set its *Action* to *DataSetNext1*.
- Select the next menu item and set its *Action* to *DataSetLast1*.

10 Click on the **X** to close the Menu Designer.

Press *F9* to run your program and see how it looks.



Close the application when you're ready to continue.

Adding a toolbar

1 On the Win32 page of the Component palette, double-click the *ToolBar* to add it to the form.

- Set the toolbar's *Indent* property to 4.
- Set its *Images* property to *ImageList1*.
- Set *ShowHint* to *True*.

2 Add buttons to the toolbar.

- With the toolbar selected, right-click and choose New Button three times.
- Right-click and choose New Separator.
- Right-click and choose New Button four more times.

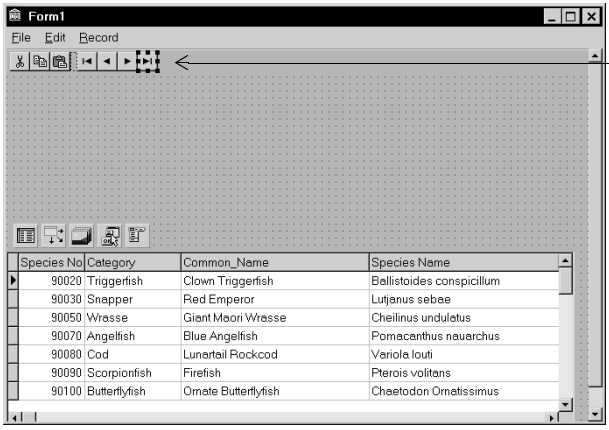
3 Assign actions to the first set of buttons.

- Select the first button and set its *Action* to *EditCut1*.
- Select the second button and set its *Action* to *EditCopy1*.
- Select the third button and set its *Action* to *EditPaste1*.

4 Assign actions to the second set of buttons.

- Select the first button and set its *Action* to *DataSetFirst1*.
- Select the second button and set its *Action* to *DataSetPrior1*.
- Select the third button and set its *Action* to *DataSetNext1*.
- Select the last button and set its *Action* to *DataSetLast1*.

Here's how it looks:



The toobar uses standard actions supplied with Delphi.

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

Check out the toolbar. The First, Prior, Next, and Last buttons work. Select text within a cell in the grid; the Cut, Copy, and Paste buttons work as well.

Close the application when you're ready to continue.

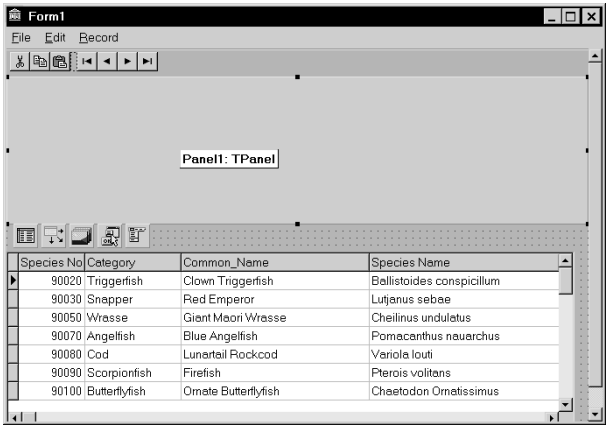
Displaying images

Each record in the BIOLIFE database has a picture associated with it. In this section, we'll expand our application to display pictures.

1 From the Standard page of the Component palette, drop a *Panel* component onto the form below the toolbar. Delphi names this *Panel1* by default.

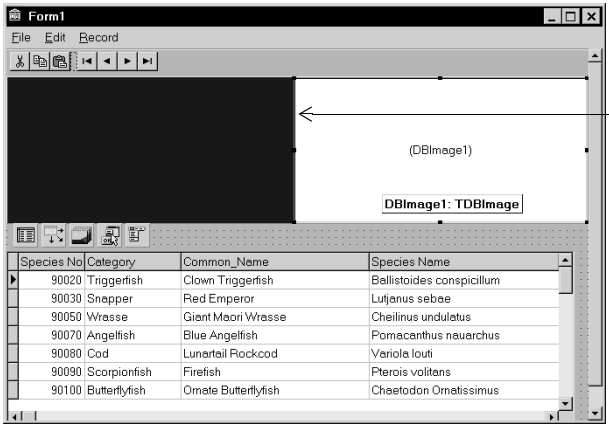
2 In the Object Inspector, delete the *Panel1* string from the panel's *Caption* property. Leave the *Caption* property blank.

3 Align *Panel1* to the top of the form by setting its *Align* property to *alTop*. Next, drag the bottom of the panel down so it fills the portion of the form between the toolbar and the grid.



4 Set the panel's color to *clBlue*.

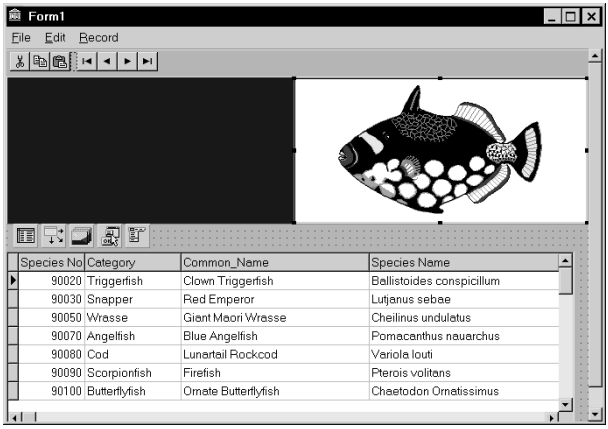
5 From the Data Controls palette page, drop a *DBImage* component on top of *Panel1* and set its *Align* property to *alRight*. Size the *DBImage* by dragging out its left side so your form resembles the one shown in the following figure.



You can drag to set the width of *DBImage*, or you can set its *Width* property in the Object Inspector.

6 Set the *DataSource* property of *DBImage* to *DataSource1*. Then set its *DataField* property to *Graphic*. (In the Object Inspector, the drop-down list next to *DataField* shows the fields in the BIOLIFE table. *Graphic* is one of the field names.)

As soon as you set *DataField* to *Graphic*, the *DBImage* component displays the image of a fish corresponding to the first record of the table. This shows that you have correctly hooked up to the database.



7 Press *F9* to compile and run your application.

Close the application when you're ready to continue.

Adding text and memo objects

In this section, you'll add two components that display individual text fields from the database.

1 Select *Panel1*.

2 From the Data Controls page of the Component palette, drop a *DBMemo* component onto *Panel1* and position it so it occupies the upper left corner of the panel (below the menus and toolbar).

3 Resize the *DBMemo* by dragging its lower right corner. Extend the right edge of the *DBMemo* until it touches the left edge of the *DBImage*. Extend the bottom of the *DBMemo* to within a half inch or so of the bottom of *Panel1*.

4 In the Object Inspector, set the following properties for the *DBMemo*.

- Set *DataSource* to *DataSource1*.
- Set *DataField* to *Notes* (information about the fish appears).
- Set *ScrollBars* to *ssVertical*.

5 Drop a *DBText* component on *Panel1* under the *DBMemo* object. Enlarge the *DBText* so it fills the area under the *DBMemo*, then set its properties as follows.

- Set *DataSource* to *DataSource1*.
- Set *DataField* to *Common_Name*.
- Set *Alignment* to *taCenter*.

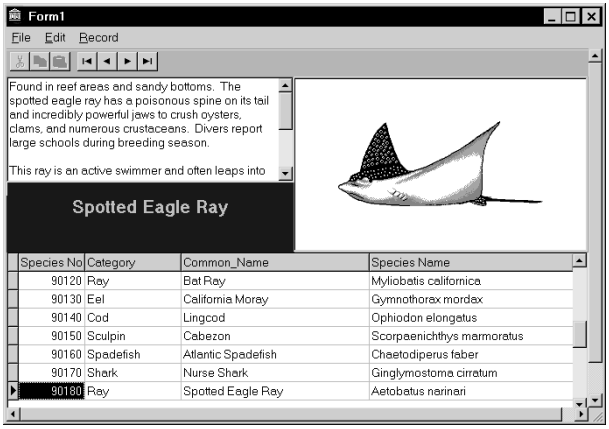
6 Customize the *Font* property of the *DBText* component using the Font editor.

The Font editor is a property editor that you can access through the Object Inspector. Select the *Font* property in the Object Inspector; an ellipsis button appears on the right side of the property setting. Click the ellipsis button to display the Font editor.

Modify the following *DBText* settings using the Font editor, then click OK.

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

7 Compile and run your application by pressing *F9*.



You can view and edit the data in the *DBMemo* component. The *DBText* component displays data for reading only.

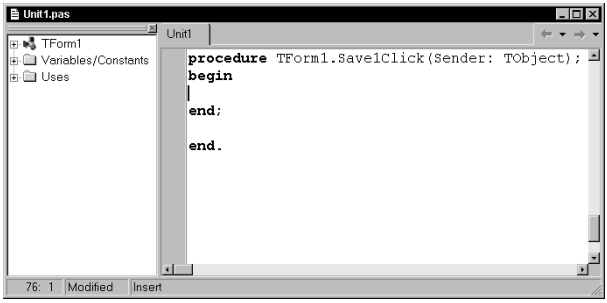
Close the application when you're ready to continue.

Writing an event handler

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, however, you'll write procedures called *event handlers* that respond to user input while the application is running. You'll connect the event handlers to menu items, so that when a menu item is selected your application executes the code in the handler.

1 From the Dialogs page of the Component palette, drop a *SaveDialog* component onto the form. This is a nonvisual component, so it doesn't matter where you place it. Delphi names it *SaveDialog1* by default. (When *SaveDialog*'s *Execute* method is called, it invokes a standard Windows dialog for saving files.)

2 From the menu on your form, choose File | Save. Delphi creates a skeleton event handler for the event that occurs at runtime when the user selects Save from the File menu. The Code editor opens with the cursor inside the event handler.



This event handler is attached to the *OnClick* event of the main menu's first menu item. The menu item is an instance of the class *TMenuItem*, and *OnClick* is its *default event*. Hence the *Save1Click* method is a *default event handler*.

3 Complete the event handler by adding the code shown below in the **var** section and between the outermost **begin** and **end**.

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

This event handler calls the *Execute* method in the *SaveDialog* component. When the dialog box opens and the user specifies a file name, it saves fields from the current database record into a file.

4 To add code for the Exit command, choose File | Exit. Delphi generates another skeleton event handler and displays it in the editor.

```
procedure TForm1.Exit1Click(Sender: TObject);
begin

end;
```

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

```
Close;
```

5 Choose File | Save All to save your work. Then press *F9* to run the application.

You can exit the program using the now functional File | Exit command.

Most components on the Component palette have events, and most components have a default event. A common default event is *OnClick*, which gets called whenever the component is clicked; for example, if you placed a *Button* component (*TButton*) on a form, you would almost certainly write an *OnClick* event handler for it. When you double-click certain objects on a form, Delphi creates a skeleton handler for the default event.

You can also access all of a component's events through the Object Inspector. Select an object on a form, then click the Events tab on the object Inspector; you'll see a list of the object's events. To create a skeleton handler for any event, double-click in the space to its right.

For more information about events and event handlers, see "Developing the application user interface" in the *Developer's Guide* or online Help.

Chapter

4

Customizing the environment

This chapter explains some of the ways you can customize the Delphi development environment.

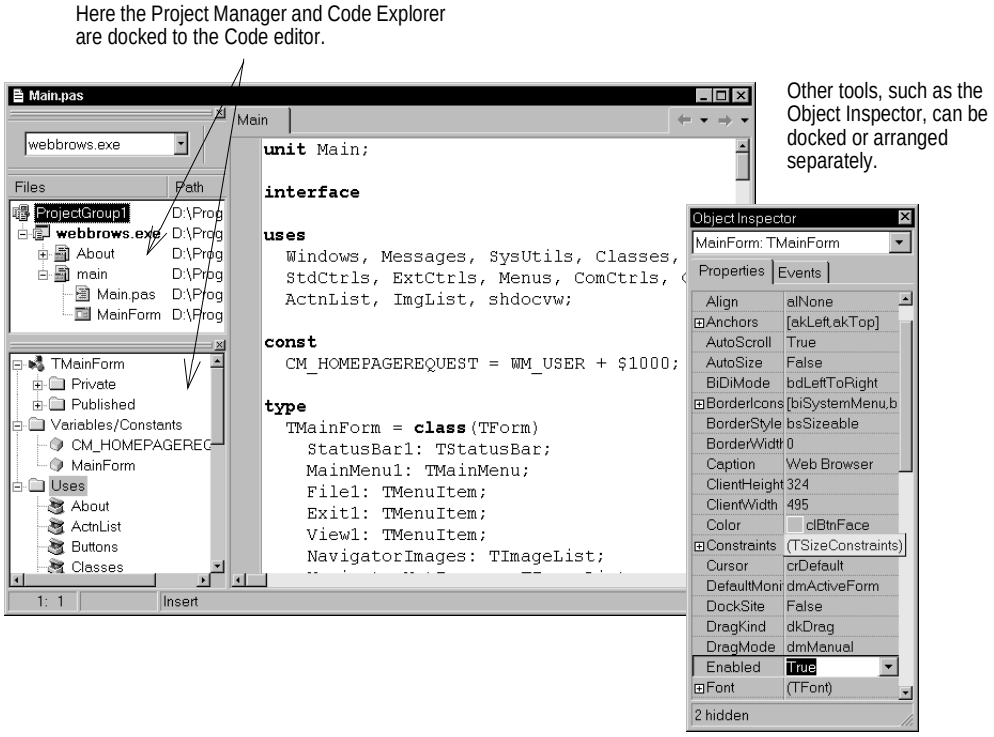
Organizing your work area

The IDE provides many tools to support development, including the Form Designer, Object Inspector, Code editor, Project Manager, Project Browser, and debugging windows. With so many tools available, you'll want to organize your work area for maximum convenience.

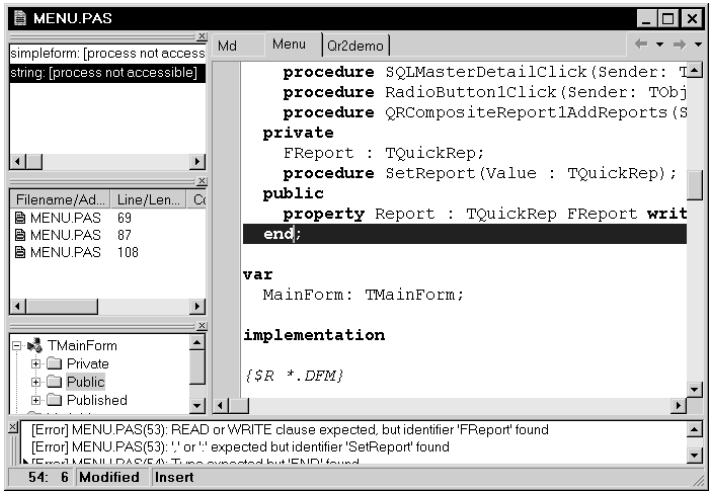
Docking tool windows

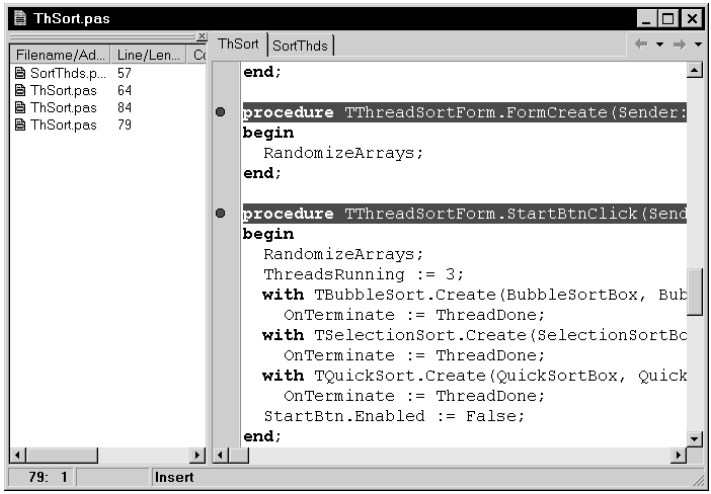
You can open and close individual tool windows and arrange them on the desktop as you wish. Many windows can also be *docked* to one another for easy management. Docking—which means either attaching windows to each other so that they move together or combining several windows into a tabbed "notebook"—helps 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 the Code editor for use while coding and debugging. For example, when you first open Delphi in its default configuration, the Code Explorer is docked to the left of the Code editor. If you want, you can add the Project Manager to the first two to create three docked windows.



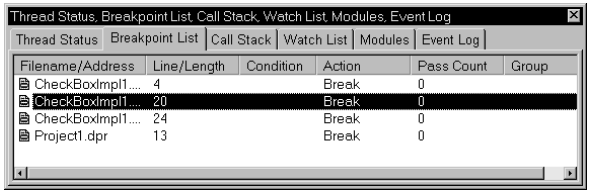
While debugging, you can dock Watch and Breakpoint windows onto the Code editor.





Here, only the breakpoint list is docked to the Code editor.

You can also dock tools to form a tabbed window.



Here, various debugging views are docked to form tabbed pages.

To dock a window, drag it over another window until the first window's rectangular outline becomes narrow and vertical; then release the mouse. To undock a window, click its title bar and drag it in any direction.

For more information...

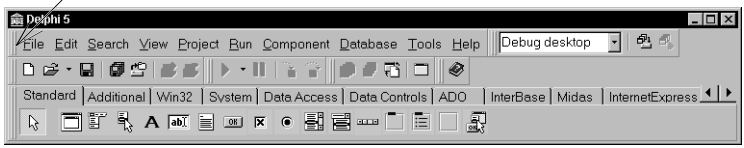
Search for "docking tools" in the Help index.

Arranging menus and toolbars

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

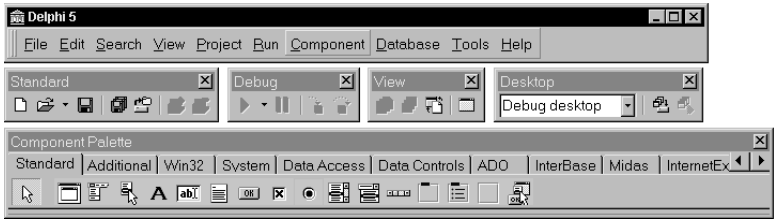
Main window in its default arrangement.

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

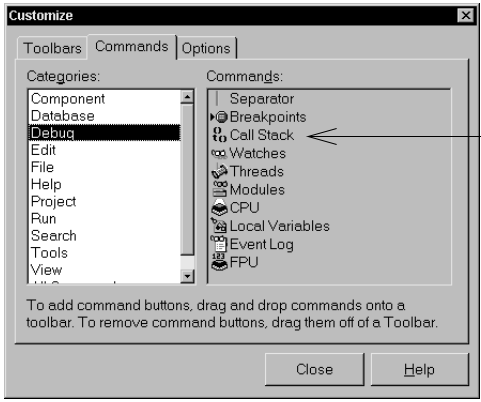


Main window organized differently.

You can even separate parts from the main window and place them elsewhere on the screen or remove them from the desktop altogether.



You can also customize the toolbars by adding or deleting tools.



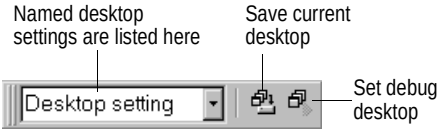
Choose View|Toolbars|Customize.

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

Customizing desktop settings

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

Arrange the desktop as you want including displaying, sizing, and docking particular windows, and placing them where you want on the display. Click the Save current desktop icon on the Desktop toolbar or choose View | Desktops | Save Desktop.



For more information...

Search for "desktop layout" in the Help index.

Setting default project options

The Project Options dialog, accessed by choosing Project | Options, controls settings that are maintained separately for each application you develop. (See "Setting project and environment options" on page 2-10.) However, by choosing the Default check box in the lower left corner of the dialog, you can save your selections as the default settings for all new projects.

Checking Default writes the current settings from the dialog to the options file DEFPROJ.DOF. To restore Delphi's original default settings, delete or rename the DEFPROJ.DOF file, which is located in the Delphi\Bin directory.

Specifying default projects and forms

When you choose File | New Application, a new project opens in the IDE. If you haven't specified a default project, Delphi creates its standard new application with an empty form. But you can select any item from the Projects page of the Object Repository (see "Templates and the Object Repository" on page 2-15) as your *default* project. Once you've specified a default project, Delphi uses it as a template whenever you choose File | New Application. If you select a *wizard* as your default project, Delphi runs the wizard whenever you choose File | New Application; the wizard creates your new project based on your responses to a series of dialog boxes.

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

You always have the option to override your default project or forms by choosing File | New and selecting from the New Items dialog box.

For more information...
See “projects, specifying default” and “forms, specifying default” in the Help index.

Setting tool preferences

The Environment Options dialog, accessed by choosing Tools | Environment Options, controls many aspects of the appearance and behavior of the IDE. Changes made in the Environment Options dialog are global; that is, they affect not just the current project, but projects that you open and compile later.

For more information...
Click the Help button on any page of the Environment Options dialog, or search for “Environment Options dialog box” in the Help index.

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 have editor settings. 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. See “Help with coding” on page 2-12 to learn about these tools.

For more information...
Click the Help button on the following pages in the Editor Options dialog: General, Display, Key Mapping, Color, and Code Insight.

Customizing the Form Designer

The Preferences page of the Environment Options dialog has settings that affect the Form Designer. For example, you can adjust or disable the “grid snap” feature.

For more information...
Click the Help button on the Preferences page of the Environment Options dialog.

Setting Explorer options

The Code Explorer (described in “Exploring code” on page 2-7) opens automatically when you start Delphi. You can disable this behavior—and set other options for the Code Explorer—from the Explorer page of the Environment Options dialog.

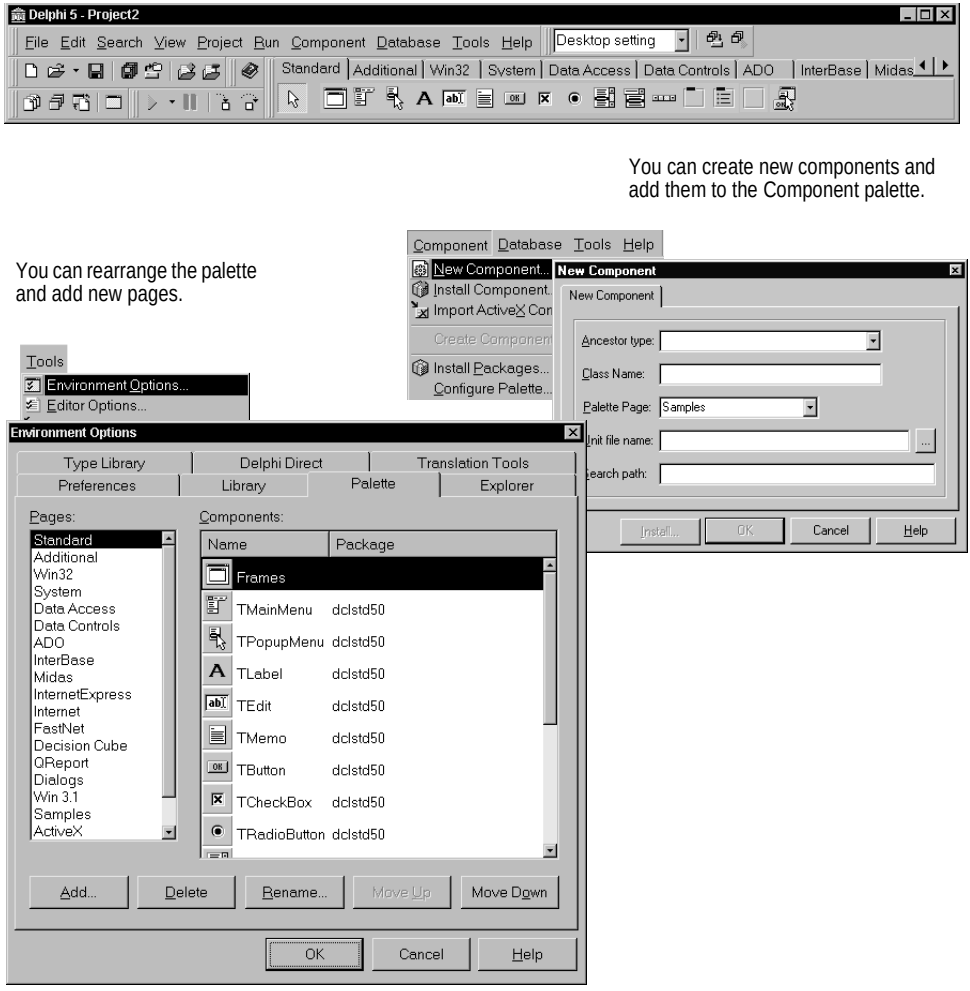
You can also set options such as the initial browser view and the browser scope which affect the Code Browser on the Explorer page of the Environment Options dialog. The Code Browser is shown in “Browsing project elements and structure” on page 2-8.

For more information...
Click the Help button on the Explorer page of the Environment Options dialog.

Customizing the Component palette

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

- hiding or rearranging components
- adding, removing, rearranging, or renaming pages
- installing new components
- creating component templates and adding them to the palette



Arranging the Component palette

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

- Choose Component | Configure Palette.
- Choose Tools | Environment Options and click the Palette tab.
- Right-click on the Component palette an select Properties.

For more information...

Click the Help button in the Palette Properties dialog.

Installing components

You can supplement the components in the VCL with custom components that you write yourself or obtain from third-party developers. To make new components available at design time, you need to install them in the IDE.

For more information...

To install third-party components, follow the vendor's instructions. To learn about writing your own components, see "Creating custom components" in the *Developer's Guide* or online Help.

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. 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.

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 for reuse on other forms.

To create a component template, simply arrange one or more components on a form, set their properties in the Object Inspector, and select all of the components. Then choose Component | Create Component Template. When the Component Template Information dialog opens, you can 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.

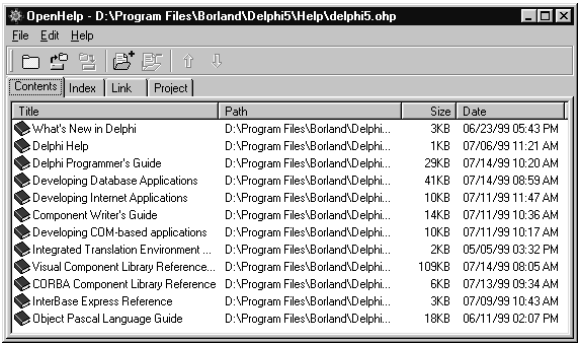
For more information...

Search for "Template" in the Help index or choose Component | Create Component Template and press *F1*.

Customizing the Help system

Delphi's online Help system comprises more than a dozen WinHelp (.HLP) files and includes documentation for the IDE, the Visual Component Library, and additional products and tools supplied with Delphi. A utility called OpenHelp allows you to customize the Help system by choosing which files to make available through the master table of contents, the index, and the IDE's context-sensitive Help.

To start OpenHelp, choose Help | Customize.



The default Help system is set up in the Delphi5.ohp file in the Help directory. You can customize your Help system by adding or deleting files.

This list controls which Help topics appear in the Help Contents.

OpenHelp lets you add any WinHelp files to Delphi's Help system, including documentation for third-party products. OpenHelp also allows you to remove references to obsolete Help files from the system registry.

For an overview of the Help files supplied with Delphi, see "Online Help" on page 1-2.

For more information...

Choose Help | Customize, then choose Help | Contents from the OpenHelp main window.

Chapter

5

Programming with Delphi

The following sections provide an overview of software development with Delphi and describe some features that are not covered in earlier chapters of this *Quick Start*.

Development tools and features

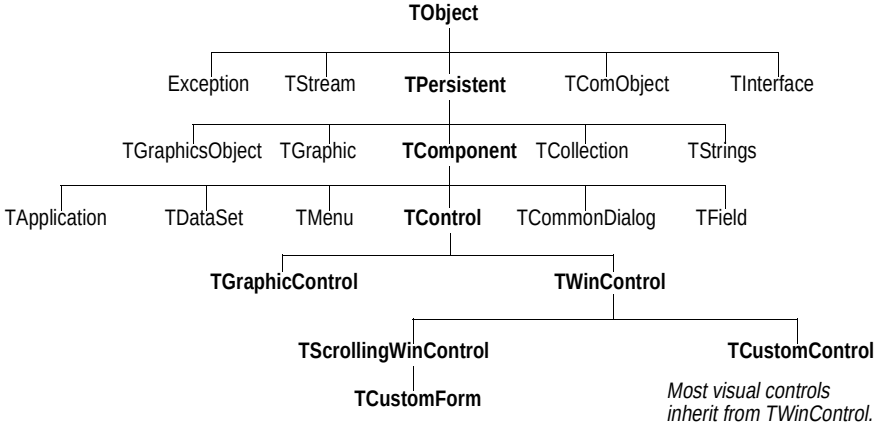
The integrated development environment (IDE) includes the Form Designer, Object Inspector, Component palette, Project Manager, Code Explorer, Code editor, Data Module Designer, software localization tools, debugger, and many other tools. The particular features and components available to you will depend on which version of Delphi you've purchased.

All versions of Delphi support general-purpose 32-bit Windows programming, multithreading, COM and Automation controllers, and multiprocess debugging. Some versions add support for server applications such as COM servers and Web applications, database development with report and chart generation for a variety of DBMS back ends, support for SQL database servers (such as Oracle 8 and InterBase), Microsoft Transaction Server (MTS), multi-tiered database applications, CORBA, and decision-support systems. For up-to-date product information, refer to www.borland.com or contact your Inprise distributor.

Using the VCL

Delphi comes with components that are part of a class hierarchy called the Visual Component Library (VCL). The VCL includes 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 diagram below shows some of the principal classes that make up the VCL.



Objects descended from *TComponent* have properties and methods that allow them to be installed on the Component palette and added to Delphi forms. Because VCL 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 button control, you don't have to write code to handle Windows messages when the button is clicked; you are responsible only for the application logic that executes in response to the event.

Most versions of Delphi come with complete source code for the VCL. In addition to supplementing the online documentation, the VCL source code provides invaluable examples of Object Pascal programming techniques.

For more information...

See "Visual Component Library Reference" and "Creating Custom Components" in the online Help.

Exception handling

Delphi's error-handling is based on *exceptions*, which are special objects generated in response to unanticipated input or faulty program execution. Exceptions can be raised at both design time and runtime, and the VCL contains many exception classes that are associated with specific error conditions. In your applications, you'll want to write *exception handlers* to deal gracefully with runtime errors. Exceptions can also be a valuable debugging tool, since the class of an exception often provides a clue about what caused it to be raised.

For more information...

See the entries for "Exception" and its specialized descendant classes in the online VCL reference. Look up "exception handling" in the Help index.

Database connectivity and utilities

Delphi and the VCL offer a variety of connectivity tools to simplify the development of database applications. The Borland Database Engine (BDE) is a collection of drivers that support many popular database formats, including dBASE, Paradox, FoxPro, 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.

Delphi includes components that you can use to access data through InterBase Express (IBX). IBX applications provide access to advanced InterBase features and offer the highest performance component interface for InterBase 5.5 and later.

IBX is based on the custom data access Delphi component architecture, and is integrated with the Data Module Designer. IBX is compatible with Delphi's library of data-aware components, and does not require the Borland Database Engine (BDE).

You can create database tables at design time in the Form Designer. First, create field definitions using the Object Inspector, then right-click on the table component and choose Create Table.

Some versions of Delphi include components to connect to databases using ActiveX Data Objects (ADO). ADO is Microsoft's high-level interface to any data source, including relational and non-relational databases, email and file systems, text and graphics, and custom business objects.

For more information...

See "Developing Database Applications" in the *Developer's Guide* or online Help.

In addition, Delphi provides the following tools for database developers.

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...

Start the BDE Administrator from the Delphi program group under the Windows Start menu. 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 to open the Explorer; then press *F1*. Or search for "Database Explorer" in the main 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...

Start the Database Desktop from the Delphi program group under the Windows Start menu. Then press *F1*.

Data Dictionary

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 for additional sharing of information.

For more information...

Search for "Data Dictionary" in the Help index.

Kinds of development project

You can use Delphi to write Windows GUI applications, console applications, service applications, dynamic-link libraries (DLLs), packages (a special type of DLL used by Delphi), and other programs.

Applications and servers

Delphi has features that make it easy to write distributed applications, including client/server, multi-tiered, and Web-based systems. In addition to support for standards like COM and a suite of Internet components, some versions of Delphi provide extensive tools for CORBA development.

For more information...

See "Building applications, components, and libraries" and "Developing distributed applications" in the *Developer's Guide* or online Help.

DLLs

Dynamic-link libraries (DLLs) are compiled modules containing routines that can be called by applications and by other DLLs. Since a DLL contains sharable code or resources, it is typically used by more than one application.

For more information...

Search for "DLLs" in the Help index.

Custom components and packages

A *package* is a special dynamic-link library used by Delphi applications, the IDE, or both. While packages can be used in a variety of ways, their most common purpose is the encapsulation Delphi components. In fact, all components installed in the IDE must be compiled as packages.

The components that come with Delphi are preinstalled in the IDE 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 encapsulate particular kinds of behavior that require custom components.

Custom components supplement the VCL while promoting code reuse and consistency across applications. Many Delphi components are available through third-party developers, and Delphi provides a New Component wizard that makes it easy to create and install components on your own.

For more information...

See "Creating Custom Components" in the *Developer's Guide* or online Help. Search for "packages" in the Help index.

Frames

A frame (*TFrame*), like a form, is a container for other components. In some ways, 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.

For more information...

Search for "frames" and "TFrame" in the Help index.

COM and ActiveX

Delphi supports Microsoft's COM (Component Object Model) standard and provides wizards for easy creation of ActiveX controls. 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...

Search for "COM" and "ActiveX" in the 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...

Search for "type libraries" in the Help index.

Deploying applications

When you deploy an application, be sure to supply all the required files—including executables, DLLs, packages, and BDE drivers—to your users. To make this process easier, Delphi includes a special version of InstallShield Express, a popular tool for developing installation utilities.

For more information...

Search for "deploying applications" in the Help index.

Internationalizing applications

Delphi offers many features for internationalizing and localizing applications. Support for input method editors (IMEs) and extended character sets is provided throughout the VCL, and tools like the Resource DLL wizard make it easy to prepare a project for localization. To get the maximum benefit from these features, you need to start thinking about internationalization requirements as early as possible in the development process.

The Integrated Translation Environment (ITE), available in some versions of Delphi, is a suite of tools for software localization and simultaneous development for different locales. It is integrated with the IDE to let you manage multiple localized versions of an application as part of a single project.

The ITE includes three tools:

- Translation Manager, a grid for viewing and editing translated resources
- Translation Repository, a sharable database for translations
- Resource DLL wizard, a DLL wizard that generates and manage resource DLLs

For more information...

Search for "international applications" and "ITE" in the Help index.

Index

V

W

Y