



Quick Start



Borland®
C++Builder™ 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 C++Builder 6 product for a complete list of files that you can distribute in accordance with the C++Builder 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. Please refer to the product CD or the About dialog box for the list of applicable patents.

COPYRIGHT © 1983–2002 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. All other marks are the property of their respective owners.

Printed in the U.S.A.

CPE1360WW21000 6E2R0102
0203040506-9 8 7 6 5 4 3 2 1
D3

Contents

Chapter 1

Introduction 1-1

What is C++Builder?	1-1
Registering C++Builder	1-2
Finding information	1-3
Online Help	1-3
F1 Help	1-4
Printed documentation	1-6
Developer support services and Web site	1-6
Typographic conventions	1-6

Chapter 2

A tour of the environment 2-1

Starting C++Builder	2-1
The IDE	2-1
The menus and toolbars	2-3
The Component Palette, Form Designer, and Object Inspector	2-4
The Object TreeView	2-4
The Object Repository	2-5
The Code Editor	2-6
Code Insight	2-6
Code Browsing	2-7
The Diagram page	2-7
Viewing form code	2-9
The ClassExplorer	2-9
The Project Manager	2-9
To-do lists	2-10

Chapter 3

Programming with C++Builder 3-1

Creating a project	3-1
Adding data modules	3-2
Building the user interface	3-2
Placing components on a form	3-2
Setting component properties	3-4
Writing code	3-5
Writing event handlers	3-5
Using the VCL and CLX libraries	3-6
Compiling and debugging projects	3-7
Deploying applications	3-8
Internationalizing applications	3-8
Types of projects	3-9
CLX applications	3-9
Web server applications	3-9

Database applications	3-10
BDE Administrator	3-11
SQL Explorer (Database Explorer)	3-11
Database Desktop	3-11
Data Dictionary	3-12
Custom components	3-12
DLLs	3-12
COM and ActiveX	3-12
Type libraries	3-13

Chapter 4

Creating a text editor—a tutorial 4-1

Starting a new application	4-1
Setting property values	4-2
Adding components to the form	4-3
Adding support for a menu and a toolbar	4-6
Action Manager editor and Action List editor differences	4-6
Adding menu and toolbar images (Enterprise and Professional)	4-7
Adding actions to the Action Manager (Enterprise and Professional)	4-8
Adding standard actions (Enterprise and Professional)	4-11
Adding a menu (Enterprise and Professional)	4-12
Adding a toolbar (Enterprise and Professional)	4-13
Adding an image list and images (Personal edition)	4-13
Adding actions to the action list (Personal edition)	4-15
Adding standard actions to the action list (Personal edition)	4-17
Adding a menu (Personal edition)	4-18
Adding a toolbar (Personal edition)	4-20
Clearing the text area (all editions)	4-22
Writing event handlers	4-22
Creating an event handler for the New command	4-23
Creating an event handler for the Open command	4-25
Creating an event handler for the Save command	4-27

Creating an event handler for the Save As command	4-27
Creating a Help file	4-29
Creating an event handler for the Help Contents command	4-30
Creating an event handler for the Help Index command	4-30
Creating an About box	4-31
Completing your application.	4-33

Chapter 5

Creating a CLX database application—a tutorial **5-1**

Overview of database architecture.	5-1
Creating a new CLX application	5-2
Setting up data access components	5-2
Setting up the database connection.	5-3
Setting up the unidirectional dataset.	5-5
Setting up the provider, client dataset, and data source	5-5
Designing the user interface	5-6
Creating the grid and navigation bar	5-6
Adding support for a menu.	5-8
Adding a menu	5-9
Adding a button	5-11
Displaying a title and an image	5-11
Writing an event handler	5-13

Writing the Update Now! command event handler.	5-13
Writing the Exit command event handler	5-13
Writing the FormClose event handler	5-14

Chapter 6

Customizing the desktop **6-1**

Organizing your work area	6-1
Arranging menus and toolbars	6-1
Docking tool windows	6-2
Saving desktop layouts	6-4
Customizing the Component palette	6-5
Arranging the Component palette.	6-5
Creating component templates	6-5
Installing component packages	6-6
Using frames	6-7
Adding ActiveX controls	6-8
Setting project options	6-8
Setting default project options	6-8
Specifying project and form templates as the default.	6-8
Adding templates to the Object Repository	6-9
Setting tool preferences.	6-10
Customizing the Form Designer.	6-10
Customizing the Code Editor	6-11

Index

I-1

Introduction

This *Quick Start* provides an overview of the C++Builder 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 C++Builder.

Chapter 2, “A tour of the environment” describes the main tools on the C++Builder desktop, or integrated desktop environment (IDE). Chapter 3, “Programming with C++Builder” 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, “Creating a CLX database application—a tutorial” walks you through the creation of a database application. Chapter 6, “Customizing the desktop” describes how you can customize the C++Builder IDE for your development needs.

What is C++Builder?

C++Builder is an object-oriented, visual programming environment for rapid application development (RAD). Using C++Builder, you can create highly efficient applications for Microsoft Windows XP, Microsoft Windows 2000 and Microsoft Windows 98 with a minimum of manual coding. C++Builder 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.

Registering C++Builder

C++Builder can be registered in several ways. The first time you launch C++Builder after installation, you will first be prompted to enter your serial number and authorization key. Once this has been entered, a registration dialog offers three choices:

- Register and activate the software online.

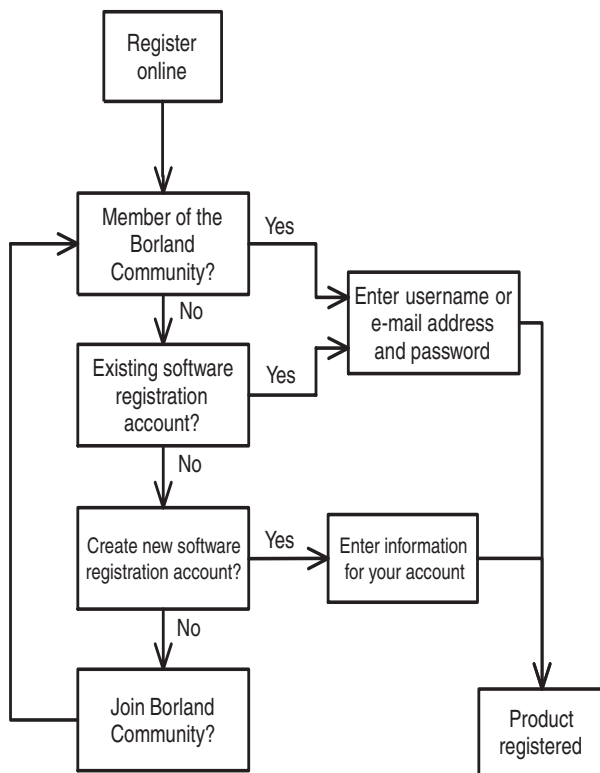
Use this option to register online using your existing internet connection.

- Register or activate by phone or web browser.

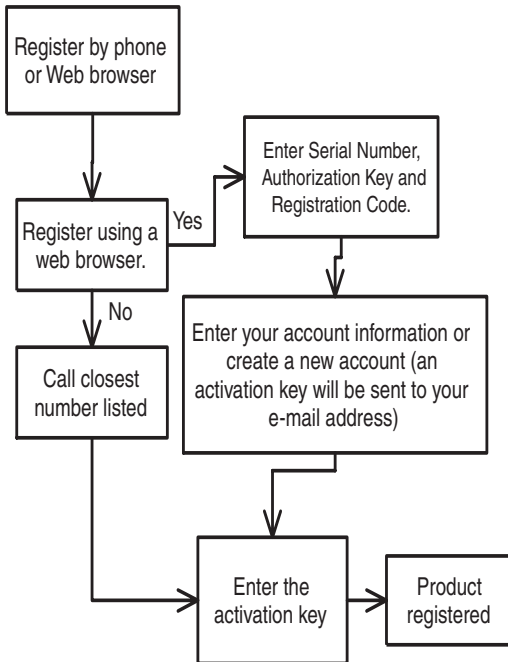
Use this option to register by phone or through your web browser. If you received an activation key via email, use this option to enter it.

- I will register at a later time.

Online registration is the easiest way to register C++Builder, but it requires that you have an active connection to the internet. If you are already a member of the Borland Community, or have an existing software registration account, simply enter the relevant account information. This will automatically register C++Builder. If not, the registration process provides a way to create either of these accounts.



The second option (register or activate by phone or web browser) is useful if the machine you are installing on is not connected to the internet, you are behind a firewall that is blocking online registration, or if you have previously received an activation key.



Note Unless you have a specific reason not to, use the online registration option.

Finding information

You can find information on C++Builder 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

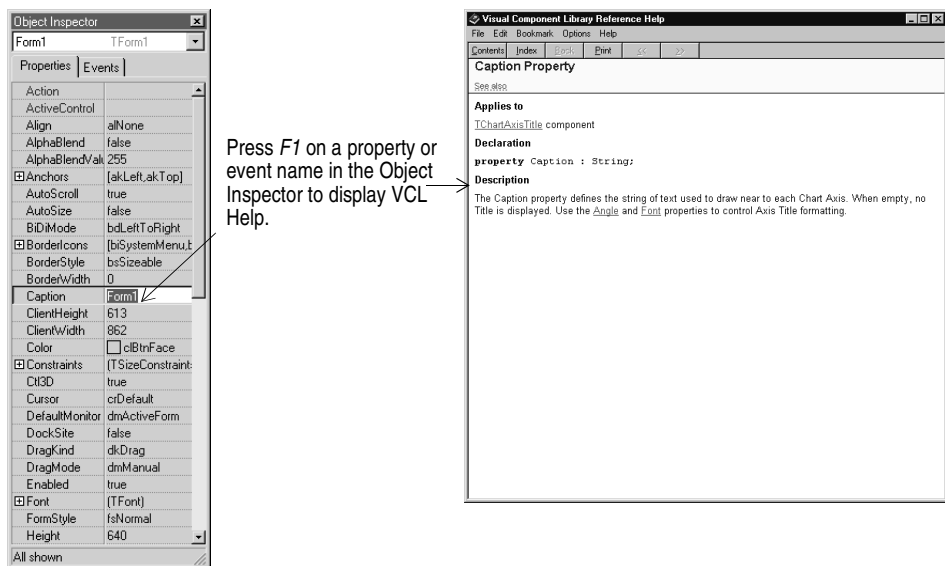
The online Help system provides detailed information about user interface features, language implementation, programming tasks, and the components in the Visual Component Library Reference (VCL) and Borland Component Library for Cross

Reference (CLX). It includes all the material in the *Developer's Guide*, and a host of Help files for other features bundled with C++Builder.

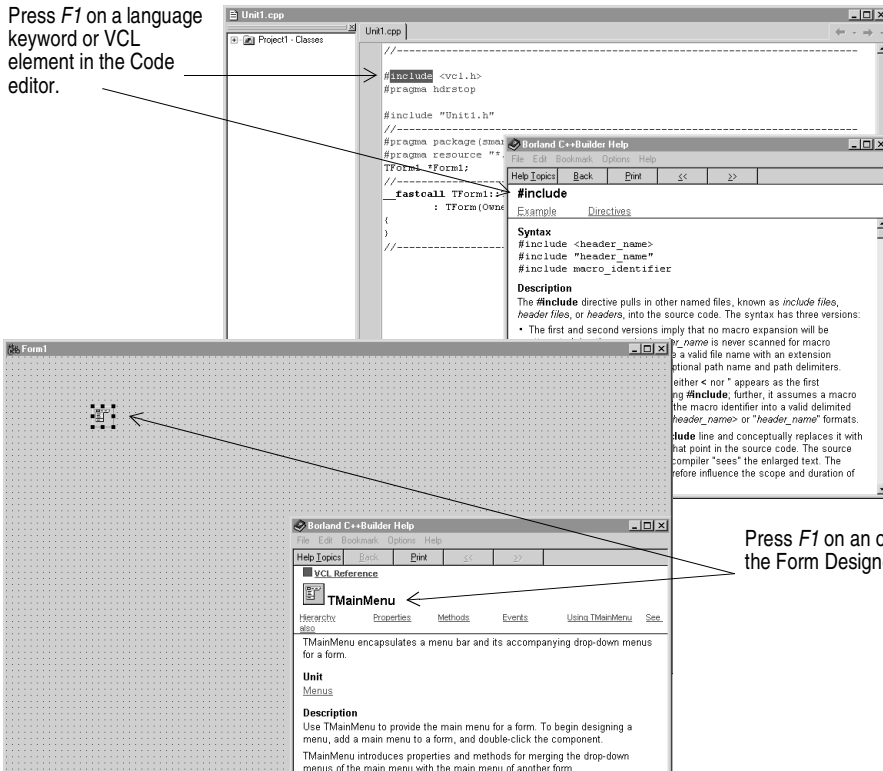
To view the table of contents, choose Help | C++Builder Help and Help | C++Builder Tools, and click the Contents tab. To look up VCL or CLX objects or any other topic, click the Index or Find tab and type your request.

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*.

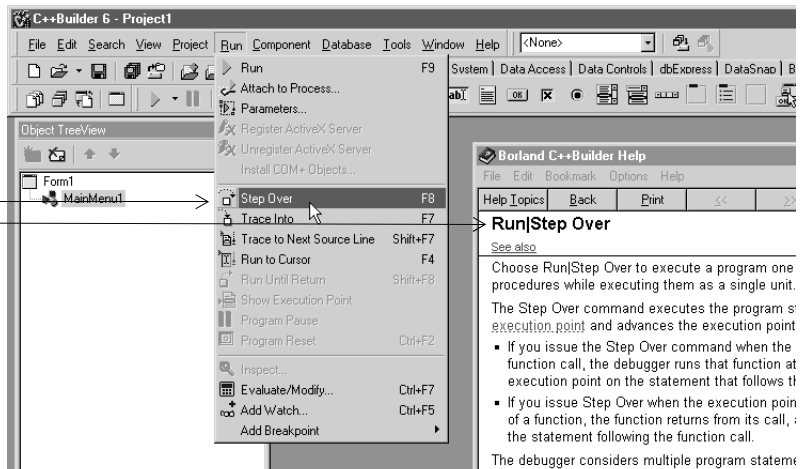


Press **F1** on a language keyword or VCL element in the Code editor.



Press **F1** on an object in the Form Designer.

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



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**.

Printed documentation

This *Quick Start* is an introduction to C++Builder. 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 C++Builder developers exchange information, tips, and techniques. The site also includes a list of books about C++Builder, additional C++Builder technical documents, and Frequently Asked Questions (FAQs).

Typographic conventions

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

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.
<i>Italics</i>	Italicized words in text represent C++Builder identifiers, such as variable or type names. Italics are also used to emphasize certain words, such as new terms.
<i>Keycaps</i>	This typeface indicates a key on your keyboard. For example, "Press <i>Esc</i> to exit a menu."

A tour of the environment

This chapter explains how to start C++Builder and gives you a quick tour of the main parts and tools of the integrated development environment (IDE).

Starting C++Builder

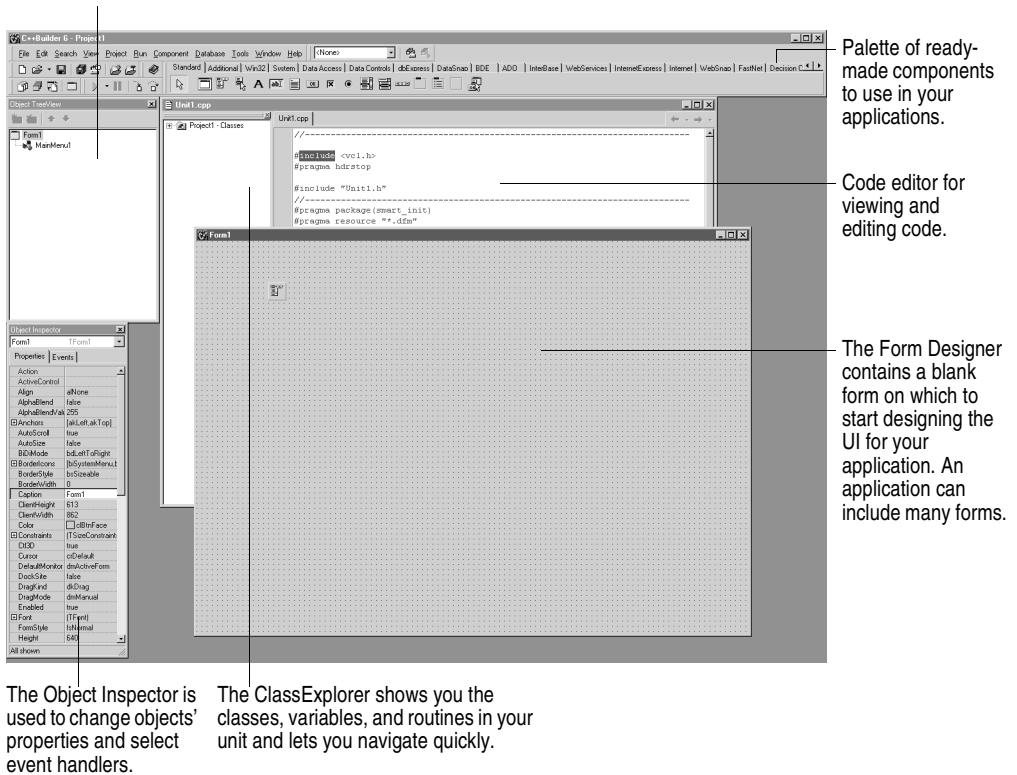
You can start C++Builder in the following ways:

- Double-click the C++Builder icon (if you've created a shortcut).
- Choose Programs | Borland C++Builder from the Windows Start menu.
- Choose Run from the Windows Start menu, then enter Bcb.
- Double-click Bcb.exe in the CBuilder6\Bin directory.

The IDE

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

The Object TreeView displays a hierarchical view of your components' parent-child relationships.



C++Builder'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 C++Builder 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.

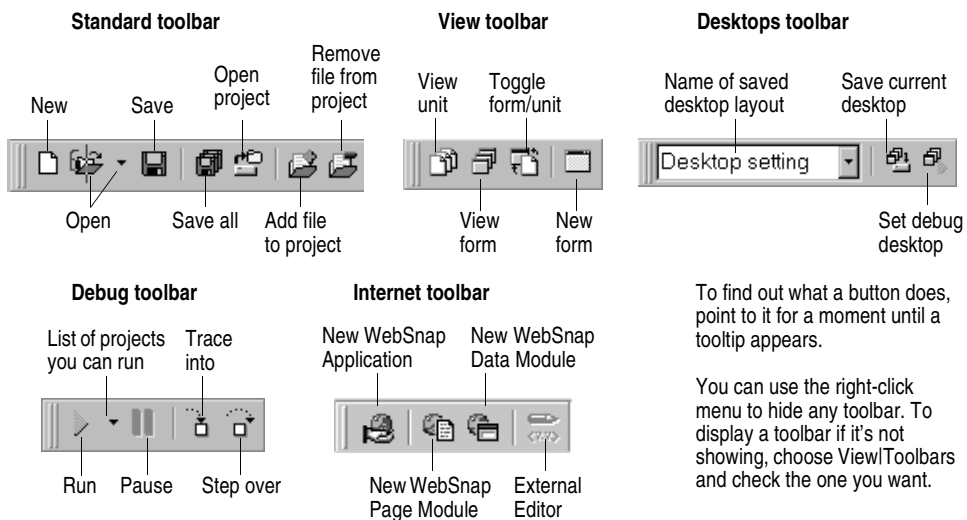
To learn about organizing and configuring the IDE, see Chapter 6, "Customizing the desktop."

The menus and toolbars

The main window, which occupies the top of the screen, contains the main menu, toolbars, and Component palette. C++Builder's toolbars provide quick access to frequently used operations and commands. Most toolbar operations are duplicated in the drop-down menus.



Main window in its default arrangement.



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 6-1 and “Saving desktop layouts” on page 6-4.

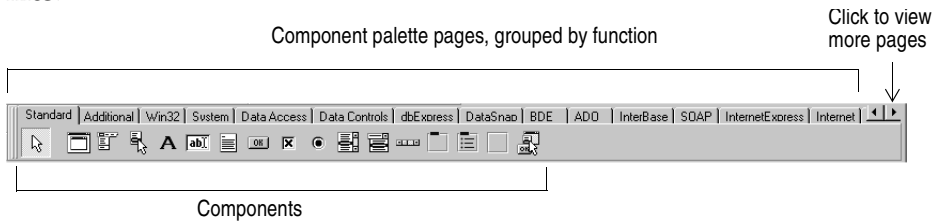
For more information...

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.

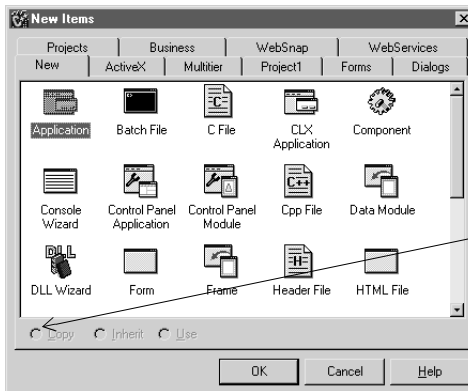
The Object TreeView is especially useful for displaying the relationships between database objects.

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.

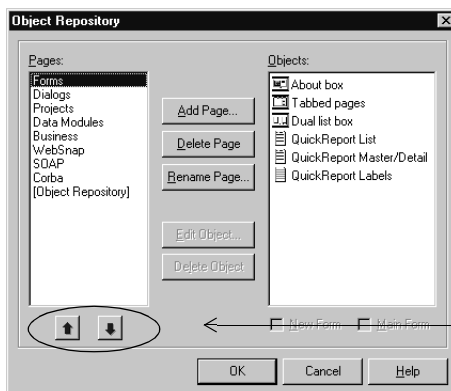


The Repository's tabbed pages include objects like forms, frames, units, and wizards to create specialized items.

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.

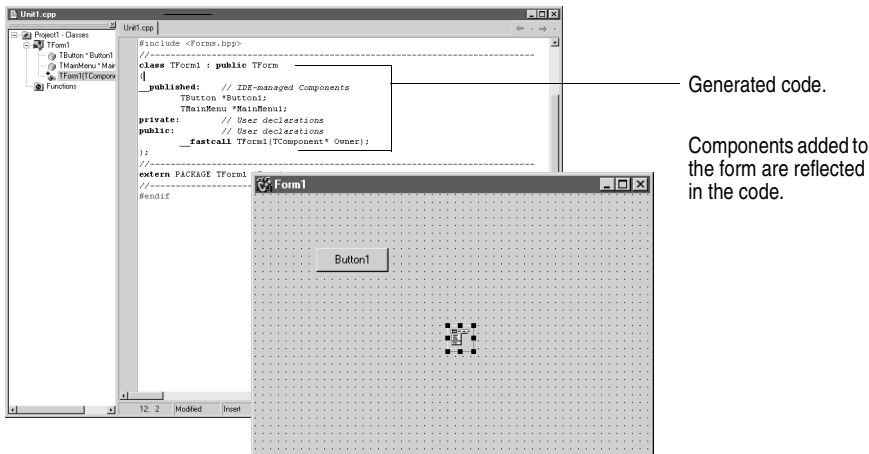
To add project and form templates to the Object Repository, see "Adding templates to the Object Repository" on page 6-9.

For more information...

See “Object Repository” in the online Help index. The objects available to you will depend on which edition of C++Builder you purchased.

The Code Editor

As you design the user interface for your application, C++Builder generates the underlying Object 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. C++Builder 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.

Tool	How it works
Code completion	Type the name of a variable that represents a pointer to an object followed by an arrow (->) or that represents a non-VCL object followed by a dot. 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 C++Builder.

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.

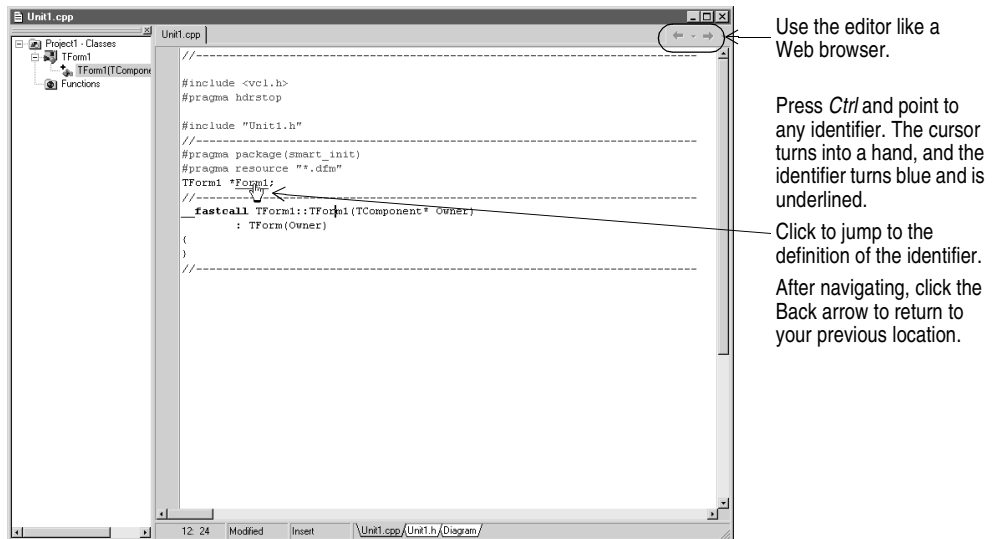
For more information...

See “Code Insight” 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.



To customize your code editing environment, see “Customizing the Code Editor” on page 6-11.

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 C++Builder 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.

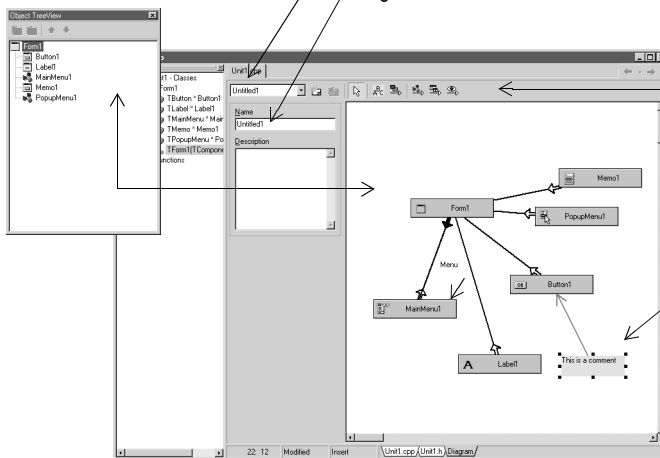
To create a diagram, click the Diagram page. From the Object TreeView, simply drag one or multiple icons to the Diagram page to arrange them vertically. To arrange them horizontally, press *Shift* while dragging. When you drag icons with parent-children or component-property dependencies onto the page, 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.

From the Object TreeView, drag the icons of the components to the Diagram page.

To view other diagrams you've named in the current project, click the drop-down list box.

Type a name and description for your diagram.



Use the Diagram page toolbar buttons—Property, Master/Detail and Lookup—to designate the relationship between components and components and their properties. The appearance of the connecting line varies for each type of relationship.

Click the Comment block button to add a comment, and the Allude connector button to draw a connection to another comment or 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 C++Builder projects—they are where you design the user interface of an application. Normally, you design forms using C++Builder’s visual tools, and C++Builder 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.

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 ClassExplorer

When you open C++Builder, the ClassExplorer is docked to the left of the Code editor window, depending on whether the Code Explorer is available in the edition of C++Builder you have. The ClassExplorer 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.

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

To configure how the Code Explorer displays its contents, choose Tools | Environment Options and click the Explorer tab.

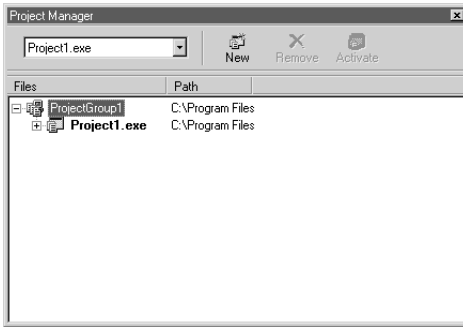
For more information...

See “ClassExplorer” in the online Help index.

The Project Manager

When you first start C++Builder, it automatically opens a new project. 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.



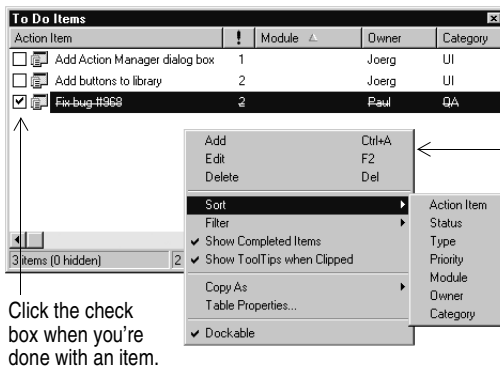
You can use the Project Manager to combine and display information on related projects into a single *project group*. By organizing related projects into a group, such as multiple executables, you can compile them at the same time. To change project options, such as compiling a project, see “Setting project options” on page 6-8.

For more information...

See “Project Manager” 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.



Right-click on a to-do list to display commands that let you sort and filter the list.

Click the check box when you're done with an item.

For more information...

See “to-do lists” in the online Help index.

Programming with C++Builder

The following sections provide an overview of software development with C++Builder, 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 C++Builder, 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 C++Builder, see “Types of projects” on page 3-9.

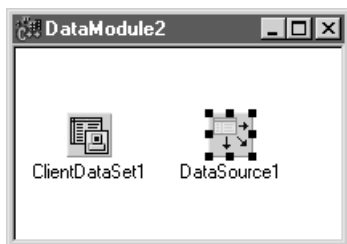
For more information...

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. C++Builder 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, C++Builder displays its components.

For more information...

See “data modules” in the online Help index.

Building the user interface

With C++Builder, 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.

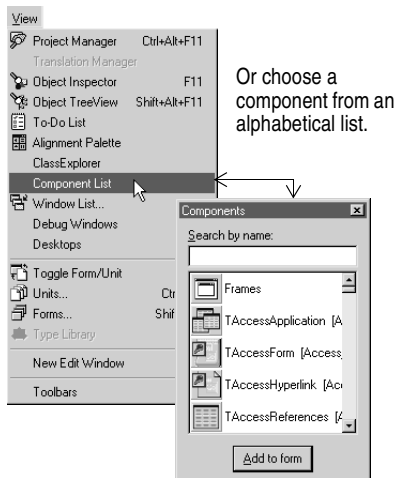
Select the component and drag it to wherever you want on the form.

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

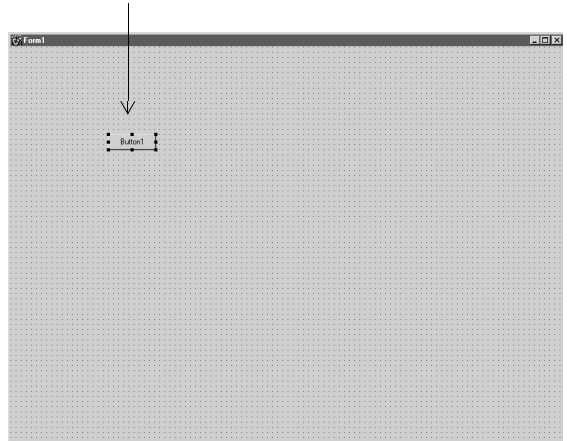


Click a component on the Component palette.

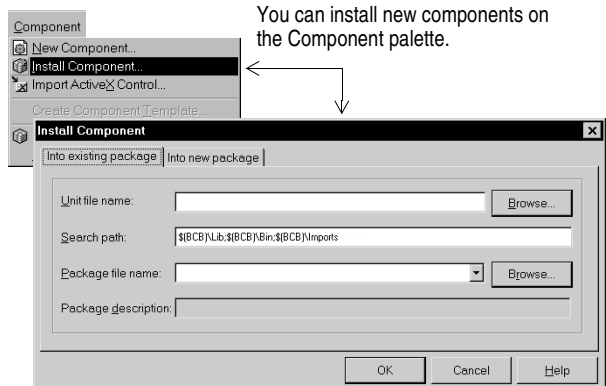
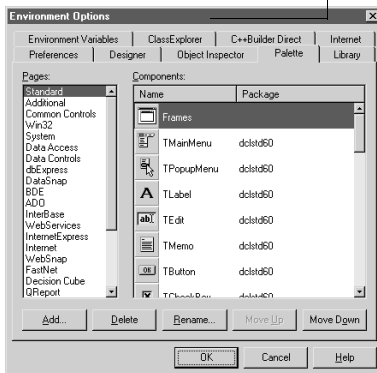
Then click where you want to place it on the form.



Or choose a component from an alphabetical list.



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



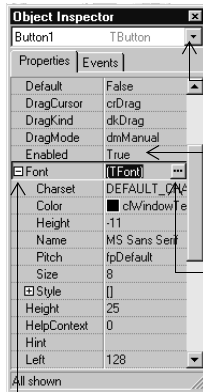
You can install new components on the Component palette.

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.

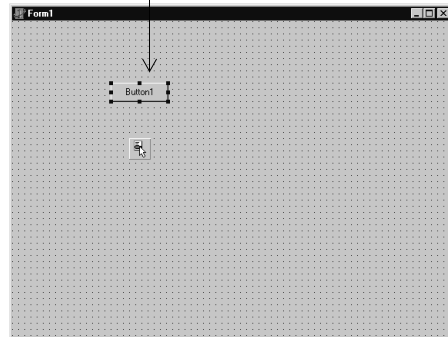


Or use this drop-down list to select an object. Here, Button1 is selected, and its properties are displayed.

You can select a component, or object, on the form by clicking on it.

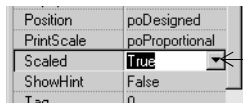
Select a property and change its value in the right column.

Click an ellipsis to open a dialog box where you can change the properties of a helper object.

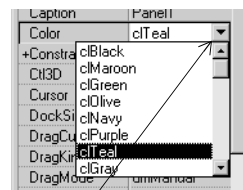


You can also click a plus sign to open a detail list.

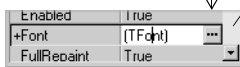
Many properties have simple values—such as names of colors, *True* or *False*, and integers. For Boolean properties, you can double-click the word to toggle between *True* and *False*. Some properties have associated property editors to set more complex values. When you click on such a property value, you'll see an ellipsis. For some properties, such as size, enter a value.



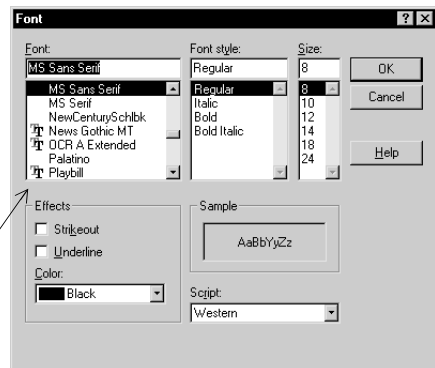
Double-click here to change the value from *True* to *False*.



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



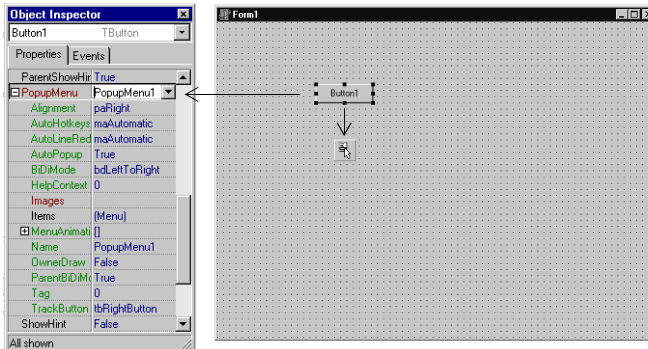
Click on the down arrow to select from a list of valid values.



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.

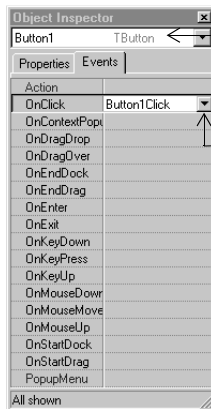
Writing code

An integral part of any application is the code behind each component. While C++Builder'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 C++Builder'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.

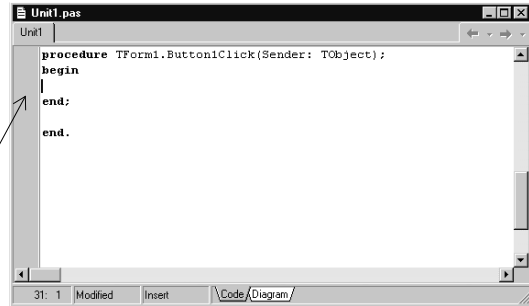
To view predefined event handlers for a component on your form, select the component and, on the Object Inspector, click the Events tab.



Here, Button1 is selected and its type is displayed: *TButton*. Click the Events tab in the Object Inspector to see the events that the Button component can handle.

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

Or double-click in the value column, and C++Builder generates skeleton code for the new event handler.

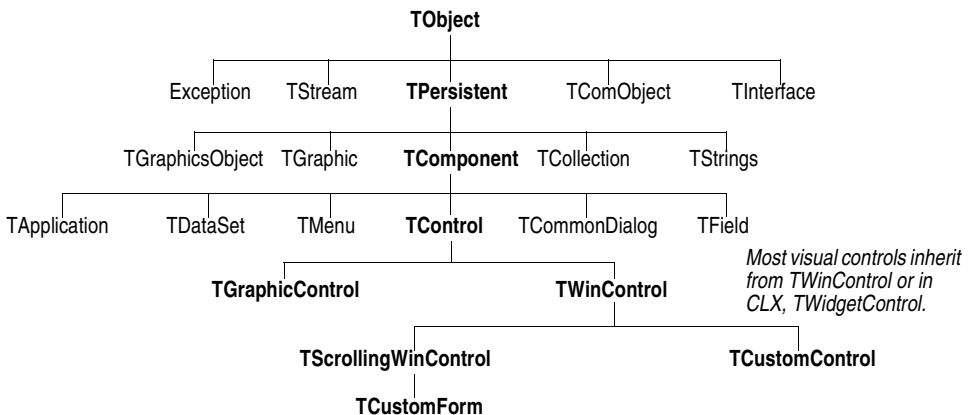


For more information...

See “events” in the online Help index.

Using the VCL and CLX libraries

C++Builder 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 Windows and 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 C++Builder 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 C++Builder come with VCL and CLX source code.

For more information...

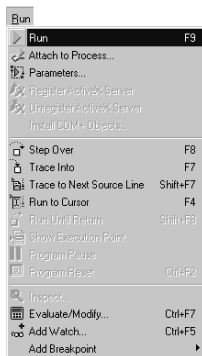
See "Visual Component Library Reference" and "CLX Reference" in the Help contents and "VCL" in the online Help index.

Compiling and debugging projects

After you have written your code, you will need to compile and debug your project. With C++Builder, 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.

C++Builder 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**.



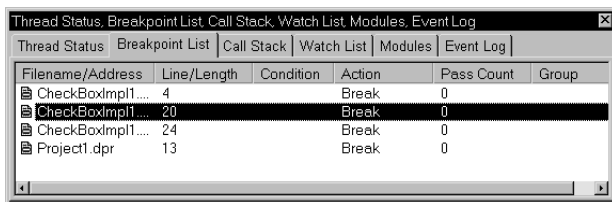
Choose any of the debugging commands from the **Run** menu. Some commands are also available on the toolbar.



Run button

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 C++Builder.



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 6-2.

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 6-4.

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. C++Builder 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 C++Builder setup screen, choose InstallShield Express Custom Edition for C++Builder.

For more information...

See “deploying, applications” in the online Help index.

Internationalizing applications

C++Builder 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. C++Builder includes a translation suite, not available in all editions of C++Builder, 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 C++Builder 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.

CLX applications

You can use C++Builder, to develop cross-platform 32-bit applications that run on both the Windows and Linux operating systems. For Linux, a Borland C++ solution is not yet available, but you can prepare ahead of time by developing the application with C++Builder today. To develop a CLX application, choose File | New | CLX Application. The IDE is similar to that of a regular C++Builder 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 C++Builder 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.

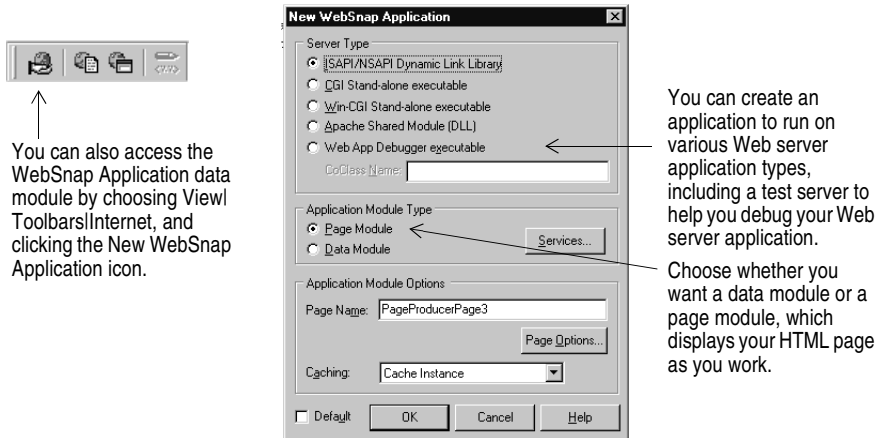
Web server applications

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, C++Builder includes two different technologies, depending on what edition of C++Builder you have.

C++Builder’s oldest Web server application technology is called Web Broker. Web Broker applications can dispatch requests, perform actions, and return Web pages to users. Most of the business logic of an application is defined in event handlers written by the application developer. 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. These extra features are designed to handle common Web server application tasks automatically. WebSnap development is more visual and simple than Web Broker development. A WebSnap application developer can spend more time designing the business logic of an application, and less time writing event handlers for common page transfer tasks. 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.



For more information...

See "Web applications" in the online Help index.

Database applications

C++Builder 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 C++Builder, 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 C++Builder 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 C++Builder's library of data-aware components.

Certain database connectivity tools are not available in all editions of C++Builder.

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 C++Builder | 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 C++Builder 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 C++Builder | 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 | C++Builder Tools to see “Data Dictionary.”

Custom components

The components that come with C++Builder 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 C++Builder 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 6-6.

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

C++Builder 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.

C++Builder provides a Type Library editor for creating and maintaining type libraries.

For more information...

See “type libraries” in the online Help index.

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 C++Builder.

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\CBuilder6\Projects` directory.
- 2 Begin a new project by choosing `File | New | Application` or use the default project that is already open when you started C++Builder.

Each application is represented by a *project*. When you start C++Builder, it creates a blank project by default, and automatically creates the following files:

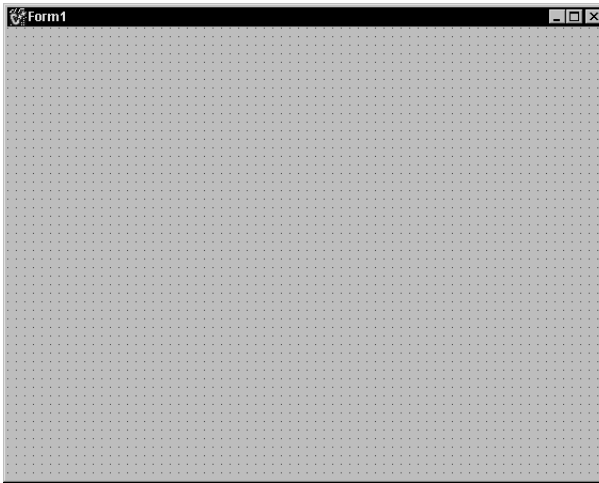
- *Project1.cpp*: a source-code file associated with the project. This is called a *project file*.
- *Unit1.cpp*: a source-code file associated with the main project form. This is called a *unit file*.
- *Unit1.h*: a header file associated with the main project form. This is called a *unit header file*.
- *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.cpp*), header (*Unit1.h*), and form (*Unit1.dfm*) files. If you create a second form, a second unit (*Unit2.cpp*), header (*Unit2.h*), and form (*Unit2.dfm*) file are automatically created.

- 3 Choose File | Save All to save your files to disk. When the Save As dialog box appears:
 - Navigate to your TextEditor folder.
 - Save Unit1 using the default name Unit1.cpp.
 - 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, C++Builder creates various additional files in your project directory. Do not delete these files.



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.

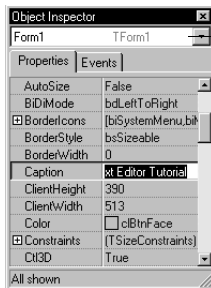
To return to design mode, click the **X** to close the form.

Setting property values

When you open a new project, C++Builder 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, C++Builder 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.
- 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.

C++Builder 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 a toolbar with buttons for easy access to commands. The beauty of designing the interface using C++Builder 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 text area and a status bar to the form.



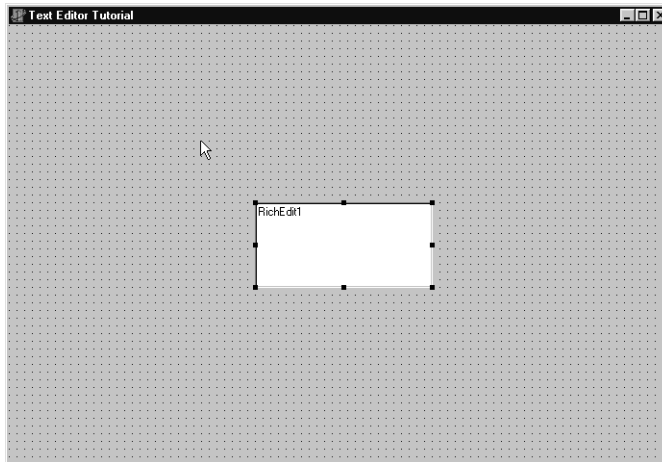
- 1 To create a text area, first add a *RichEdit* component. To find the *RichEdit* component, on the Win32 page of the Component palette, point to an icon on the

palette for a moment; C++Builder displays a Help tooltip showing the name of the component.



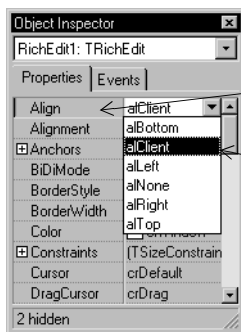
When you find the *RichEdit* component, either:

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



Each C++Builder component is a *class*; placing a component on a form creates an *instance* of that class. Once the component is on the form, C++Builder 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`.



Make sure the *RichEdit1* component is selected on the form.

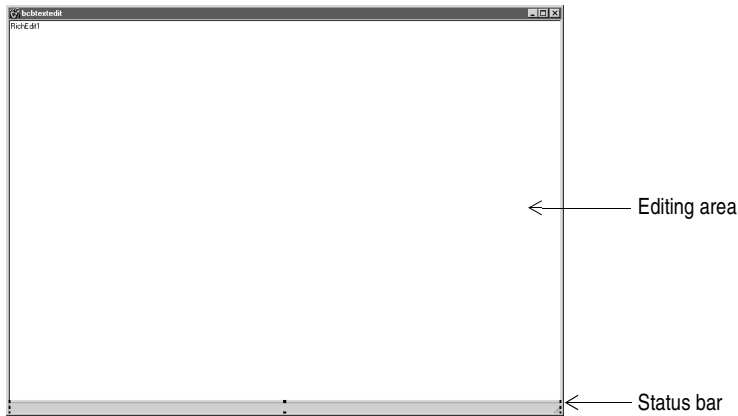
Look for the *Align* property in the Object Inspector. Click the down arrow to display the property's drop-down list.

Select `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 to add a status bar to the bottom of the form.

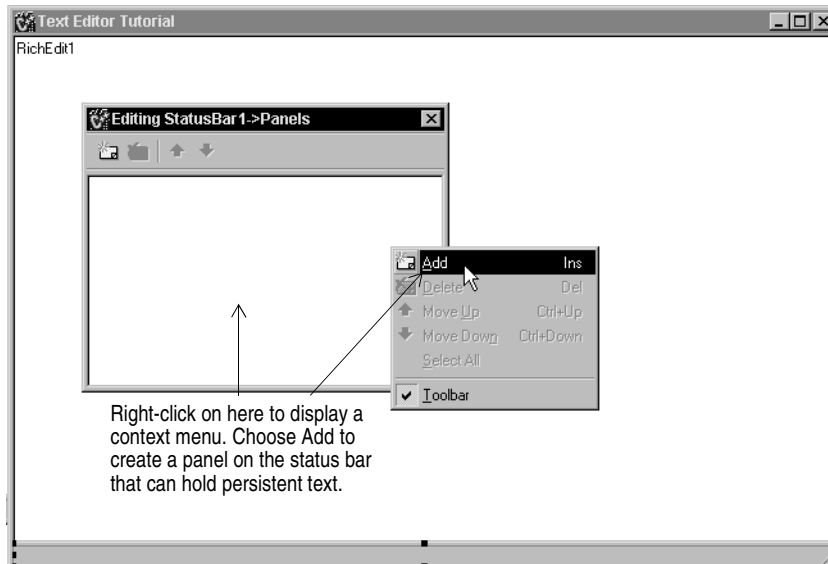


- 4 Double-click the status bar to open the Editing StatusBar1->Panels dialog box.

Tip

You can also open the Editing StatusBar1->Panels dialog box by clicking the (*TStatusBar*) ellipse of the status bar's *Panels* property.

- 5 Click the Add New button  on the toolbar of the dialog box, or right-click and choose *Add*, to add a panel to the status bar. The panel will display the path and file name of the file being edited by your text editor.



- 6 In the Object Inspector, set the *Text* property to `untitled.txt`. When you use the text editor, if the file being edited is not yet saved, the file name will be `untitled.txt`.
- 7 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, C++Builder provides an *action list* or an *action manager* to centralize the actions, images, and code for menu commands and toolbar buttons.

By convention, the actions that are connected to menu commands have a top-level menu name and a command name. For example, the FileExit action refers to the Exit command on the File menu. The following table lists the kinds of menu commands your text editor application needs and whether the action has an associated toolbar button:

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.

Action Manager editor and Action List editor differences

Depending on your edition of C++Builder, there are two ways to manage actions and images for your menus and toolbar. All editions of C++Builder include the *Action List editor*, which provides a location to centralize the response to user commands. The *Action List editor* is part of the Borland Component Library for Cross Platform (CLX) and should be used instead of the *Action Manager editor* if migrating to a different platform (such as Linux) is a future possibility.

The *Action Manager editor* provides some special functionality, but is only available as part of the Visual Component Library (VCL), which is specific to the Windows platform. Using the *Action Manager editor Customize dialog* can provide menu actions that are customizable by the end user, and that have some of the properties of Microsoft Office (such as having seldomly used menu items hidden from view). Additionally, the *Action Manager* provides a more rapid development process, as actions can simply be dragged from the *Action Manager Customize dialog* to the menu component on the form.

Warning This tutorial uses the *Action Manager Customize dialog* for Enterprise and Professional editions of C++Builder. If you have the **Enterprise** or **Professional** edition, proceed to “Adding menu and toolbar images (Enterprise and Professional)” below.

If you have the Personal edition or want to use the Action List editor, skip to “Adding an image list and images (Personal edition)” on page 4-13.

Adding menu and toolbar images (Enterprise and Professional)

In this section, you’ll add images for use with Action Bands.

In many cases you would add an *ImageList1* component to your form and import your own images. For this tutorial we will save time by importing the image list that was used to create the C++Builder IDE. Unless you add your own graphics, the *ImageList1* on the component palette will use default images for standard actions.

To add the existing image list:

- 1 If you installed C++Builder to the default directory, choose File | Open and select C:\Program Files\Borland\CBuilder6\Source\vcl\actnres.pas. To see this file, set Files of type: to (Any file. *.*) in the Open dialog.



Note

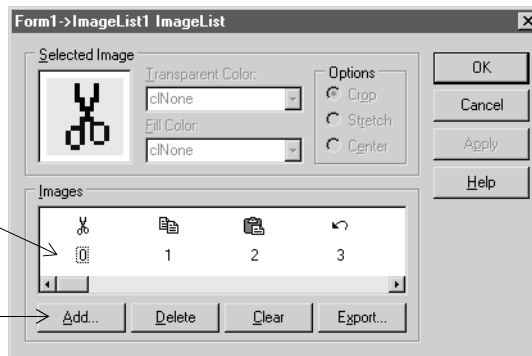
- 2 Select the *ImageList1* component and copy and paste it to your form. It is a nonvisual component, so it doesn’t matter where you paste it.

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 the *ImageList1* component to display all the possible images you can use.

The numbers underneath the images correspond to the image index property for each action.

You can click the Add button to add images from another source.



The image index properties for both standard actions and the *ImageList1* we have added include:

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. See “Adding an image list and images (Personal edition)” on page 4-13. Click **OK** to close the *ImageList1* dialog.

Adding actions to the Action Manager (Enterprise and Professional)

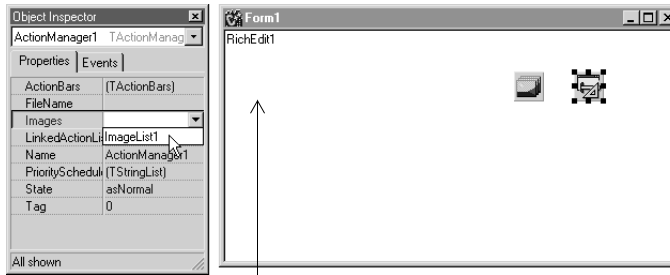
The Enterprise and Professional editions provide an Action Manager that makes it easy to add actions to menus and toolbars. You first add the Action Manager and then add the actions.



- 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.

Tip To display the captions for nonvisual components you drop on the form, choose Tools | Environment Options, click the Designer page, check Show component captions, and click OK. Also, hovering over a component with the mouse will display its name.

- 2 With *ActionManager1* selected on the form, set the Images property in the Object Inspector to *ImageList1*.



Click on the Images property, then on the down arrow next to Images. *ImageList1* is listed for you. Select it. This associates the images in the image list with the actions in the action list.

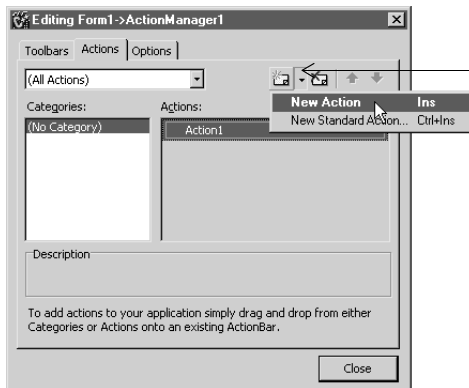
Next you'll add the actions to the Action Manager and set their properties. You will add both nonstandard actions for which you set all the properties, and standard actions, which have their properties automatically set.

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

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

- 4 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 on the Action Manager editor and choose New Action.



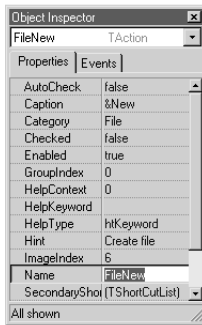
Click the drop-down arrow next to the New Action button to create new actions for the Action Manager.

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

- 5 Make sure *No Category* is selected, in the Actions list and *Action1* is selected under *Actions*. 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).

- Make sure the *ImageIndex* is set to 6 (This should match the image list we imported. You can also click the down arrow and select the proper image).
- 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.

- 6 Make sure File is selected in the Editing Form1->ActionManager1 window. Click the drop-down arrow next to the *New Action* button and select *New Action*.
- 7 In the Object Inspector, set the following properties:
 - After *Caption*, type &Save.
 - Make sure the *Category* is set to File.
 - After *Hint*, type Save file.
 - After *ImageIndex*, select image 8.
 - After *Name*, enter FileSave (for the File | Save command).
- 8 Click the drop-down arrow next to the New Action button and click New Action.
- 9 In the Object Inspector, set the following properties:
 - After *Caption*, type &Index.
 - After *Category*, type Help.
 - No *ImageIndex* is needed. Leave the default value.
 - After *Name*, enter HelpIndex (for the Help | Index command).
- 10 Click the drop-down arrow next to the New Action button and click New Action.
- 11 In the Object Inspector, set the following properties:
 - After *Caption*, type &About.
 - Make sure *Category* says Help.
 - No *ImageIndex* is needed. Leave the default value.
 - After *Name*, enter HelpAbout (for the Help | About command).
- 12 Keep the *Action Manager Customize* dialog on the screen.
- 13 Save your work by clicking File | Save All.

Adding standard actions (Enterprise and Professional)

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

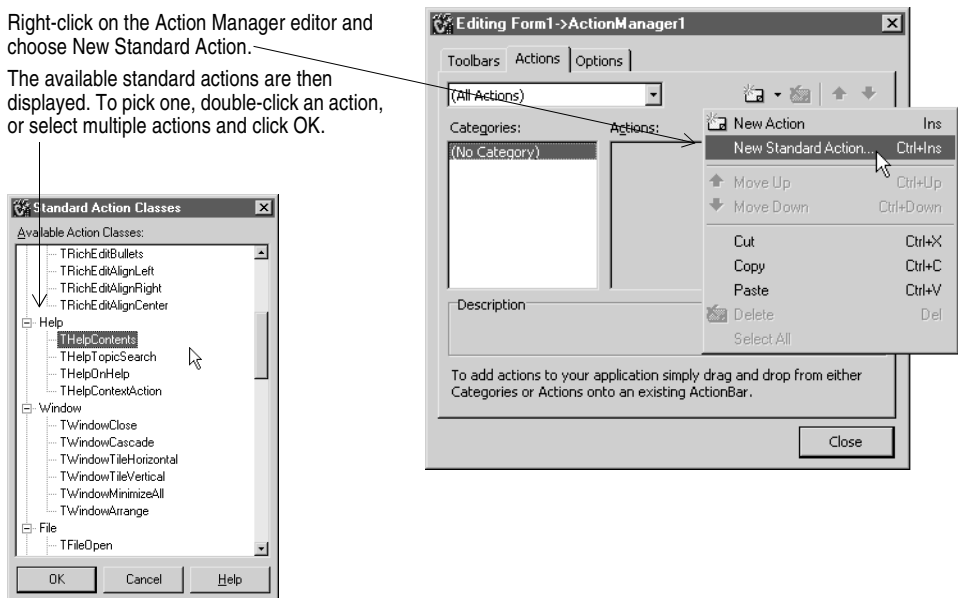
- 1 The Action Manager editor should still be displayed. If it's not, double-click the *ActionManager* component to open it.
- 2 Click the drop-down arrow next to the New Action button and click New Standard Action.

The Standard Action Classes dialog box appears.

- 3 Scroll to the Edit category and use the *Ctrl* key to select *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.

Right-click on the Action Manager editor and choose New Standard Action.

The available standard actions are then displayed. To pick one, double-click an action, or select multiple actions and click OK.



- 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 *TFileOpen*, *TFileSaveAs*, and *TFileExit*. 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 Scroll to the Help category and select *THelpContents*. Click OK to add this action to the Help category.

Note

The custom Help | Contents command displays a Help file always showing the 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. You can change the image index to display a different image.

- 8 You can change the image for standard actions if you prefer. For instance, select the File|Open action in the Editing Form1->ActionManager1 dialog box. Now, change the default image to image 7 in the list.
- 9 Click the Close button to close the Action Manager editor.
- 10 Click File|Save All to save your changes.

Adding a menu (Enterprise and Professional)

In the next two sections, you'll add customizable menu bar and tool action bands. The text editor menu bar includes three drop-down menus—File, Edit, and Help—and their menu commands. With the Action Manager Customize dialog, 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 Customize dialog 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+↓*.
- 3 Select the Open action and click the Move Up button on the Action Manager Customize dialog toolbar, so that the File commands are listed in the following order: New, Open, Save, Save As, and Exit.
- 4 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, you can click File on the menu bar so its submenu commands appear, and drag Open above New and then back again.

- 5 From the Categories list of the Action Manager Customize dialog, drag Edit to the right of File on the menu bar.
- 6 From the Categories list of the Action Manager Customize dialog, drag Help to the right of the Edit on the menu bar.
- 7 Click the Help menu to view its submenu commands. Drag the Contents command to above the Index command.
- 8 Press *Esc* or click the Help menu again to close it.
- 9 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 (Enterprise and Professional)

Since you've set up actions in the Action Manager Customize dialog, 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 Action Band toolbar appears under the menu bar.

Tip

You can also add an Action Band toolbar by opening the Action Manager Customize dialog, 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. 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.

- 3 In the Action Manager Customize dialog, drag the Edit category to the toolbar. All of the Edit commands should appear on the toolbar.

Note

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 *Del* key. You can reposition the buttons simply by dragging them to the left or right of each other.

- 4 Choose File | Save All to save your changes.

- 5 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, C++Builder opens the program in a runtime window like the one you designed on the form.

Your text editor already has lots of functionality. If you select text in the text area, the Cut, Copy, and Paste buttons should work. The menus and toolbar buttons work although some of the commands are grayed out. To activate some of the commands you will need to write event handlers.

- 6 To return to design mode, click **X** in the upper right corner of the application.

To continue the tutorial, skip to "Clearing the text area (all editions)" on page 4-22.

Adding an image list and images (Personal edition)

In this section, you'll add an *ImageList* component to your form and images to that list. Select the ImageList component on the Component palette and click on form to add an *ImageList1* component.

The images to use for each command include:

Command	Image name	ImageIndex property
File New	Filenew.bmp	0
File Open	Fileopen.bmp	1
File Save	Filesave.bmp	2
File Exit	Doorshut.bmp	3
Edit Cut	Cut.bmp	4
Edit Copy	Copy.bmp	5
Edit Paste	Paste.bmp	6
Help Contents	Help.bmp	7

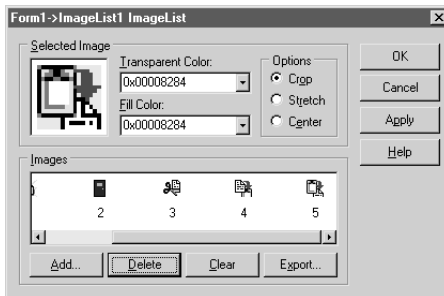
Note Without importing graphics, the *ImageList* uses default images for standard actions.

To add images to the image list:

- 1 Double-click the *ImageList* component on the form to display the Image List editor.
- 2 Click the Add button.
- 3 In the Add Images dialog box, navigate to the Buttons directory provided with the product. The default location is C:\Program Files\Common Files\Borland Shared\Images\Buttons.
- 4 Double-click filenew.bmp.

When a message asks if you want to separate the bitmap into two separate ones, click Yes each time. Each of the icons includes an active and a grayed out version of the image. You'll see both images. Delete the grayed out (second) image.

- 5 Add the rest of the images:
 - Click Add. Double-click fileopen.bmp. Delete the grayed out image.
 - Click Add. Double-click filesave.bmp. Delete the grayed out image.
 - Click Add. Double-click doorshut.bmp. Delete the grayed out image.
 - Click Add. Double-click cut.bmp. Delete the grayed out image.
 - Click Add. Double-click copy.bmp. Delete the grayed out image.
 - Click Add. Double-click paste.bmp. Delete the grayed out image.



Tip You can use the Control key when clicking on the images to select multiple images. Then go back and delete the greyed out images.

Click OK to close the Image List editor.

You've added seven images to the image list and they're numbered 0-6 consistent with the *ImageIndex* properties for each of the actions.

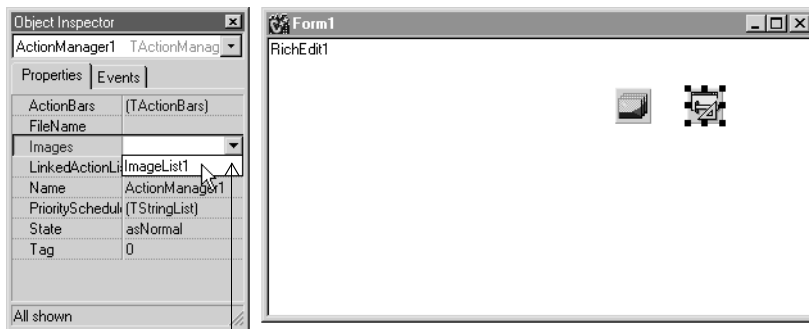
Note If you get them out of order, you can drag and drop them into their correct positions in the Image List editor.

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

Adding actions to the action list (Personal edition)

You have already added an *ImageList* component to the form and added images to the *ImageList*. In this section you will add an action list and actions.

- 1 Double-click the *ActionList* component in the Standard tab of the Component palette. Since the *ActionList1* that is added to the form is a nonvisual component it doesn't matter where you place it on the form.
- 2 With the *ActionList* component still selected on the form, set its *Images* property to *ImageList1*.

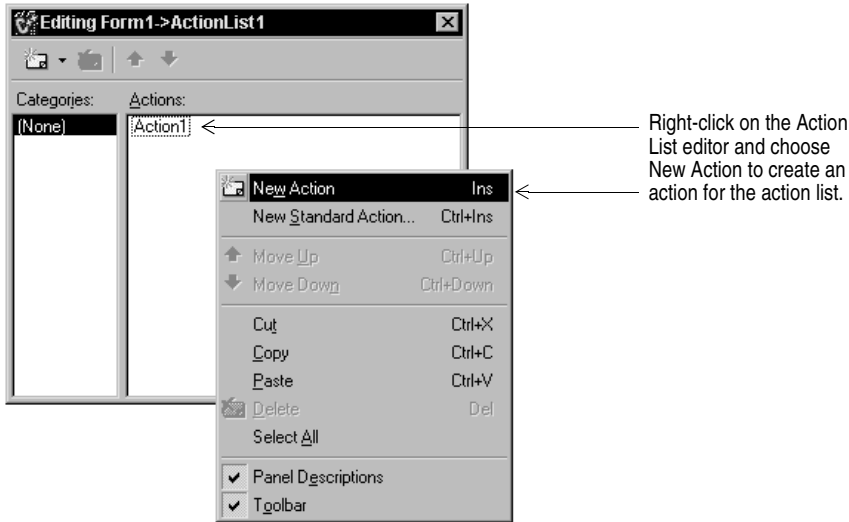


Click on the Images property, then on the down arrow next to Images. *ImageList1* is listed for you. Select it. This associates the images that we'll add to the image list with the actions in the action list.

- 3 Double-click the *ActionList* component to open it.

The Editing Form1->ActionList1 dialog box appears. This is also called the *Action List editor*.

4 Right-click on the Action List editor and choose New Action.



Tip You can also click the drop-down arrow next to the New Action button and click New Action.

5 For 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*, select the related image (image 0 if you added your image list in the order described above).
- After *Name*, type `FileNew` (for the File | New command) and press *Enter* to save the change.

6 Right-click on the Action List editor and choose New Action.

7 For Action1, in the Object Inspector, set the following properties:

- After *Caption*, type `&Save`.
- Make sure *Category* says `File`.
- After *Hint*, type `Save file`.
- After *ImageIndex*, select the related image (image 2 if you added your images in the order described above).
- After *Name*, enter `FileSave` (for the File | Save command).

8 Right-click on the Action List editor and choose New Action.

9 For Action1, in the Object Inspector, set the following properties:

- After *Caption*, type `&Index`.
- After *Category*, type `Help`.
- No *ImageIndex* is needed. Leave the default value.
- After *Name*, enter `HelpIndex` (for the Help | Index command).

10 Right-click on the Action List editor and choose New Action.

11 For Action1, in the Object Inspector, set the following properties:

- After *Caption*, type &About.
- After *Category*, type Help.
- No *ImageIndex* is needed. Leave the default value.
- After *Name*, enter HelpAbout (for the Help | About command).

Keep the Action List editor on the screen.

Adding standard actions to the action list (Personal edition)

C++Builder provides several standard actions that are often used when developing applications. Next you'll add these standard actions, such as cut, copy, and paste, to the action list.

1 The Action List editor should still be displayed. If it's not, double-click the *ActionList* component on the form.

2 Right-click the Action List editor and click New Standard Action.

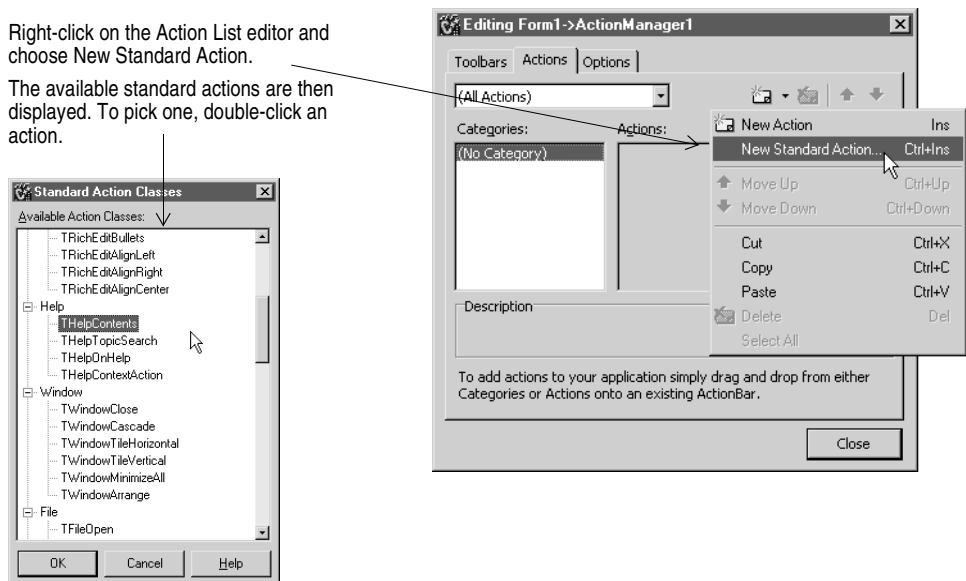
Tip

You can also click the drop-down arrow next to the New Action button and click New Standard Action.

3 In the Standard Action Classes dialog box, scroll to the Edit category, and use the *Ctrl* key to select *TEditCut*, *TEditCopy*, and *TEditPaste*. Click OK to add these actions to a new Edit category in the Action List editor.

Right-click on the Action List editor and choose New Standard Action.

The available standard actions are then displayed. To pick one, double-click an action.



4 Right-click the Action List editor and click New Standard Action.

5 Scroll to the File category and select *TFileOpen*, *TFileSaveAs*, and *TFileExit*. Click OK to add these actions to the File category.

- 6 Right-click the Action List editor and click New Standard Action.
- 7 Scroll to the Help category and select *THelpContents*. Click OK to add this action to the Help category.

Note

The custom Help | Contents command displays a Help file always showing the Help Contents tab. The standard Help | Contents command brings up the last tabbed page that was displayed, either Contents, Index, or Find.

The standard actions have their properties set automatically. However, you need to change the image index property to associate the actions with the correct images that come with the Personal edition.

- 8 In the Action List editor's Categories list, choose (All Actions).
- 9 Standard Actions include default images. Now, change the default images to the images you added earlier. In the Actions list, select the following actions one at a time and change their *ImageIndex* property in the Object Inspector:
 - Select EditCut1 and set its *ImageIndex* property to 4.
 - Select EditCopy1 and set its *ImageIndex* property to 5.
 - Select EditPaste1 and set its *ImageIndex* property to 6.
 - Select FileOpen1 and set its *ImageIndex* property to 1.
 - Select FileExit1 and set its *ImageIndex* property to 3.
- 10 Click the **X** to close the Action List editor.
- 11 Choose File | Save All to save your changes.

Adding a menu (Personal edition)

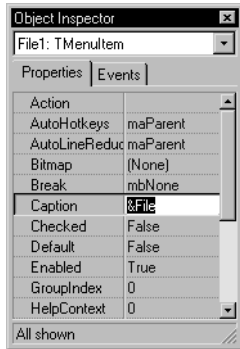
In this section, you'll add a main menu bar with three drop-down menus—File, Edit, and Help—and you'll add menu items to each one using the actions in the action list.



- 1 From the Standard tab 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* so you can add the images to the menu commands.
- 3 Double-click the *MainMenu* component to open the Menu Designer.



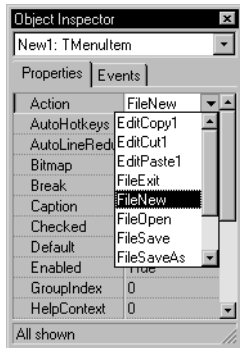
- 4 To set the first top-level menu item in the Menu Designer, in the Object Inspector, set the *Caption* property to `&File` and press *Enter*.



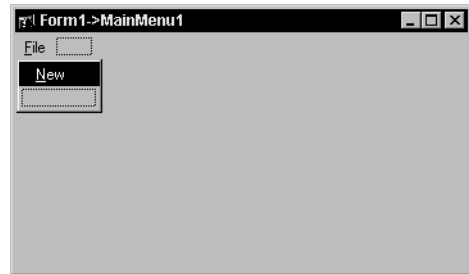
When you type `&File` and focus on the Menu Designer, the top-level File command appears ready for you to add the first menu item.



- 5 Select the empty item below the File command you just created.
- 6 In the Object Inspector, set the *Action* property to `FileNew`. All actions from the action list appear there.

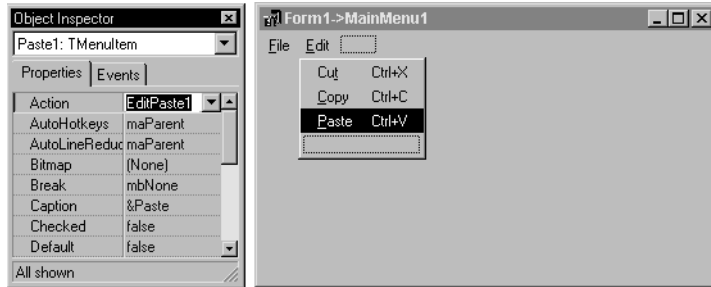


When you select `FileNew` from the Action property list, the New command appears with the correct Caption and ImageIndex.



- Select the item below New and set its *Action* property to `FileOpen1`.
 - Select the item below Open and set its *Action* property to `FileSave`.
 - Select the item below Save and set its *Action* property to `FileSaveAs1`.
 - Select the item below Save As, type a hyphen after its *Caption* property, and press *Enter*. This creates a separator bar on the menu.
 - Select the item below the separator bar and set its *Action* property to `FileExit1`.
- 7 Next create the Edit menu:
- Select the item to the right of the File command, set its *Caption* property to `&Edit`, and press *Enter*.
 - The focus is now on the item under Edit; set its *Action* property to `EditCut1`.
 - Select the item below Cut and set its *Action* property to `EditCopy1`.

- Select the item below Copy and set its *Action* property to `EditPaste1`.



8 Create a Help menu:

- Select the item to the right of the Edit command, set its *Caption* property to `&Help`, and press *Enter*.
- Select the item below Help and set its *Action* property to `HelpContents`.
- Select the item below Contents and set its *Action* property to `HelpIndex`.
- Select the item below Index, type a hyphen after its *Caption* property, and press *Enter* to create a separator bar on the Help menu.
- Select the item below the separator bar and set its *Action* property to `HelpAbout`.

9 Click the **X** to close the Menu Designer.

10 Choose File | Save All.

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

Note If you lose the form, click View | Forms, select Form1, and click OK.

Adding a toolbar (Personal edition)

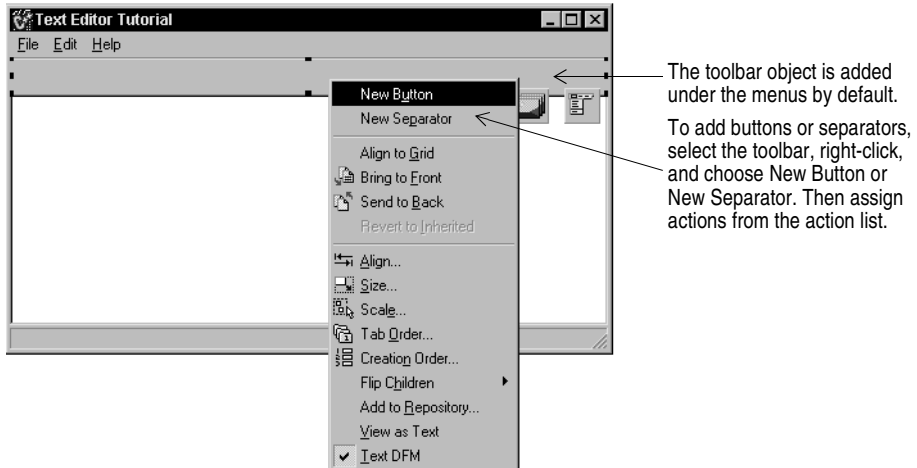
Since you've set up actions in an action list, you can add some of the same actions that were used on the menus onto a toolbar.



- 1 On the Win32 tab of the Component palette, double-click the *ToolBar* component to add it to the form. A blank toolbar is added under the main menu.
- 2 With the toolbar selected, change the following properties in the Object Inspector:
 - Set the *Images* property to `ImageList1`.
 - Set the toolbar's *Indent* property to 4. (This indents the icons four pixels from the left of the toolbar.)
 - Set *ShowHint* to *true*. (**Tip:** Double-click *false* to change it to *true*.)

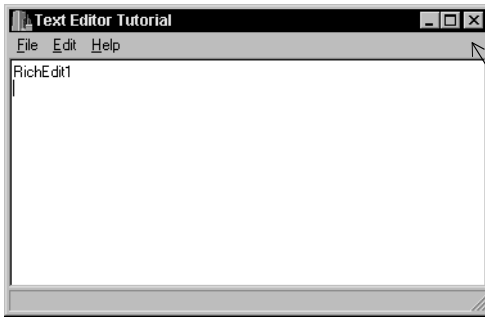
- 3 To add buttons to the toolbar, right-click and choose New Button four times.
- 4 To add a separator to the toolbar, right-click and choose New Separator.
- 5 Right-click and choose New Button three more times.

Note Don't worry if the images aren't correct yet. The correct images will be added when you assign actions to the buttons.



- 6 Assign actions from the action list to the first set of buttons.
 - Select the first button and set its *Action* property to `FileNew`.
 - Select the second button and set its *Action* property to `FileOpen1`.
 - Select the third button and set its *Action* property to `FileSave`.
 - Select the fourth button and set its *Action* property to `FileExit1`.
- 7 Assign actions to the second set of buttons.
 - Select the fifth button and set its *Action* property to `EditCut1`.
 - Select the sixth button and set its *Action* property to `EditCopy1`.
 - Select the last button and set its *Action* property to `EditPaste1`.
- 8 Choose `File | Save All`.
- 9 Press `F9` to compile and run the project.

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



When you press *F9* to run your project, the application interface is displayed. The menus, text area, and status bar all appear on the form.

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

When you run your project, C++Builder opens the program in a window like the one you designed on the runtime form. The menus all work although most of the commands are grayed out. The images appear next to menu items with which you associated an image index.

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.

- 10 Click the **X** in the upper right corner to close the application and return to the design-time view.

Clearing the text area (all editions)

Important The rest of the tutorial is for all editions.

When you ran your program, the name *RichEdit1* appeared in the text area. You can remove that text using the String 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 String List editor.
- 3 In the String List editor, select and delete the text (*RichEdit1*) and click OK.
- 4 Save your changes and run the program again.

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

Writing event handlers

Up to this point, you've developed your application without writing any code. By using the Object Inspector to set property values at design time, you've taken full advantage of C++Builder's RAD environment. In this section, you'll write functions

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 nonstandard actions, you must create event handlers. For 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 can write your own event handler to customize the command.

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

Important If you have the Personal edition of C++Builder, use the *ActionList* component instead of the *ActionManager* component in the following steps.

Creating an event handler for the New command

To create an event handler for the New command:

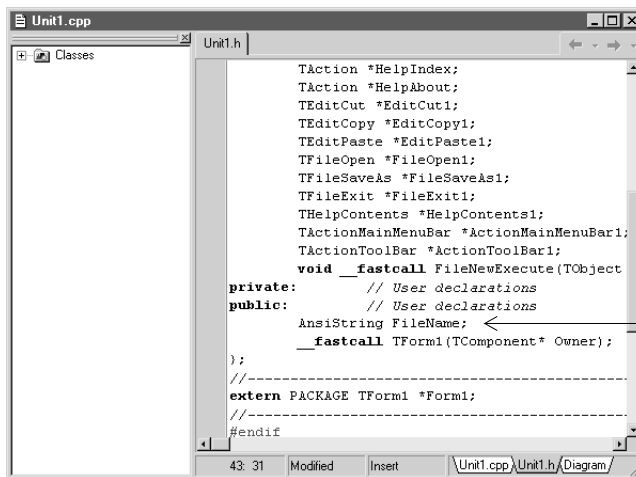
- 1 Choose View | Units and select Unit1 to display the code associated with Form1.
- 2 First, you need to declare a file name that will be used in the event handler, adding a custom property for the file name to make it globally accessible from other methods. Open the Unit1.h file by right-clicking in the Unit1.cpp file in the Code editor and choosing Open Source/Header File (or by clicking on the Unit1.h tab in the Code Editor). In the header file, locate the public declarations section for the class TForm1, and on the line after

```
public:    // User declarations
```

type:

```
AnsiString FileName;
```

Your screen should look like this:



This line defines `FileName` as a string which is globally accessible from other methods.

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

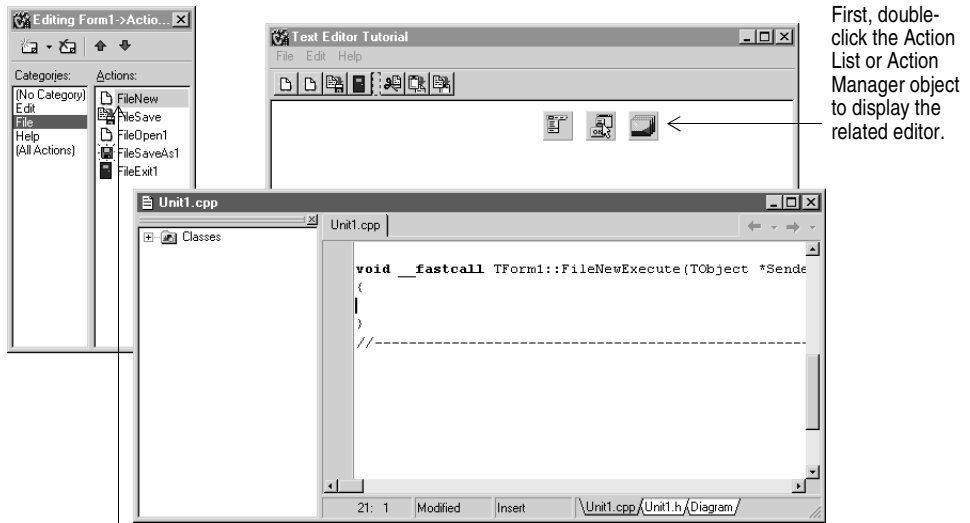
Tip *F12* is a toggle that takes you back and forth from the form to the associated code. You can also choose View | Forms and choose Form1.

4 Double-click the *ActionManager* or *ActionList* component to open it.

5 Double-click the FileNew action.

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

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

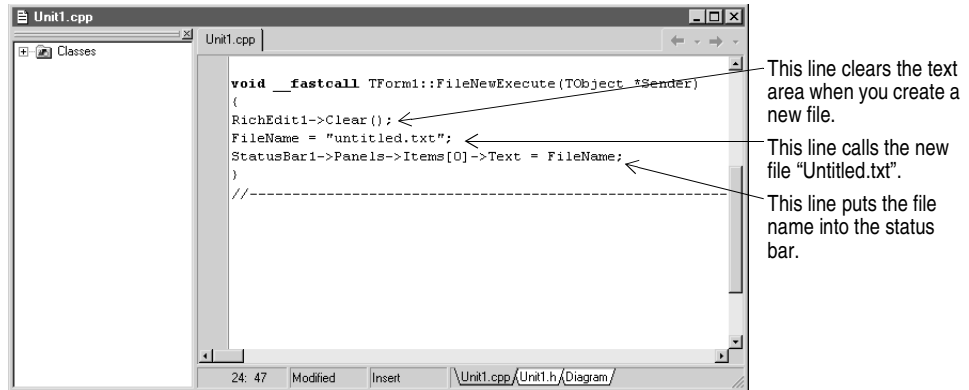


Then, double-click the action to create an empty event handler where you can specify what will happen when users execute the command.

6 Right where the cursor is positioned in the Code editor (between { and }), type the following lines:

```
RichEdit1->Clear();
FileName = "untitled.txt";
StatusBar1->Panels->Items[0]->Text = FileName;
```

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



- 7 Choose File | Save All.

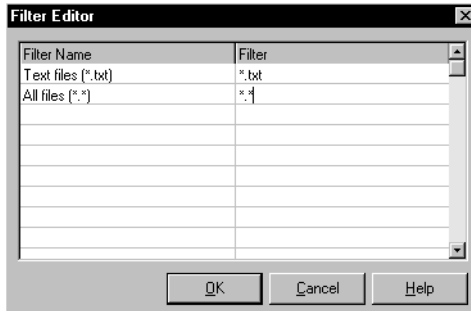
Note You can resize the code portion of the window to reduce horizontal scrolling.

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 or Action List 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 and double-click the *ActionManager* or *ActionList* component to bring it to the front.
- 2 Select the *FileOpen1* action.
- 3 In the Object Inspector, click the plus sign to the left of *Dialog* to expand its properties. *Dialog* is a referenced component that creates the Open dialog box. C++Builder 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 *DefaultExt* property to *txt*.
- 5 Double-click the text area next to *Filter* to display the Filter editor.
 - In the first row of the Filter Name column, type *Text files (*.txt)*. In the Filter column, type **.txt*.
 - In the second row of 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.

- 6 Set *Title* to *Open file*. These words will appear at the top of the Open dialog box.
- 7 Click the Events tab. Double-click the space to the right of the *OnAccept* event so that *FileOpen1Accept* appears.

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

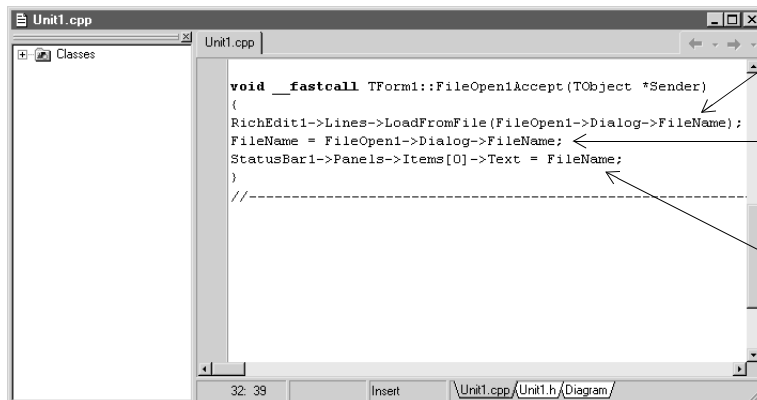
- 8 Right where the cursor is positioned (between { and }), type the following lines:

```
RichEdit1->Lines->LoadFromFile (FileOpen1->Dialog->FileName);
FileName = FileOpen1->Dialog->FileName;
StatusBar1->Panels->Items[0]->Text = FileName;
```

Tip

You can use the Code Insight tools as described on page 2-6 to help you write your code faster. For example, after you type the arrow (->) after *RichEdit1*, the code completion dialog box appears. Type an "l" so that *Lines : TStrings*; appears at the top of the dialog box. Press *Enter* or double-click it to add it to your code.

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 and double-click the *ActionManager* or *ActionList* component.
- 2 Double-click the *FileSave* action.

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

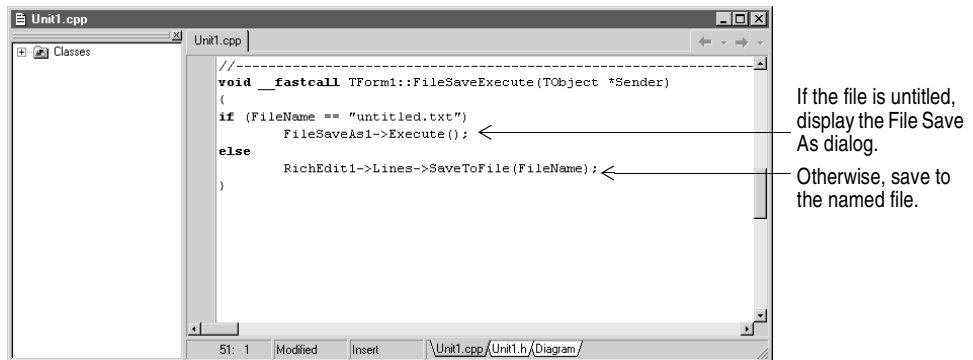
Tip You can also double-click the *FileSave* action in the Object TreeView.

- 3 Right where the cursor is positioned (between { and }), type the following lines:

```
if (FileName == "untitled.txt")
    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, it saves the file using its current name. The *SaveAs* dialog box is defined in the event handler for the *Save As* command. *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 and double-click the *ActionManager* or *ActionList* component.
- 2 Select the *FileSaveAs1* action.

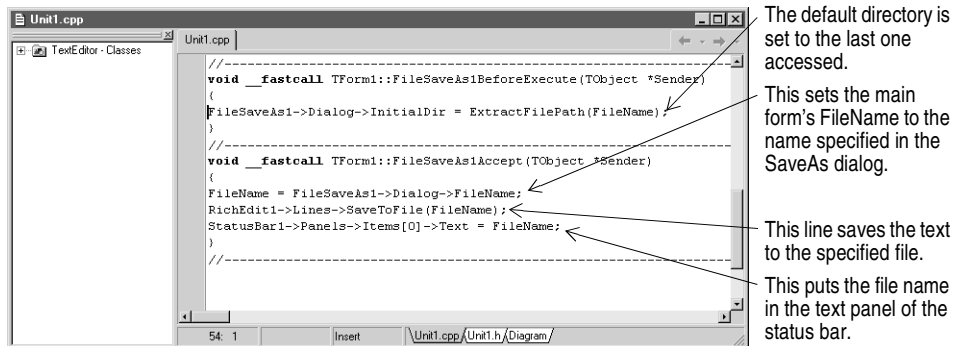
- 3 In the Object Inspector, click the Properties tab. Click the plus sign to the left of *Dialog* to expand its properties. *Dialog* references the Save As dialog box component and displays the Save As dialog box's properties.
- 4 Set *DefaultExt* to `txt`.
- 5 Double-click the text area next to *Filter* to display the Filter editor. In the Filter editor, specify filters for file types as in the Open dialog box.
 - In the first row of the Filter Name column, type `Text files (*.txt)`. In the Filter column, type `*.txt`.
 - In the second row of the Filter Name column, type `All files (*.*)` and in the Filter column, type `*.*`.
 - Click OK.
- 6 Set *Title* to `Save as`.
- 7 Click the Events tab. Double-click the text area next to *BeforeExecute* so that the Code editor opens with the cursor inside the `FileSaveAs1BeforeExecute` event handler.
- 8 Right where the cursor is positioned in the Code editor, type the following line:

```
FileSaveAs1->Dialog->InitialDir = ExtractFilePath (Filename);
```

- 9 The Events tab should still be displayed. Double-click the text area next to the *OnAccept* event so that `FileSaveAs1Accept` appears in the Code editor.
- 10 Where the cursor is positioned, type the following lines:

```
FileName = FileSaveAs1->Dialog->FileName;
RichEdit1->Lines->SaveToFile(FileName);
StatusBar1->Panels->Items[0]->Text = FileName;
```

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



- 11 Choose `File | Save All` to save your changes.

12 To see what the application looks like so far, press *F9*.



The running application looks a lot like the main form in design mode. Notice that the nonvisual objects aren't there.

You can close the application in three ways:

Click the **X**.

Choose File|Exit.

Click the Exit application button on the toolbar.

If you receive any error messages at the bottom of the Code editor, double-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.

13 To return to design mode, click **X** in the upper right corner of the application.

Creating a Help file

It's a good idea to create a Help file that explains how to use your application. C++Builder provides Microsoft Help Workshop in the C:\Project Files\Borland\CBuilder6\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 or Action List editor to display 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 In Windows Explorer, from your C:\Program Files\Borland\CBuilder6\Help directory, open B6X1.zip
- 2 Extract and save the .hlp and .cnt files to your Text Editor directory; by default, C:\Program Files\Borland\CBuilder6\Projects\TextEditor.

Note You can also use any .hlp or .cnt file (such as one of the C++Builder Help files and its associated .cnt file) in your project. You will have to copy them to your project directory and 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* or *ActionList* component.
- 2 Double-click the HelpContents1 action.

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

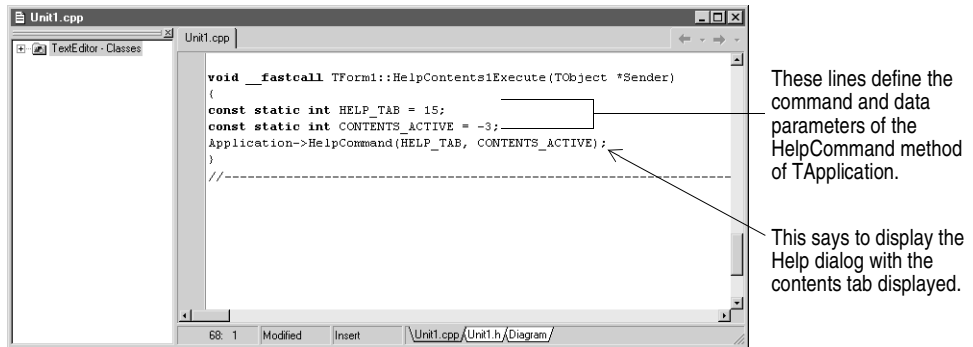
- 3 Right after where the cursor is positioned, type the following lines:

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

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 or Action List editor should still be displayed. If it's not, double-click the *ActionManager* or *ActionList* component on the form.
- 2 Double-click the HelpIndex action.

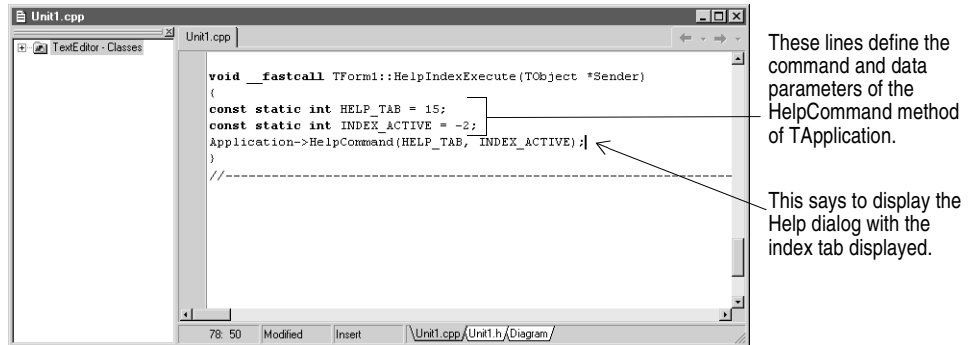
The Code editor opens with the cursor inside the event handler.

- 3 Right after where the cursor is positioned in the text editor, type the following lines:

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

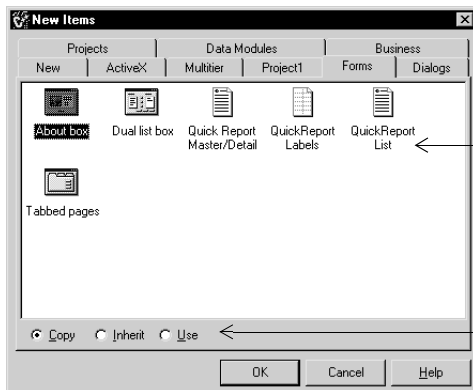
Many applications include an About box which displays information on the product such as the name, version, logos, and may include other legal information including copyright information.

You've already set up a Help About command in the Action Manager or Action List editor.

To add an About box:

- 1 Choose File | New | Other to display the New Items dialog box.

- 2 Click the Forms tab and double-click the About Box icon.



The New Items dialog box is also called the Object Repository.

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.

A predesigned form for an About box appears.

- 3 Select the form itself (click the grid portion) and in the Object Inspector, click the Properties tab and change its *Caption* property to *About Text Editor*.
- 4 Click back on the form (notice it is now called *About Text Editor*). To change each value on the form, click on it so it is highlighted and type the new value.
 - Change Product Name to *Text Editor*.
 - Change Version to *Version 1.0*.
 - Change Copyright to *Copyright 2002*.



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

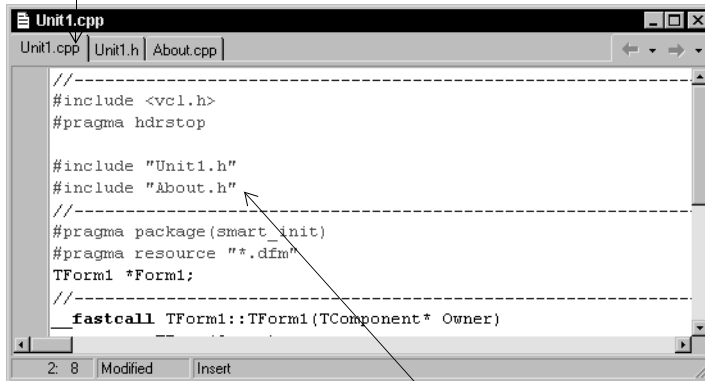
- 5 Save the About box form by choosing File | Save As and saving it as About.cpp.

In the C++Builder Code editor, you should have several files displayed: Unit1.cpp, Unit1.h, About.cpp, and ActnRes (if you have the Enterprise or Professional edition and are using the Action Manager editor). You don't need the ActnRes unit but you can leave it there.

- 6 Click the Unit1.cpp tab and scroll to the top of the Code editor. Add an include statement for the About unit to Unit1. Choose File | Include Unit Hdr and then

select **About** and click **OK**. Notice that `#include About.h` has been added to the top of the `.cpp` file.

Click on 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 editor.



When you create a new form for your application, you need to add it to the main form. Choose **File|Include Unit Hdr** and select the header to add.

- 7 Press *F12* to return to design mode. Double-click the *ActionManager* or *ActionList* component to open it.
- 8 Double-click the **Help | About** 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.

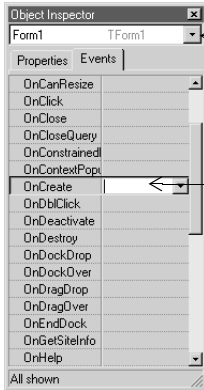
- 9 Choose **File | Save All**.

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 Select the form. The focus should be on the form itself, not any of its components. If it isn't, in the Object Inspector, select `Form1: TForm1` from the drop-down list box at the top.

- 3 Click the Events tab, and double-click the area next to OnCreate.



Check here to make sure focus is on the main form. If it's not, select Form1 from the drop-down list.

Double-click here to create an event handler for the form's OnCreate event.

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

```
Application->HelpFile = ExtractFilePath(Application->ExeName) + "TextEditor.hlp";
FileName = "untitled.txt";
StatusBar1->Panels->Items[0]->Text = FileName;
RichEdit1->Clear();
```

This code initializes the 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.

Creating a CLX database application—a tutorial

This tutorial guides you through the creation of a cross-platform application that lets you view and update a sample employee database. Cross-platform applications use CLX, the Borland Component Library for Cross-Platform. Designed to compile and run on different platforms, CLX applications require a minimum of changes between Windows and Linux ports. (See Borland's latest product offerings for cross-platform compiler support.)

Note This tutorial is written for product editions that include the database components. It sets up database access that requires features not available in the Personal edition. In addition, you must have InterBase installed to successfully complete this tutorial.

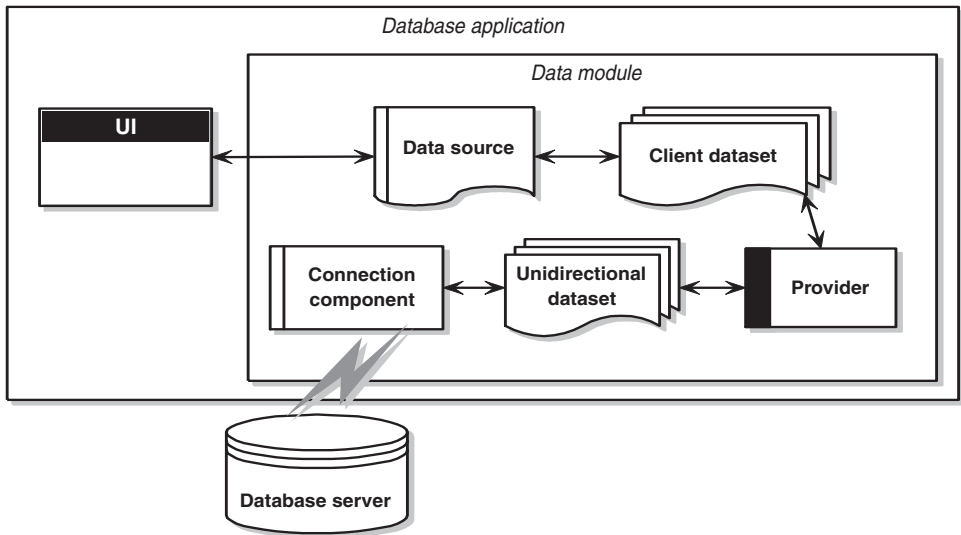
Overview of database architecture

The architecture of a database application may seem complicated at first, but the use of multiple components simplifies the development and maintenance of actual database applications.

Database applications include three main parts: the user interface, a set of data access components, and the database itself. In this tutorial, you will create a dbExpress database application. Other database applications have a similar architecture.

The user interface includes data-aware controls such as a grid so that users can edit and post data to the database. The data access components include the data source, the client dataset, the data provider, a unidirectional dataset, and a connection component. The data source acts as a conduit between the user interface and a client dataset. The client dataset is the heart of the application as it contains a set of records from the underlying database that are buffered in memory. The provider transfers the data between the client dataset and the unidirectional dataset, which fetches data directly from the database. Finally, the connection component establishes a

connection to the database. Each type of unidirectional dataset uses a different type of connection component.



For more information on database development, see “Designing database applications” in the *Developer’s Guide* or online Help.

Creating a new CLX application

Before you begin the tutorial, create a folder to hold the source files. Then create and save a new project.

- 1 Create a folder called Tutorial to hold the project files you’ll create while working through this tutorial.
- 2 Begin a new CLX project. Choose File | New | CLX Application to create a new cross-platform project.
- 3 Choose File | Save All to save your files to disk. When the Save dialog appears, navigate to your Tutorial folder and save each file using its default name.

Later on, you can save your work at any time by choosing File | Save All. If you decide not to complete the tutorial in one sitting, you can open the saved version by choosing File | Reopen and selecting the tutorial from the list.

Setting up data access components

Data access components represent both data (datasets) and the components that connect these datasets to other parts of your application. Each of these data access components points to the next lower component. For example, the data source points

to the client dataset, the client dataset points to the provider, and so forth. Therefore, when you set up your data access components, you add the components in the proper order.

In the following sections, you'll add the database components to create the database connection, unidirectional dataset, provider, client dataset, and data source. Afterwards, you'll create the user interface for the application. These components are located on the dbExpress, Data Access, and Data Controls pages of the Component palette.

Tip It is a good idea to isolate your user interface on its own form and place the data access components in a data module. However, to make things simpler for this tutorial, you'll place the user interface and all the components on the same form.

Setting up the database connection

The dbExpress page contains a set of components that provide fast access to SQL database servers.

You need to add a connection component so that you can connect to a database. The type of connection component you use depends on what type of dataset component you use. In this tutorial you will use the *TSQLConnection* and *TSQLDataSet* components.

To add a dbExpress connection component:



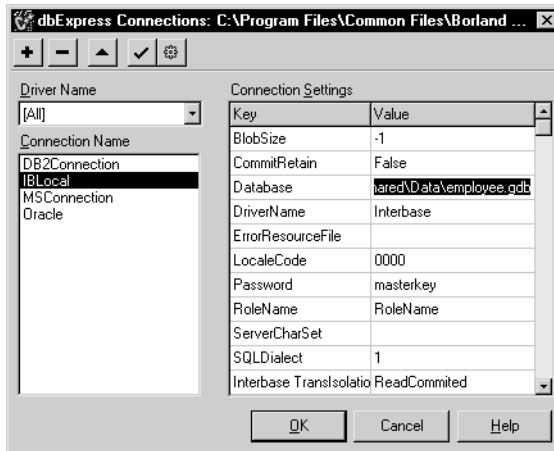
- 1 Click the dbExpress page on the Component palette and double-click the *TSQLConnection* component to place it on the form. To find the *TSQLConnection* component, point at an icon on the palette for a moment; a Help hint shows the name of the component. The component is called *SQLConnection1* by default.

The *TSQLConnection* component is nonvisual, so it doesn't matter where you put it. However, for this tutorial, line up all the nonvisual components at the top of the form.

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

- 2 In the Object Inspector, set its *ConnectionName* property to IBLocal (it's on the drop-down list).
- 3 Set the *LoginPrompt* property to *false*. (By setting this property to *false*, you won't be prompted to log on every time you access the database.)

- 4 Double-click the *TSQLConnection* component to display the Connection Editor.



You use the Connection Editor to select a connection configuration for the *TSQLConnection* component or edit the connections stored in the *dbxconnections.ini* file. Any changes you make in the dialog are written to that file when you click OK. In addition, when you click OK, the selected connection is assigned as the value of the *SQL Connection* component's *ConnectionName* property.

- 5 In the Connection Editor, specify the pathname of the database file called *employee.gdb* on your system. In this tutorial you will connect to a sample InterBase database, *employee.gdb*, that is provided with C++Builder. By default, the InterBase installation places *employee.gdb* in *C:\Program Files\Common Files\Borland Shared\Data*.
- 6 Check the *User_Name* and *Password* fields for acceptable values. If you have not altered the default values, you do not need to change the fields. If database access is administered by someone else, you may need to get a username and password to access the database.
- 7 When you are done checking and editing the fields, click OK to close the Connection Editor and save your changes.

These changes are written to the *dbxconnections.ini* file and the selected connection is assigned as the value of the *SQL Connection* component's *ConnectionName* property

Tip If you need additional help while using the Connection Editor, click the Help button.

- 8 Choose *File | Save All* to save your project.

Setting up the unidirectional dataset

A basic database application uses a dataset to access information from the database. In dbExpress applications, you use a unidirectional dataset. A unidirectional dataset reads data from the database but doesn't update data.

To add the unidirectional dataset:



- 1 From the dbExpress tab, drop *TSQLDataSet* at the top of the form.
- 2 In the Object Inspector, set its *SQLConnection* property to *SQLConnection1* (the database connection created previously).
- 3 Set the *CommandText* property to `"select * from SALES"` to specify the command that the dataset executes. You can either type the Select statement in the Object Inspector or click the ellipsis to the right of *CommandText* to display the CommandText Editor where you can build your own query statement.

Tip If you need additional help while using the CommandText Editor, click the Help button.

- 4 Set *Active* to *true* to open the dataset.
- 5 Choose File | Save All to save the project.

Setting up the provider, client dataset, and data source

The Data Access page contains components that can be used with any data access mechanism, not just dbExpress.

Provider components are the way that client datasets obtain their data from other datasets. The provider receives data requests from a client dataset, fetches data, packages it, and returns the data to the client dataset. If using dbExpress, the provider receives updates from a client dataset and applies them to the database server.

To add the provider:



- 1 From the Data Access page, drop a *TDataSetProvider* component at the top of the form.
- 2 In the Object Inspector, set the provider's *DataSet* property to *SQLDataSet1*.

The client dataset buffers its data in memory. It also caches updates to be sent to the database. You can use client datasets to supply the data for data-aware controls on the user interface using the data source component.

To add the client dataset:



- 1 From the Data Access page, drop a *TClientDataSet* component to the right of the *TDataSetProvider* component.
- 2 Set the *ProviderName* property to *DataSetProvider1*.
- 3 Set the *Active* property to *true* to allow data to be passed to your application.

A data source connects the client dataset with data-aware controls. Each data-aware control must be associated with a data source component to have data to display and manipulate. Similarly, all datasets must be associated with a data source component for their data to be displayed and manipulated in data-aware controls on a form.

To add the data source:



- 1 From the Data Access page, drop a *TDataSource* component to the right of the *TClientDataSet* component.
- 2 Set the data source's *DataSet* property to *ClientDataSet1*.
- 3 Choose File | Save All to save the project.

So far, you have added the nonvisual database infrastructure to your application. Next, you need to design the user interface.

Designing the user interface

Now you need to add visual controls to the application so your users can view the data, edit it, and save it. The Data Controls page provides a set of data-aware controls that work with data in a database and build a user interface. You'll display the database in a grid and add a few commands and a navigation bar.

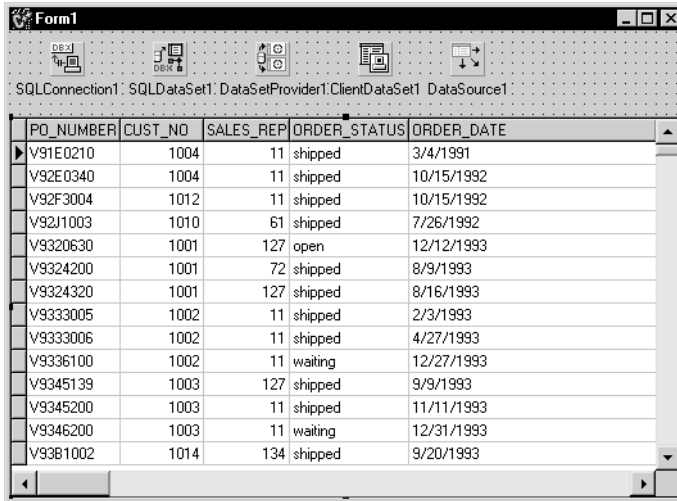
Creating the grid and navigation bar

To create the interface for the application:



- 1 You can start by adding a grid to the form. From the Data Controls page, drop a *TDBGrid* component onto the form.
- 2 Set *DBGrid*'s properties to anchor the grid. Click the **+** next to *Anchors* in the Object Inspector to display *akLeft*, *akTop*, *akRight*, and *akBottom*; set them all to *true*. The easiest way to do this is to double-click *false* next to each property in the Object Inspector.
- 3 Align the grid with the bottom of the form by setting the *Align* property to *alBottom*. You can also enlarge the size of the grid by dragging it or setting its *Height* property to 400.
- 4 Set the grid's *DataSource* property to *DataSource1*. When you do this, the grid is populated with data from the employee database. 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 previous instructions.

So far your application should look like this:



PO_NUMBER	CUST_NO	SALES_REP	ORDER_STATUS	ORDER_DATE
V91E0210	1004	11	shipped	3/4/1991
V92E0340	1004	11	shipped	10/15/1992
V92F3004	1012	11	shipped	10/15/1992
V92J1003	1010	61	shipped	7/26/1992
V9320630	1001	127	open	12/12/1993
V9324200	1001	72	shipped	8/9/1993
V9324320	1001	127	shipped	8/16/1993
V9333005	1002	11	shipped	2/3/1993
V9333006	1002	11	shipped	4/27/1993
V9336100	1002	11	waiting	12/27/1993
V9345139	1003	127	shipped	9/9/1993
V9345200	1003	11	shipped	11/11/1993
V9346200	1003	11	waiting	12/31/1993
V93B1002	1014	134	shipped	9/20/1993

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.



- 5 From the Data Controls page, drop a *TDBNavigator* control onto the form. A database navigator is a tool for moving through the data in a dataset (using next and previous arrows, for example) and performing operations on the data.
- 6 Set the navigator bar's *DataSource* property to *DataSource1* so the navigator is looking at the data in the client dataset.
- 7 Set the navigator bar's *ShowHint* property to *true*. (Setting *ShowHint* to *true* allows Help hints to appear when the cursor is positioned over each of the items on the navigator bar at runtime.)
- 8 Choose File | Save All to save the project.



- 9 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.

PO_NUMBER	CUST_NO	SALES_REP	ORDER_STATUS	ORDER_DATE
V91E0210	1004	11	shipped	3/4/1991
V92E0340	1004	11	shipped	10/15/1992
V92F3004	1012	11	shipped	10/15/1992
V92J1003	1010	61	shipped	7/26/1992
V9320630	1001	127	open	12/12/1993
V9324200	1001	72	shipped	8/9/1993
V9324320	1001	127	shipped	8/16/1993
V9333005	1002	11	shipped	2/3/1993
V9333006	1002	11	shipped	4/27/1993
V9336100	1002	11	waiting	12/27/1993
V9345139	1003	127	shipped	9/9/1993
V9345200	1003	11	shipped	11/11/1993
V9346200	1003	11	waiting	12/31/1993
V93B1002	1014	134	shipped	9/20/1993

When you run your project, the program opens in a window like the one you designed on the form. You can test the navigation bar with the employee database. For example, you can move from record to record using the arrow commands, add records using the **+** command, and delete records using the **-** command.

Tip

If you should encounter an error while testing an early version of your application, choose Run | Program Reset to return to the design-time view.

Adding support for a menu

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

In this section, you'll add an *action list*. While you can create menus, toolbars, and buttons without using action lists, action lists simplify development and maintenance by centralizing responses to user commands. (Note that for Windows-only development, you can also use Action Bands to simplify development of toolbars and menus.)

- 1 If the application is still running, click the **X** in the upper right corner to close the application and return to the design-time view of the form.

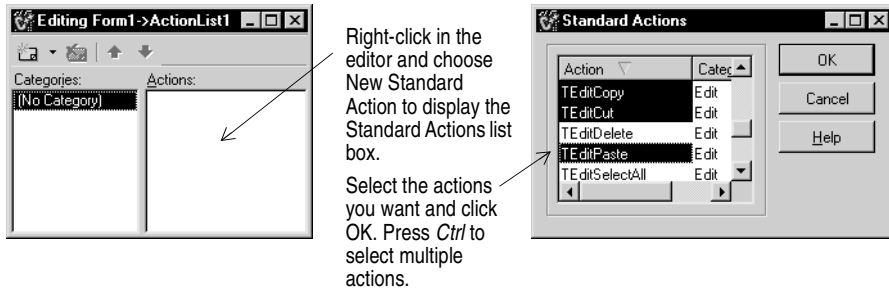


- 2 From the Common Controls page of the Component palette, drop an *ImageList* component onto the form. Line this up next to the other nonvisual components. The *ImageList* will contain icons that represent standard actions like cut and paste.

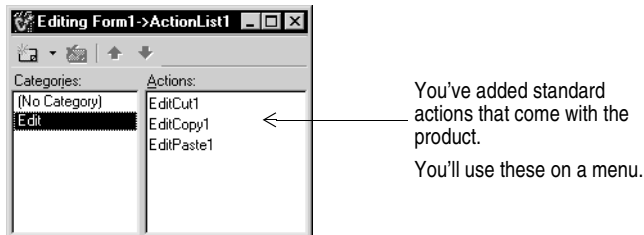


- 3 From the Standard page of the Component palette, drop an *ActionList* component onto the form. Set the action list's *Images* property to *ImageList1*.

- 4 Double-click the action list to display the Action List editor.



- 5 Right-click the Action List editor and choose New Standard Action. The Standard Actions list box appears.
- 6 Select the following actions: *TEditCopy*, *TEditCut*, and *TEditPaste*. (Use the *Ctrl* key to select multiple items.) Then click OK.



- 7 Right-click on the Action List editor and choose New Action to add another action (not provided by default). Action1 is added by default. In the Object Inspector, set its *Caption* property to *Update Now!*

This same action will be used on a menu and a button. Later on, we'll add an event handler so it will update the database.

- 8 Click (No Category), right-click and choose New Action to add another action. Action2 is added. Set its *Caption* property to *E&xit*.
 - 9 Click the **X** (in the upper right corner) to close the Action List editor.
- You've added three standard actions plus two other actions that we'll connect to event handlers later.
- 10 Choose File | Save All to save the project.

Adding a menu

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

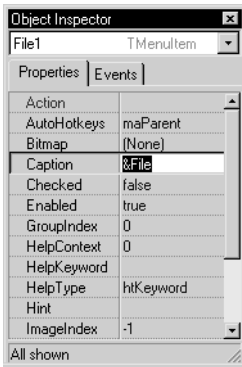


- 1 From the Standard page of the Component palette, drop a *TMainMenu* component onto the form. Drag it next to the other nonvisual components.

- 2 Set the main menu's *Images* property to `ImageList1` to associate the image list with the menu items.
- 3 Double-click the menu component to display the Menu Designer.

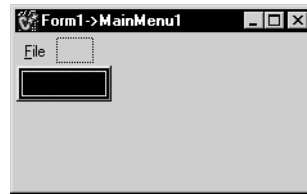


- 4 Type `&File` to set the *Caption* property of the first top-level menu item and press *Enter*.



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

The ampersand before a character activates an accelerator key.



- 5 Select the blank menu item below the File menu. Set the blank menu item's *Action* property to `Action2`. An Exit menu item appears under the File menu.
- 6 Click the second top-level menu item (to the right of File). Set its *Caption* property to `&Edit` and press *Enter*. Select the blank menu item that appears under the Edit menu.
- 7 In the Object Inspector, set the *Action* property to `EditCut1` and press *Enter*. The item's caption is automatically set to `Cut` and a default cut bitmap appears on the menu.
- 8 Select the next blank menu item (under *Cut*) and set its *Action* property to `EditCopy1` (a default copy bitmap appears on the menu).
- 9 Select the next blank menu item (under *Copy*) and set its *Action* property to `EditPaste1` (a default paste bitmap appears on the menu).
- 10 Select the next blank menu item (under *Paste*) and set its *Caption* property to a hyphen (-) to create a divider line in the menu. Press *Enter*.
- 11 Select the next blank menu item (under the divider line) and set its *Action* property to `Action1`. The menu item displays `Update Now!`
- 12 Click the **X** to close the Menu Designer.

13 Choose File | Save All to save the project.

14 Press *F9* or Run on the toolbar to run your program and see how it looks.



PO_NUMBER	CUST_NO	SALES_REP	ORDER_STATUS	ORDER_DATE
V91E0210	1004	11	shipped	3/4/1991
V92E0340	1004	11	shipped	10/15/1992
V92F3004	1012	11	shipped	10/15/1992
V92J1003	1010	61	shipped	7/26/1992
V9320630	1001	127	open	12/12/1993
V9324200	1001	72	shipped	8/9/1993
V9324320	1001	127	shipped	8/16/1993
V9333005	1002	11	shipped	2/3/1993
V9333006	1002	11	shipped	4/27/1993
V9336100	1002	11	waiting	12/27/1993
V9345139	1003	127	shipped	9/9/1993
V9345200	1003	11	shipped	11/11/1993
V9346200	1003	11	waiting	12/31/1993
V93B1002	1014	134	shipped	9/20/1993

Many of the commands on the edit menu and the navigation bar are operational at this time. Copy and Cut are grayed on the Edit menu until you select some text in the database. You can use the navigation bar to move from record to record in the database, insert a record, or delete a record. The Update command does not work yet.

Close the application when you're ready to continue.

Adding a button

This section describes how to add an Update Now button to the application. This button is used to apply any edits that a user makes to the database, such as editing records, adding new records, or deleting records.

To add a button:



- 1 From the Standard page of the Component palette, drop a *TButton* onto the form. (Select the component then click the form next to the navigation bar.)
- 2 Set the button's *Action* property to *Action1*.

The button's caption changes to Update Now! When you run the application, it will be grayed out until an event handler is added to make it work.

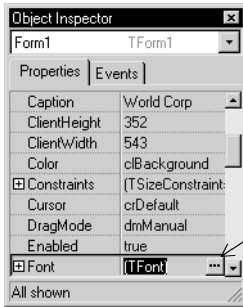
Displaying a title and an image

You can add a company title and an image to make your application look more professional:



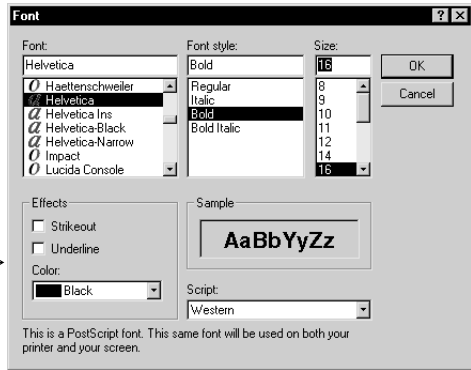
- 1 From the Standard page of the component palette, drop a *TLabel* component onto the form (named *Label1* by default).

- 2 In the Object Inspector, change the label's *Caption* property to World Corp or another company name.
- 3 Change the company name's font by clicking the *Font* property. Click the ellipsis that appears on the right and in the Font dialog box, change the font to Helvetica Bold, 16-point type. Click OK.

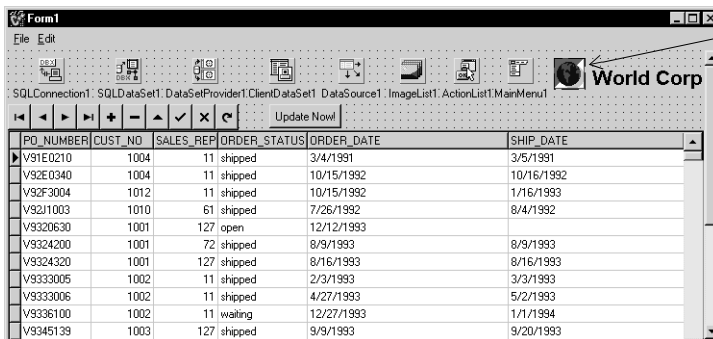


You can change the font of the label using the *Font* property in the Object Inspector.

Click on the ellipsis to display a standard font dialog.



- 4 Position the label in the upper right corner.
- 5 From the Additional Component palette page, drop a *TImage* component next to the label (named *Image1* by default).
- 6 To add an image to the *Image1* component, click the *Picture* property. Click the ellipsis to display the Picture editor.
- 7 In the Picture editor, choose Load and navigate to earth.ico. On C++Builder, its path is Program Files\Common Files\Borland Shared\images\icons\earth.ico.
- 8 Double-click earth.ico. Click OK to load the picture and to close the Picture editor.
- 9 Size the default image area to the size of the picture. Place the image next to the label.



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

- 10 To align the text and the image, select both objects on the form, right-click, and choose Align. In the Alignment dialog box, under Vertical, click Bottoms.
 - 11 Choose File | Save All to save the project.
 - 12 Press *F9* to compile and run your application.
- Close the application when you're ready to continue.

Writing an event handler

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 a component, such as *TButton*, is clicked. If you select a component on a form and click the Object Inspector's Events tab, you'll see a list of the component's events.

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

Writing the Update Now! command event handler

First, you'll write the event handler for the Update Now! command and button:

- 1 Double-click the *ActionList* component to display the Action List editor.
- 2 Select (No Category) to see Action1 and Action2.
- 3 Double-click Action1. In the Code editor, the following skeleton event handler appears:

```
void __fastcall TForm1::Action1Execute(TObject *Sender)
{
    //
}
```

Right where the cursor is positioned (between the braces), type:

```
if(ClientDataSet1->State == dsEdit || ClientDataSet1->State == dsInsert)
    ClientDataSet1->Post();

    ClientDataSet1->ApplyUpdates(-1);
```

This event handler first checks to see what state the database is in. When you move off a changed record, it is automatically posted. But if you don't move off a changed record, the database remains in edit or insert mode. The *if* statement posts any data that may have been changed but was not passed to the client dataset. The next statement applies updates held in the client dataset to the database.

Note Changes to the data are not automatically posted to the database when using *dbExpress*. You need to call the *ApplyUpdates* method to write all updated, inserted, and deleted records from the client dataset to the database.

Writing the Exit command event handler

Next, we'll write the event handler for the Exit command:

- 1 Double-click the *ActionList* component to display the Action List editor if it is not already displayed.
- 2 Click (No Category) so you see Action2.

- 3 Double-click Action2. The Code editor displays the following skeleton event handler:

```
void __fastcall TForm1::Action2Execute(TObject *Sender)
{
    //
}

```

Right where the cursor is positioned (between the braces), type:

```
Close();
```

This event handler will close the application when the File | Exit command on the menu is used.

- 4 Close the Action List editor.
- 5 Choose File | Save All to save the project.

Writing the FormClose event handler

Finally, you'll write another event handler that is invoked when the application is closed. The application can be closed either by using File | Exit or by clicking the **X** in the upper right corner. Either way, the program checks to make sure that there are no pending updates to the database and displays a message window asking the user what to do if changes are pending.

You could place this code in the Exit event handler but any pending database changes would be lost if users chose to exit your application using the **X**.

- 1 Click the main form to select it (rather than any specific object on it).
- 2 Select the Events tab in the Object Inspector to see the form events.
- 3 Double-click *OnClose* (or type FormClose next to the *OnClose* event and click on it). A skeleton FormClose event handler is written and displayed in the code editor after the other event handlers:

```
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
    //
}

```

Right where the cursor is positioned (between the braces), type:

```
TMessageButton Option;
TMessageButtons msgButtons;

msgButtons << smbYes << smbNo << smbCancel;

Action = caFree;
if(ClientDataSet1->State == dsEdit || ClientDataSet1->State == dsInsert)
    ClientDataSet1->Post();
if(ClientDataSet1->ChangeCount > 0) {

```

```

    Option = Application->MessageBox("You have pending updates. Do you want to write them
to the database?", "Pending Updates", msgButtons, smsWarning, smbYes, smbNo);
    if (Option == smbYes)
        ClientDataSet1->ApplyUpdates(-1);
    else
        if (Option == smbCancel)
            Action = caNone;
}

```

This event handler checks the state of the database. If changes are pending, they are posted to the client dataset where the change count is increased. Then before closing the application, a message box is displayed that asks how to handle the changes. The reply options are Yes, No, or Cancel. Replying Yes applies updates to the database; No closes the application without changing the database; and Cancel cancels the exit but does not cancel the changes to the database and leaves the application still running.

4 Check that the whole procedure looks like this:

```

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
    TMessageButton Option;
    TMessageButtons msgButtons;

    msgButtons << smbYes << smbNo << smbCancel;

    Action = caFree;
    if (ClientDataSet1->State == dsEdit || ClientDataSet1->State == dsInsert)
        ClientDataSet1->Post();
    if (ClientDataSet1->ChangeCount > 0) {

        Option = Application->MessageBox("You have pending updates. Do you want to write them
to the database?", "Pending Updates", msgButtons, smsWarning, smbYes, smbNo);
        if (Option == smbYes)
            ClientDataSet1->ApplyUpdates(-1);
        else
            if (Option == smbCancel)
                Action = caNone;
    }
}

```

5 To finish up, choose File | Save All to save the project. Then press *F9* to run the application.

Tip Fix any errors that occur by double-clicking the error message to go to the code in question or by pressing *F1* for Help on the message.

That's it! You can try out the application to see how it works. When you want to exit the program, you can use the fully functional File | Exit command.

Customizing the desktop

This chapter explains some of the ways you can customize the tools in C++Builder 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

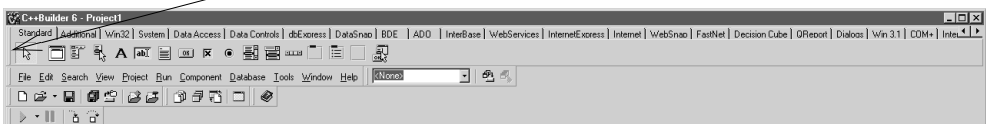
In the main window, you can reorganize the menu, toolbars, and Component palette by clicking the grabber on the left-hand side of each one and dragging it to another location.

Main window in its default arrangement.



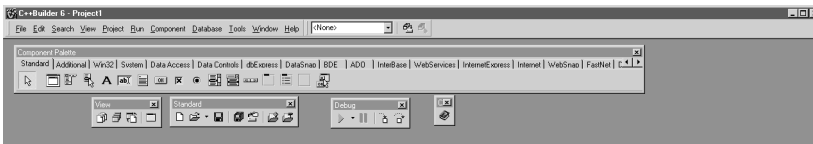
You can move toolbars and menus within the main window. Click the grabber (the double bar on the left) and drag it to where you want it.

Main window organized differently



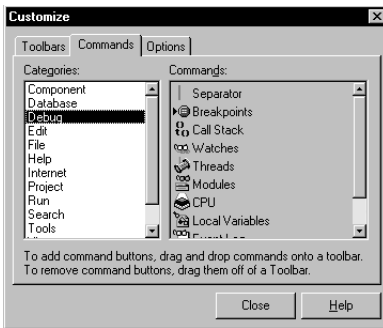
Organizing your work area

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.

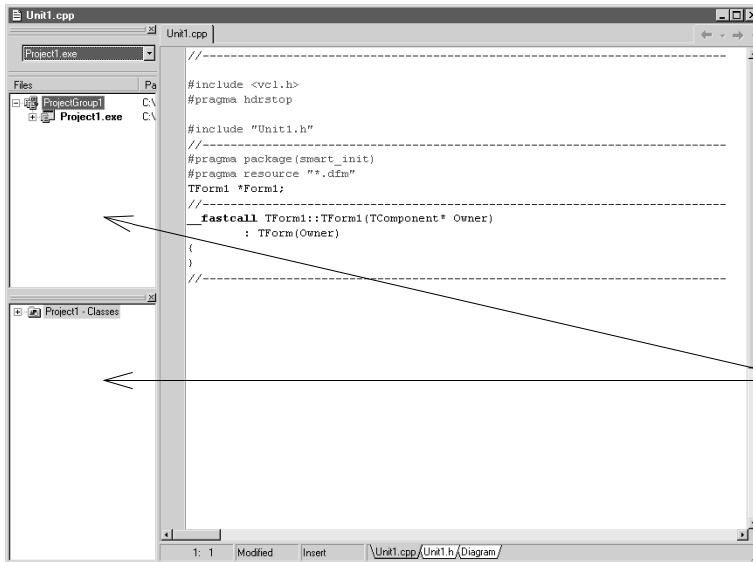
For more information...

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 C++Builder in its default configuration, the ClassExplorer is docked to the left of the Code editor. You can add the Project Manager to the first two to create three docked windows.

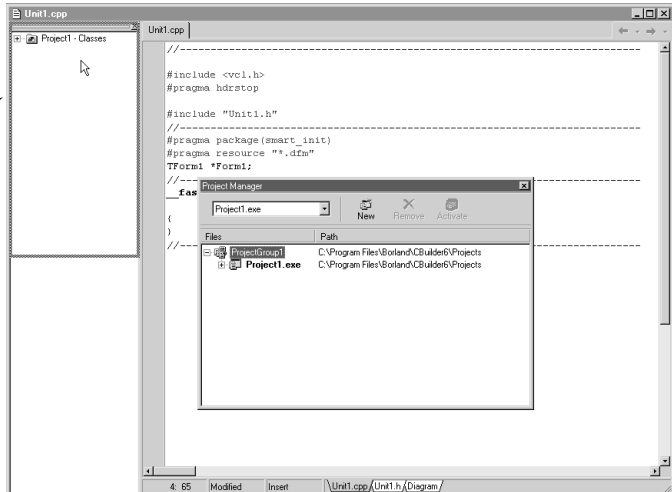


You can combine, or "dock" windows with either grabbers, as on the right, or tabs.

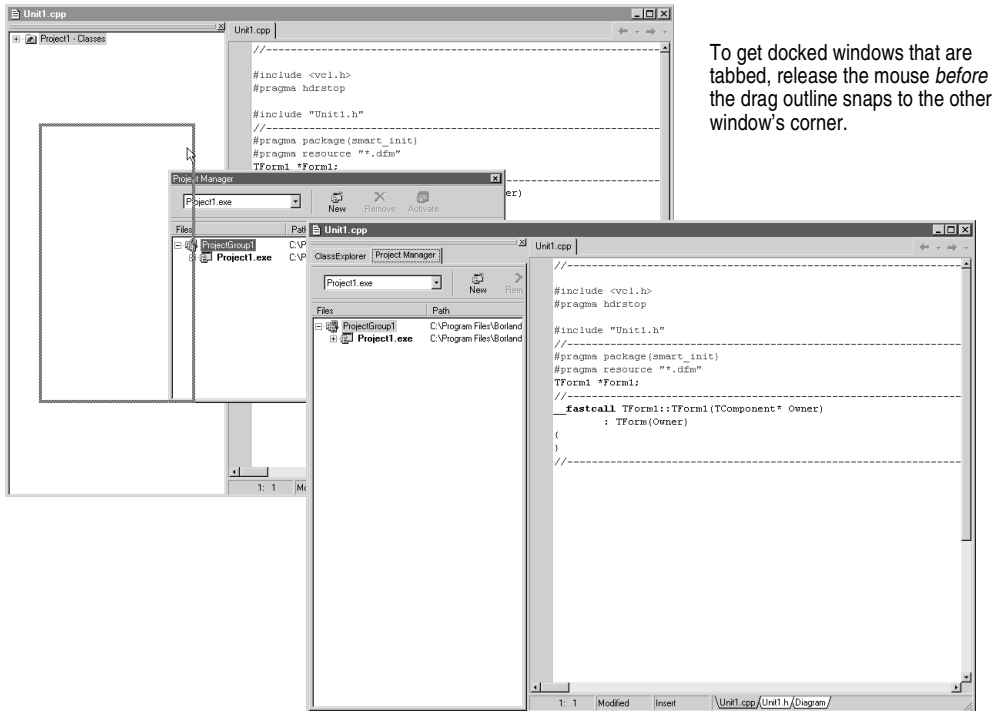
Here the Project Manager and Class Explorer are docked to the Code editor.

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 undock a window, double-click its grabber or tab, or click and drag the tab outside of the docking area.

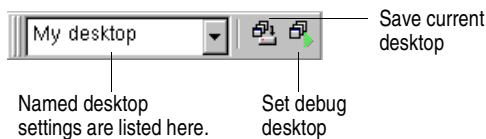
To turn off automatic docking, either press the *Ctrl* key while moving windows around the screen, or choose Tools | Environment Options, click the Preferences page, and uncheck the Auto drag docking check box.

For more information...

See “docking” in the online Help index.

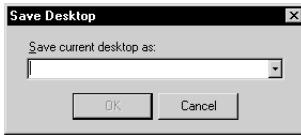
Saving desktop layouts

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.

For more information...

Click the Help button in the Palette Properties dialog box.

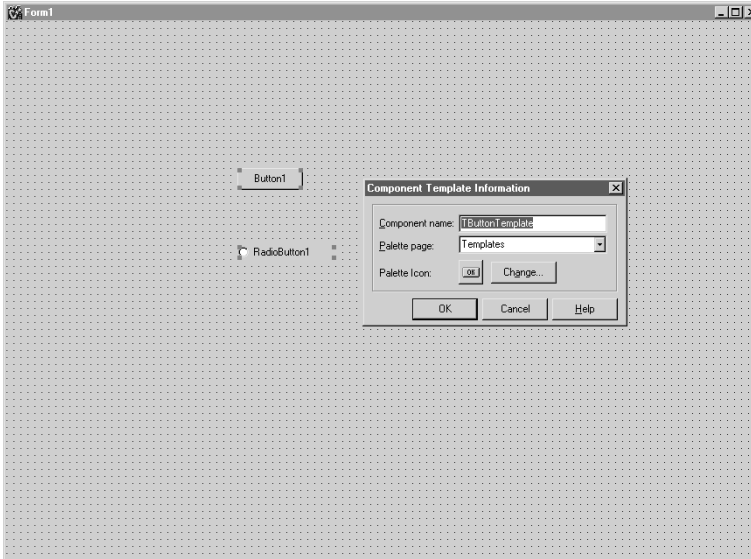
Creating component templates

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...

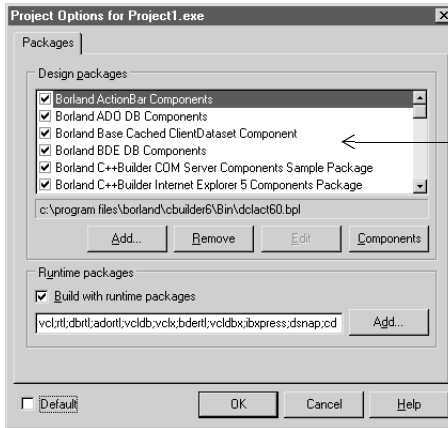
See “templates, component” in the online Help index.

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 C++Builder 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. C++Builder 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 C++Builder. When you install new components from third-party vendors, their package appears in this list. Click Components to see what components the package contains.

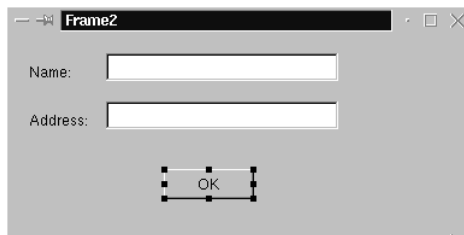
For more information...

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 C++Builder 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, C++Builder 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 default.bpr, located in the Cbuilder6\Bin directory. To restore C++Builder’s original default settings, delete or rename the default.bpr file.

For more information...

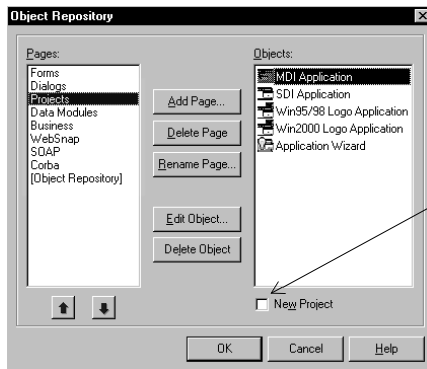
See “Project Options dialog box” in the online Help index.

Specifying project and form templates as the default

When you choose File | New | Application, C++Builder 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 6-9). Or you can choose from one of C++Builder’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, C++Builder 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, C++Builder 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.

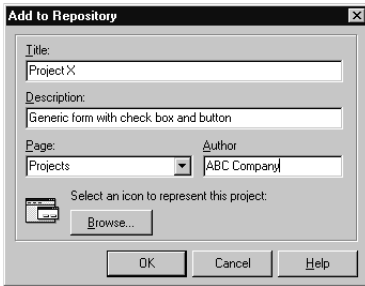
For more information...

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 C++Builder, see “Specifying project and form templates as the default” on page 6-8.

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 2-6.

For more information...

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

Index

A

- About box, adding 4-31
- Action Manager editor 4-9 to 4-12
- actions, adding to an application 4-9, 4-11, 4-17
- ActiveX
 - Component palette page 3-12
 - installing controls 6-8
- adding
 - components to a form 4-18
- adding components to a form 4-3
- adding items to Object Repository 2-5
- ADO 3-11
- applications
 - compiling and debugging 3-7, 4-13, 4-21
 - creating 3-1, 3-9
 - database 3-10
 - deploying 3-8
 - internationalizing 3-8
 - Web server 3-9

B

- BDE 3-11
- BDE Administrator 3-11
- bitmaps, adding to an application 4-7, 4-13
- Borland Component Library for Cross Platform (CLX) 3-6
- .BPR files 4-2

C

- C++Builder
 - customizing 6-1 to 6-11
 - programming 3-1
- character sets, extended 3-8
- class libraries 3-6
- classes, defined 4-4
- ClassExplorer 2-9
- closing a form 4-3
- CLX
 - adding components 2-4
 - applications, creating 3-9
 - defined 3-6
- code
 - event handlers 3-5
 - help in writing 2-6 to 2-7
 - viewing and editing 2-6 to 2-9
 - writing 3-5
- code completion 2-6
- Code editor
 - combining with other windows 6-2

- customizing 6-11
 - using 2-6 to 2-7
- Code Explorer
 - using 2-9
- Code Parameters 2-6
- Code Templates 2-6
- compiling applications 3-7
- compiling programs 4-21, 5-8
- Component palette
 - adding custom components 3-12
 - adding pages 6-5
 - customizing 6-5 to 6-7
 - defined 2-4
 - using 3-2
- component templates, creating 6-5
- components
 - adding to a form 3-2, 4-3
 - adding to Component palette 6-5
 - arranging on Component palette 6-5
 - creating custom 3-12
 - customizing 3-12, 6-5
 - defined 4-3
 - installing 3-12, 6-6
 - setting properties 3-4, 4-2
- context menus, accessing 2-3
- controls, adding to a form 3-2, 4-3
- cross-platform
 - developing applications for 3-9
- customizing
 - Code editor 6-11
 - Component palette 2-3
 - Form Designer 6-10
 - IDE 6-1 to 6-11

D

- Data Dictionary 3-12
- data modules
 - adding 3-2
 - creating 2-5
- database applications
 - accessing 5-3 to 5-4
- database applications, creating 3-10
- Database Desktop 3-11
- database example 5-1 to 5-15
- Database Explorer 3-11
- dbExpress 3-10
- debugging programs 3-7 to 3-8, 4-13
- default
 - project and form templates 6-8
 - project options 6-8

- deploying applications 3-8
- design-time view, closing forms 4-3
- desktop
 - organizing 6-1 to 6-5
 - saving layouts 6-4
- developer support 1-6
- .dfm files 2-9, 4-1
- Diagram page 2-7
- dialog boxes, in Object Repository 2-5
- DLLs
 - creating 2-5
 - defined 3-12
 - deploying 3-8
- docking windows 6-2 to 6-4
- documentation, ordering 1-6

E

- Editing StatusBar1.Panels dialog box 4-5
- Editor Options dialog box 2-7, 6-11
- Environment Options dialog box 6-10
- error messages 4-29
- event handlers 5-13 to 5-15
 - creating 4-22 to 4-29
 - defined 3-5
- events 5-13 to 5-15
- example program 5-1 to 5-15
- executables, deploying 3-8

F

- files
 - form 2-9, 4-1
 - project 4-1
 - saving 4-2
 - unit 4-1
- Form Designer
 - customizing 6-10
 - defined 2-4
- form files
 - defined 4-1
 - viewing code 2-9
- forms
 - adding components to 3-2, 4-3
 - closing 4-3
 - finding 2-5
 - main 4-2, 6-9
 - specifying as default 6-9
- frames 6-7

G

- graphics, displaying 5-11
- GUIs, creating 4-2

H

- header files 4-1
- Help files, adding to an application 4-29
- Help tooltips 4-4
- Help, F1 1-3

I

- IDE
 - customizing 6-1 to 6-11
 - defined 1-1
 - organizing 6-1
 - tour of 2-1
- images
 - displaying 5-11
- images, adding to an application 4-7, 4-13
- IMEs 3-8
- information, finding 1-3
- input method editors 3-8
- installing custom components 6-6
- integrated debugger 3-7
- integrated development environment (IDE)
 - customizing 6-1 to 6-11
 - tour of 2-1
- InterBase 3-11
- internationalizing applications 3-8

K

- keystroke mappings 6-11

L

- localizing applications 3-8

M

- main form, defined 6-9
- MainMenu component 5-9
- makefiles 4-2
- menus
 - adding to an application 4-18
 - context 2-3
 - in C++Builder 2-3
 - organizing 2-3, 6-1
- messages, error 4-29

N

- new features 1-3
- new form, defined 6-9
- New Items dialog box
 - saving templates to 6-8, 6-10
 - using 2-5, 4-31
- newsgroups 1-6

O

- Object Inspector
 - defined 2-4
 - inline component references 3-4
 - using 3-4, 4-2
- Object Repository
 - adding templates to 6-8, 6-9
 - defined 2-5, 3-1
 - using 2-5 to 2-6
- Object TreeView 2-4
- objects, defined 3-6
- ODBC 3-11
- online Help files 1-3
- options, setting for projects 6-8

P

- packages 6-6
- Panel component 5-11
- Paradox 3-11
- parent-child relationships 2-4
- .pas files 4-1
- pictures, displaying 5-11
- programming with C++Builder 3-1
- programs
 - CLX applications 3-9
 - compiling and debugging 3-7, 4-13, 4-21
 - deploying 3-8
 - internationalizing 3-8
 - Web server applications 3-9
- project files 4-2
- project files, default names 4-1
- project groups 2-10
- Project Manager 2-9 to 2-10
- Project Options dialog box 6-8
- project templates 6-9
- projects
 - adding items to 2-5
 - creating 3-1
 - managing 2-9 to 2-10
 - saving 4-2
 - setting options as default 6-8
 - specifying as default 6-8
 - types 3-9 to 3-12
- properties, setting 3-4, 4-2, 4-9, 4-11, 4-16

R

- Resource DLL Wizard 3-8
- right-click menus 2-3
- Run button 5-8
- running an application 3-7, 4-13
- running applications 4-21, 5-8

S

- sample program 4-1 to 4-34
- saving
 - desktop layouts 6-4
 - projects 4-2
- setting properties 3-4, 4-2, 4-9, 4-11, 4-16
- source code
 - help in writing 2-6 to 2-7
- SQL database servers 3-10
- SQL Explorer 3-11
- SQL Links 3-11
- SQL Server 3-11
- standard actions, adding to an application 4-17
- starting C++Builder 2-1
- support services 1-6

T

- tabbed windows, docking 6-3
- technical support 1-6
- templates
 - adding to Object Repository 6-9
 - specifying as default 6-8
- text editor tutorial 4-1 to 4-34
- to-do lists 2-10
- tool windows, docking 6-2
- toolbars 2-3
 - adding and deleting components from 6-2
 - adding to an application 4-13, 4-20
 - organizing 6-1
- Tooltip Expression Evaluation 2-6
- Tooltip Symbol Insight 2-6
- tooltips 4-4
- translation tools 3-8
- tutorial 4-1 to 4-34, 5-1 to 5-15
- type libraries, defined 3-13
- Typographic conventions 1-6
- typographic conventions 1-6

U

- unit files 4-1
- unit header files 4-1
- user interfaces, creating 3-2, 4-2, 4-3

V

- versions of C++Builder 3-9
- Visual Component Library (VCL)
 - adding components 2-4
 - using 3-6

W

Web server applications, creating 3-9

Web site, Borland 1-6

WebSnap, introduction 3-9

windows, combining 6-2

wizards, finding 2-5

Writing code 3-5

X

.xfrm files 2-9