# Quick Start

Borland®
## Kylix™ 3
### Delphi™ and C++ for Linux®

Borland Software Corporation
100 Enterprise Way, Scotts Valley, CA 95066-3249
www.borland.com

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

Borland Software Corporation may have patents and/or pending patent applications covering subject matter in this document. Please refer to the product CD or the About dialog box for the list of applicable patents. The furnishing of this document does not give you any license to these patents.



Printed in the U.S.A.

HDE7030WW21000 3E3R0702
0203040506-9 8 7 6 5 4 3 2 1
D3

# Contents

Chapter

1

# Introduction

This *Quick Start* provides an overview of the Kylix 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 Kylix.

Chapter 2, "A tour of the environment," describes the main tools on the Kylix desktop, or integrated desktop environment (IDE). Chapter 3, "Programming with Kylix" explains how you use some of these tools to create an application. Chapter 4, "Customizing the desktop" describes how you can customize the Kylix IDE for your development needs.

## What is Kylix?

Kylix is an object-oriented, visual programming environment for rapid application development (RAD). Using Kylix, you can create highly efficient applications for Linux servers and workstations with a minimum of manual coding. Kylix provides all the tools you need to develop, test, and deploy applications in both the Delphi and C++ programming languages, including a large library of reusable components, a suite of design tools, application and form templates, and programming wizards.

# Registering Kylix

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

- Register using your internet connection.

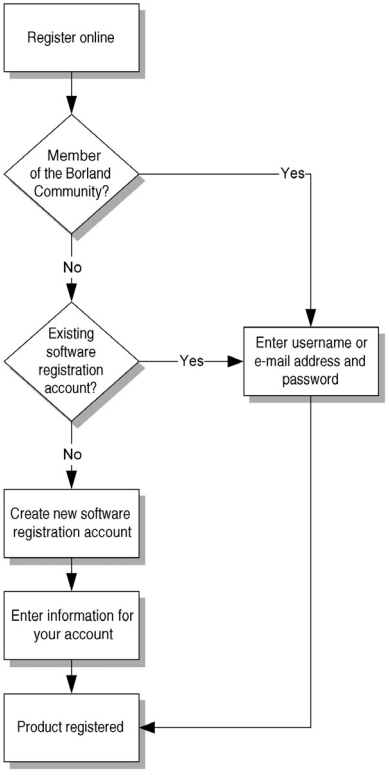
  Use this option to register online using your existing internet connection.

- Register by phone or Web browser.

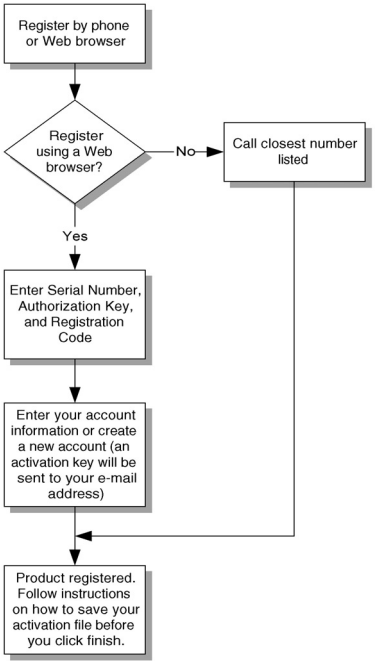
  Use this option to register by phone or through your web browser. If you received an activation key via email, use this option to select the file.

- Import software activation information from a file or email.
- Register later.

Online registration is the easiest way to register Kylix, 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 Kylix. If not, the registration process provides a way to create an account.



The second option (register by phone or Web page) is useful if the machine you are installing on is not connected to the internet, or if you are behind a firewall that is blocking online registration.




If you have previously received software activation information, you can select the *Import software activation information from a file or email* option and select the activation.slip file on your system.

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

# Finding information

You can find information on Kylix in the following ways:

- 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.

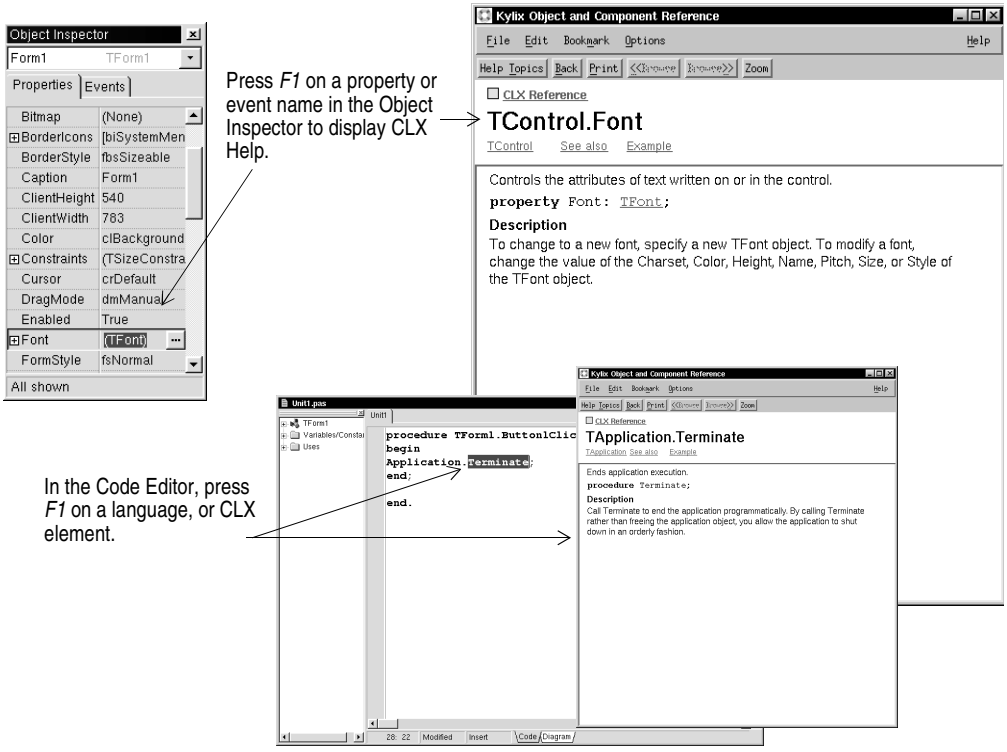
## Online Help

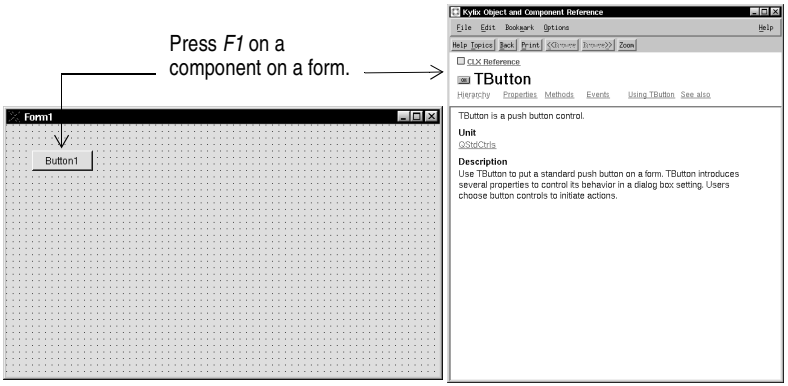
The online Help system provides detailed information about user interface features, language implementation, programming tasks, and the components in the Borland Component Library for Cross-Platform (CLX). It includes all the material in the Kylix *Developer's Guide*, *Delphi Language Guide*, and a host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

To view the table of contents, choose the first item under the Help menu and click the Contents tab. To look up CLX objects or any other topic, click the Index or Find tab and type your request.

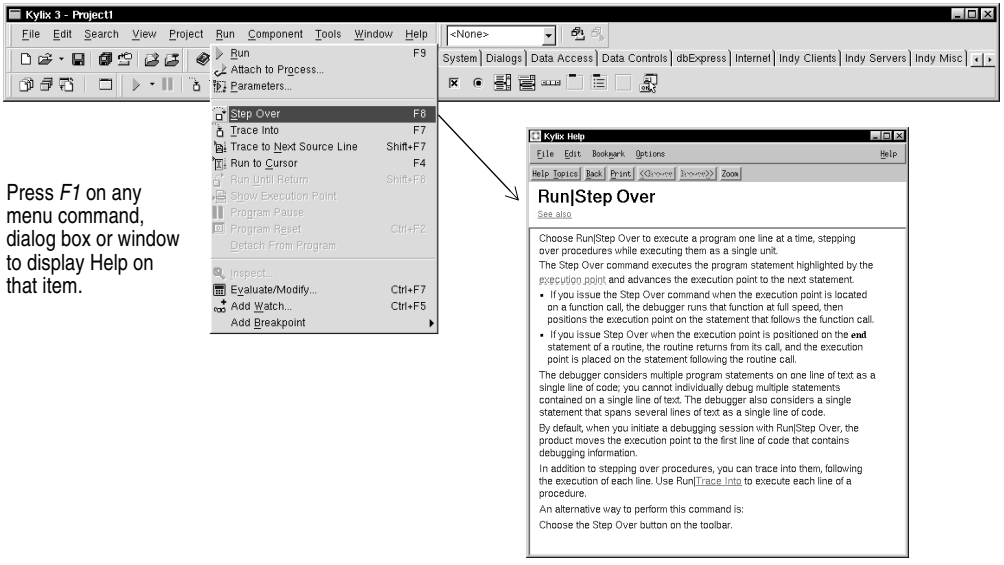
## F1 Help

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






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




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

# 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 Kylix developers exchange information, tips, and techniques. The site also includes a list of books about Kylix, additional Kylix 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. |
| *Italics* | Italicized words in text represent Kylix identifiers, such as variable or type names. Italics are also used to emphasize certain words, such as new terms. |
| *Keycaps* | This typeface indicates a key on your keyboard. For example, "Press *Esc* to exit a menu." |
|  | The Delphi printed icon represents Delphi programming language text and code examples. |
|  | The C++ printed icon represents C++ programming language text and code examples. |

Chapter

2

# A tour of the environment

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

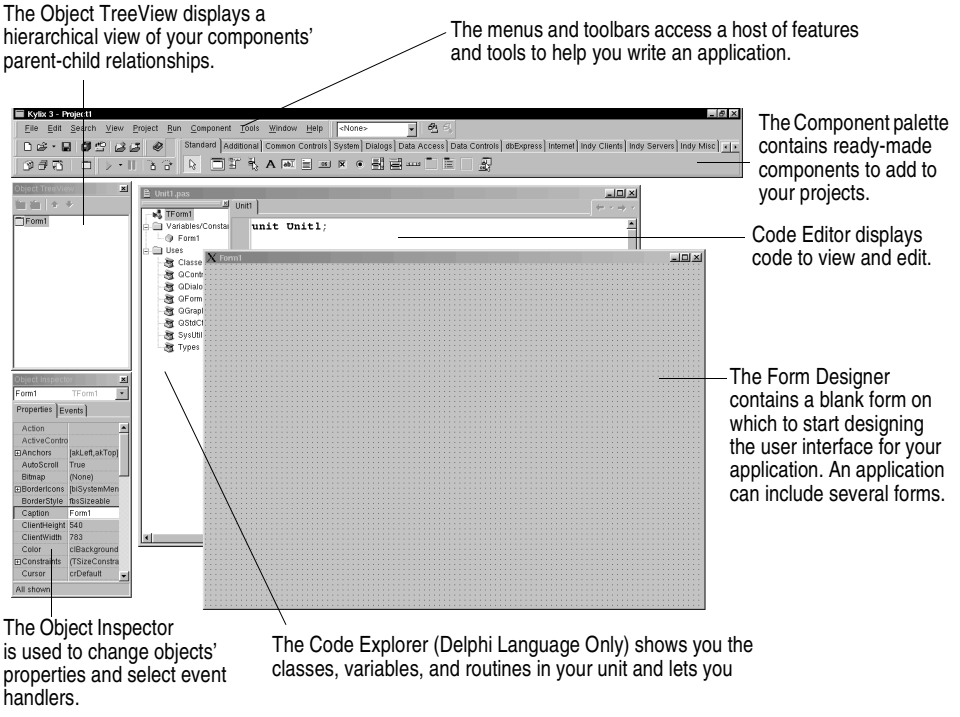
## Starting Kylix

You can start Kylix in the following ways:

- In either KDE or Gnome, open the start menu, choose Borland Kylix 3 and select either the C++ or the Delphi version of Kylix.
- From the bin directory of your Kylix installation, type ./startdelphi (Delphi language IDE) or ./startbcb (C++ IDE).

## The IDE

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




**Note** The previous and subsequent examples show Kylix 3 using the Delphi Language Integrated Development Environment (IDE). This guide will note when there are significant differences between the Delphi and C++ environments.

Kylix'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 Kylix 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, compile, test, debug, and manage projects without leaving the IDE.

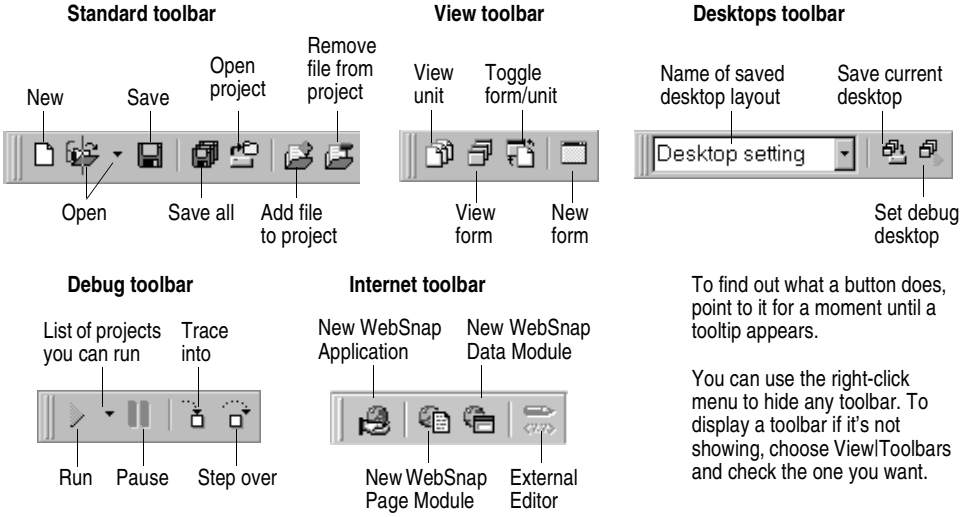
To learn about organizing and configuring the IDE, see "Customizing the desktop" on page 4-1.

# The menus and toolbars

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


Main window in its default arrangement.

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




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

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

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

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

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

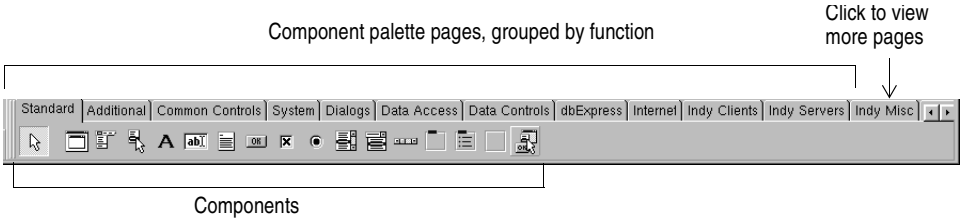
### *For more information...*

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

# The Component Palette, Form Designer, and Object Inspector

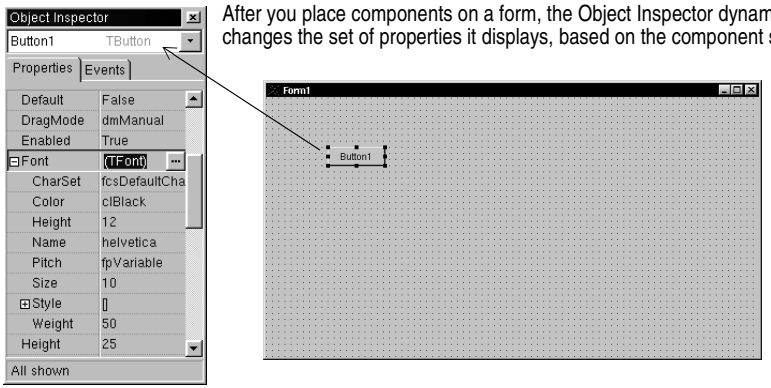
The Component palette, Form Designer, Object Inspector, and Object TreeView work together to help you build a user interface for your application.

The Component palette includes tabbed pages with groups of icons representing visual or nonvisual CLX components. The pages divide the components into various functional groups. For example, the Standard, Additional, and Common Controls pages include 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 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. See “Placing components on a form” on page 3-2.




***For more information...***

See “Component palette” in the online Help index.

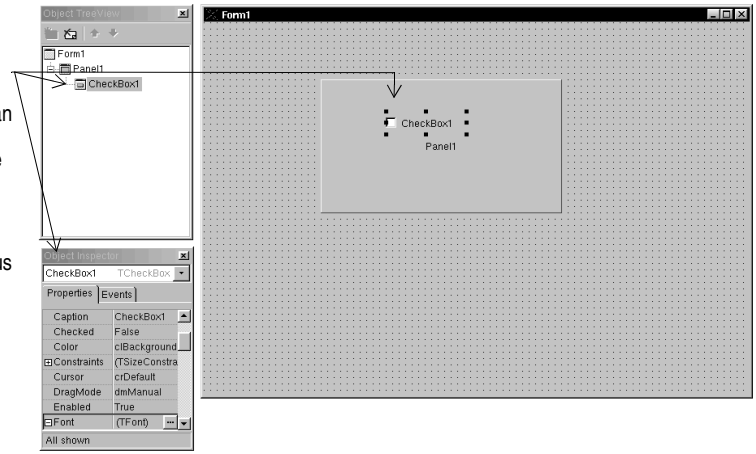
# The Object TreeView

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 CheckBox 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 CheckBox 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 double-click any object in the tree diagram to open the Code Editor to a place where you can write an event handler.

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




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

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

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

***For more information...***

See "Object TreeView" in the online Help index.

# The Object Repository

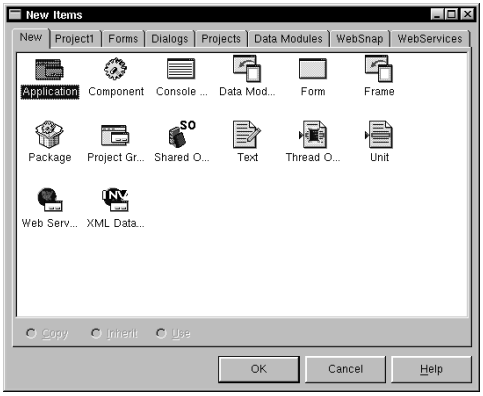
The Object Repository contains forms, dialog boxes, data modules, wizards, shared libraries, sample applications, and other items that can simplify development. Choose File | New | Other to display the New Items dialog box when you begin a

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.

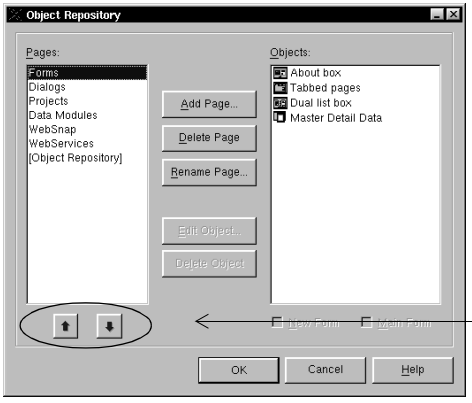
The Repository's tabbed pages include objects like forms, frames, units, and wizards to create specialized items.

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.

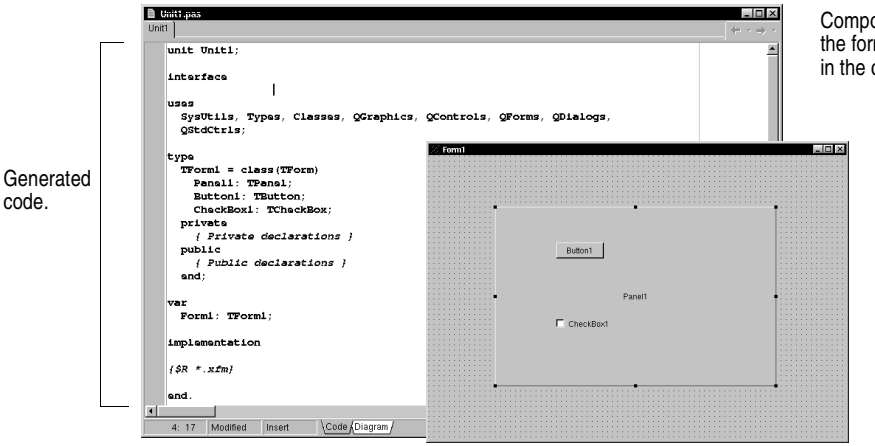
To add project and form templates to the Object Repository, see "Adding templates to the Object Repository" on page 4-9.

### *For more information...*

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

# The Code Editor

As you design the user interface for your application, Kylix generates the underlying 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. Kylix provides various aids to help you write code, including the Code Insight tools, class completion, and code browsing.

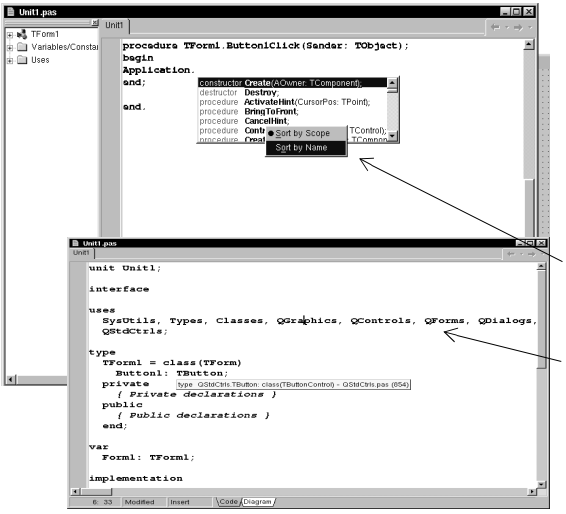



## Code Insight

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

| Tool | How it works |
|---|---|
| Code completion | For the C++ IDE, type the name of a variable that represents a pointer to an object followed by an arrow (->) or that represents a non-CLX object followed by a dot.<br>For the Delphi IDE, type a class name followed by a dot (.) to display a list of properties, methods, and events appropriate to the class, select it, and press *Enter*. In the **interface** section of your code you can select more than one item.<br>Type the beginning of an assignment statement and press *Ctrl+spacebar* 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. |

| Tool | How it works |
|---|---|
| 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 Kylix. |




With code completion, when you reference a member of an object (with a . in Delphi or a -> in C++) Kylix displays a list of properties, methods, and events for the class. As you type, the list automatically filters to the selection that pertains to that class. Select an item on the list and press *Enter* to add it to your code.

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

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

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

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

## D Class Completion for Delphi

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

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

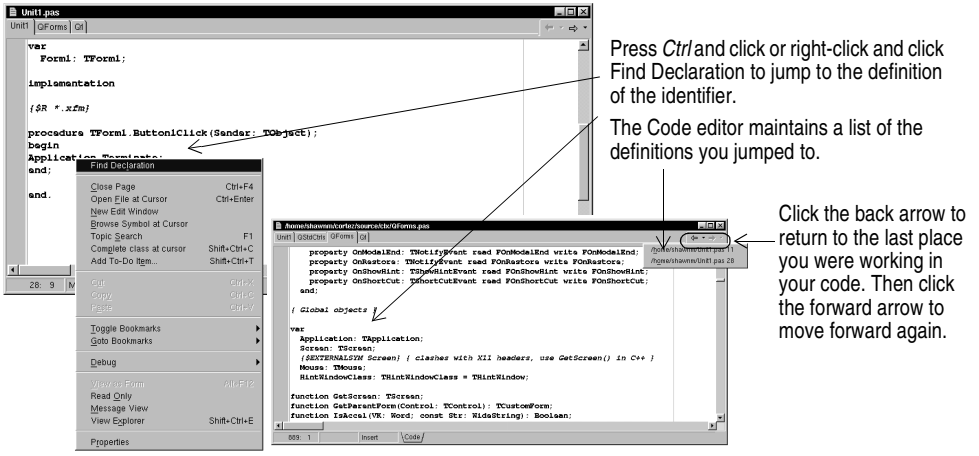
### *For more information...*

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

## Code Browsing

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

The Code Editor has forward and back buttons like the ones on Web browsers. As you jump to these definitions, the Code Editor keeps track of where you've been in the code. You can click the drop-down arrows next to the Forward and Back buttons to move forward and backward through a history of these references.




To customize your code editing environment, see "Customizing the Code Editor" on page 4-11.

### *For more information...*

See "Code Editor" in the online Help index.

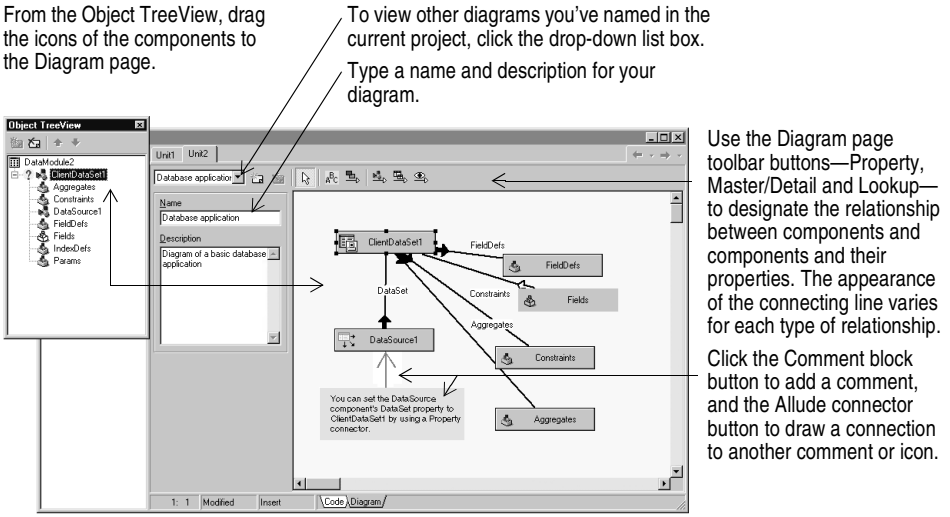
# The Diagram page

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

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

you add a dataset component to a data module and drag the dataset icon plus its property icons to the Diagram page, the property connector automatically connects the property icons to the dataset icon.

For components that don't have dependent relationships but where you want to show one, use the toolbar buttons at the top of the Diagram page to add one of four connector types, including allude, property, master/detail, and lookup. You can also add comment blocks that connect to each other or to a relevant icon.



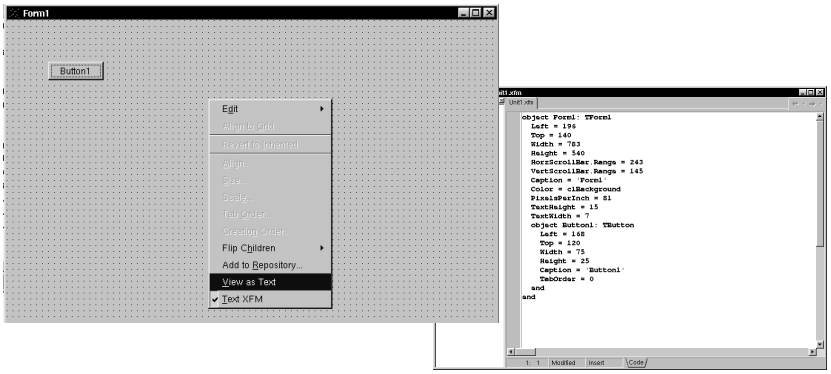

You can type a name and description for your diagram, save the diagram, and print it when you are finished.

***For more information...***

See "diagram page" in the online Help index.

## Viewing form code

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




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

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

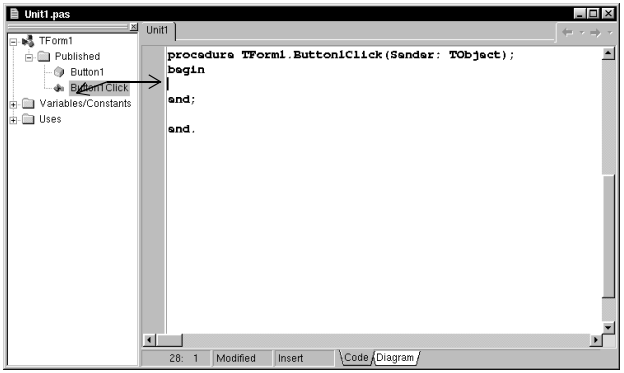
***For more information...***

See "form files" in the online Help index.

# D The Code Explorer

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

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




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

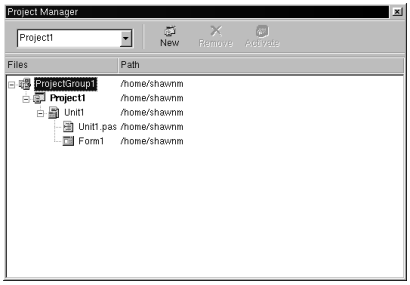
Each item in the Code Explorer has an icon that designates its type.

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

***For more information...***
See "Code Explorer" in the online Help index.

# The Project Manager

When you first start Kylix, it automatically opens a new project. A project includes several files that make up the application or shared object 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.

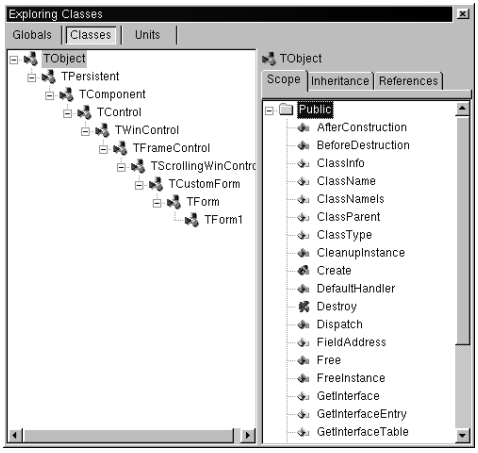



You can use the Project Manager to combine and display information on related projects into a single *project group*. By organizing related projects into a group, such as multiple executables, you can compile them at the same time. To change project options, such as compiling a project, see "Setting project options" on page 4-8.

***For more information...***
See "Project Manager" in the online Help index.

# D The Project Browser

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




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

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

Classes displays classes in a hierarchical diagram.

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

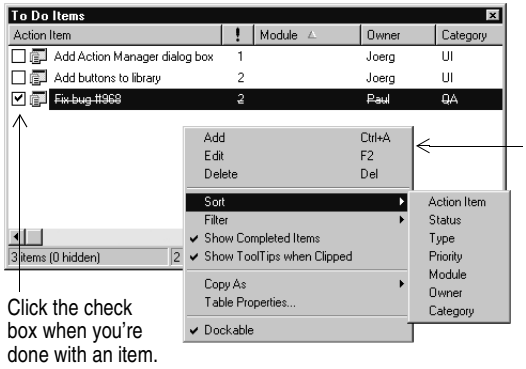
By default, the Project Browser displays the symbols from units in the current project only. You can change the scope to display all symbols available in Delphi. Choose Tools | Environment Options, and on the Explorer page, check All symbols.

***For more information...***

See “Project Browser” in the online Help index.

# To-do lists

To-do lists record items that need to be completed for a project. You can add project-wide items to a list by adding them directly to the list, or you can add specific items directly in the source code. Choose View | To-Do List to add or view information associated with a project.




***For more information...***

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

Chapter

3

# Programming with Kylix

The following sections provide an overview of software development with Kylix, 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 Kylix, a new project opens. The Delphi IDE automatically generates a project file (Project1.dpr), unit file (Unit1.pas), and resource file (Unit1.dfm; Unit1.xfm for CLX applications), among others. The C++ IDE produces Project1.bpr, Unit1.cpp, and Unit1.h along with the form files.

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 shared object. To read about what types of projects you can develop with Kylix, see "Types of projects" on page 3-8.

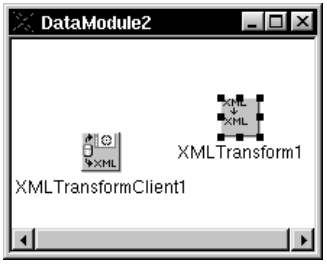
***For more information...***

See "projects" in the online Help index.

## Adding data modules

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

To create a data module, choose File | New | Data Module. Kylix opens an empty data module, which displays an additional unit file for the module in the Code EditorCode 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, Kylix displays its components.

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

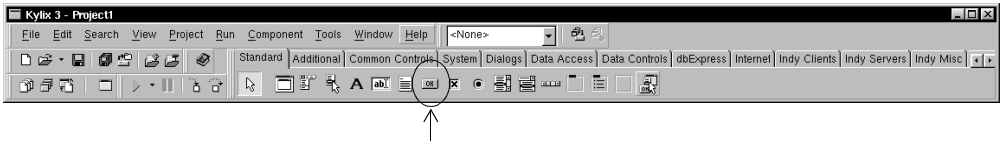
# Building the user interface

With Kylix, 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

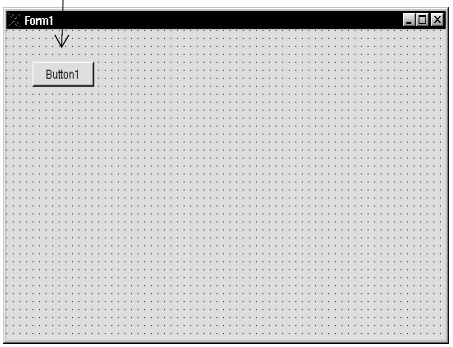
To place components on a form, do one of the following:

- Double-click the component.
- 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.
- Choose View | Component list from the main window, select a component, and click the Add to form button.

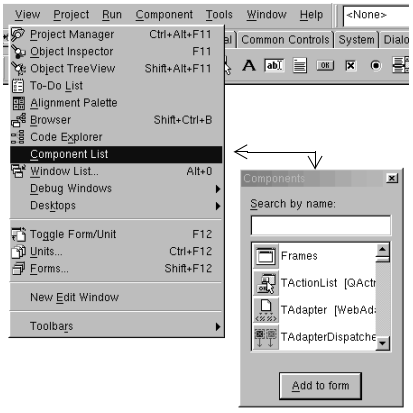



Click a component on the Component palette.

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




Or choose a component from an alphabetical list.

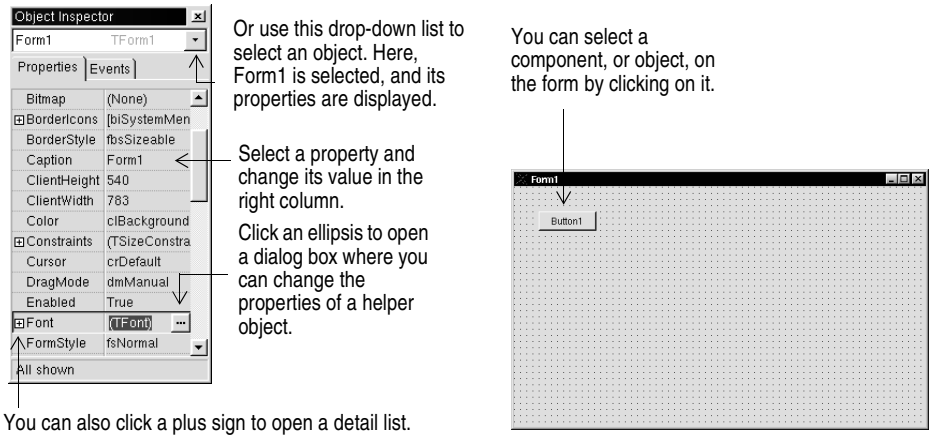



***For more information...***

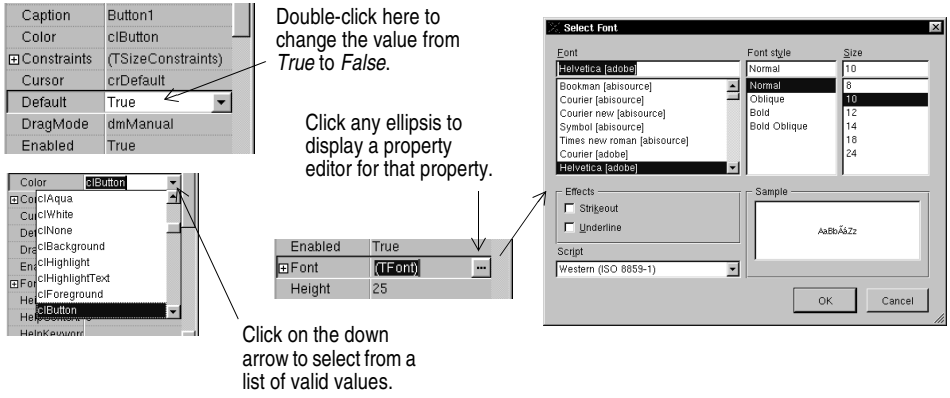
See “Component palette” in the online Help index.

## Setting component properties

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

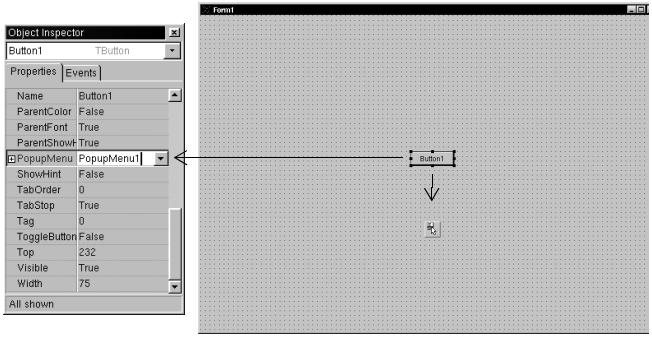



Many properties have simple values—such as names of colors, *True* or *False*, and integers. For Boolean properties, you can double-click the word to toggle between *True* and *False*. Some properties have associated property editors to set more complex values. When you click on such a property value,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




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

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




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

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

***For more information...***

See "Object Inspector" in the online Help index.

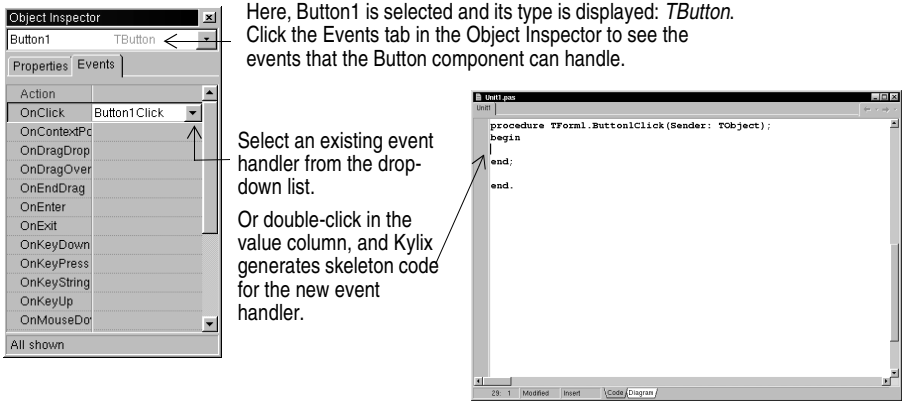
# Writing code

An integral part of any application is the code behind each component. While Kylix'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 Kylix's CLX class libraries. To work with your source code, see "The Code Editor" on page 2-7.

## Writing event handlers

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

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




***For more information...***

See "events" in the online Help index.

## Using the CLX libraries

Kylix comes with the Borland Component Library for Cross-Platform (CLX) which is made up of sublibraries of objects, some of which are also components or controls, that you use when writing code. 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. Objects descended from *TComponent* have properties and methods that allow them to be installed on the Component palette and added to Kylix forms and data modules. Because 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 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.

***For more information...***

See "CLX Reference" in the Help contents and in the online Help index.

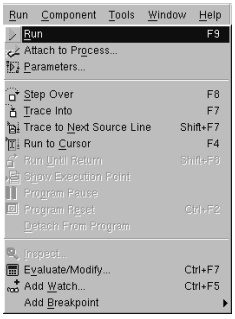
# Compiling and debugging projects

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

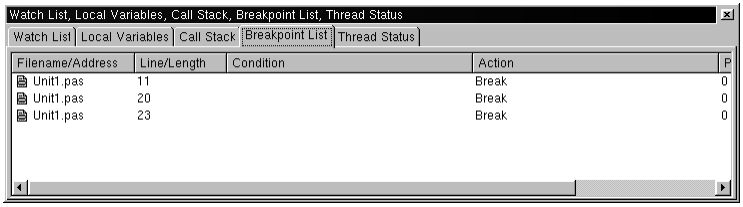
Kylix uses an integrated debugger so that you can control program execution, watch variables, and modify data values. You can step through your code line by line, examining the state of the program at each breakpoint. To use the integrated debugger, choose Tools | Debugger Options, click the General page, and make sure Integrated debugging is checked.

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




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


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




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 4-3.

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 4-5.

### *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, shared objects, package files, and helper applications.

***For more information...***

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

# Internationalizing applications

Kylix offers several features for internationalizing and localizing applications. The IDE and CLX support input method editors (IMEs) and extended character sets to internationalize your project.

***For more information...***

See "international applications" in the online Help index.

# Types of projects

All editions of Kylix support general-purpose 32-bit Linux programming, Shared Objects, packages, custom components, multithreading, and multiprocess debugging. Some editions support server applications such as Web server applications, database applications, 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.

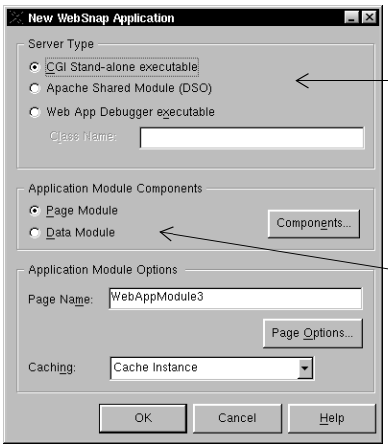
## Web server applications

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, Kylix includes two different technologies, depending on what edition of Kylix you have.

Kylix'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 Web Broker 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 Component palette page.

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.

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




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

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

***For more information...***

See “Web applications” in the online Help index.

## Database applications

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

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

## Custom components

The components that come with Kylix 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 Kylix 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 4-7.

***For more information...***
See Part V, “Creating custom components,” in the *Developer’s Guide* and “components, creating” in the online Help index.

## Shared objects

Shared Objects are compiled modules containing routines that can be called by applications and by other Shared objects. A Shared object contains code or resources typically used by more than one application. Choose File | New | Other and double-click the Shared object Wizard icon to create a template for a Shared Object.

***For more information...***
See “Shared objects” in the online Help index.

Chapter

4

# Customizing the desktop

This chapter explains some of the ways you can customize the tools in Kylix IDE.

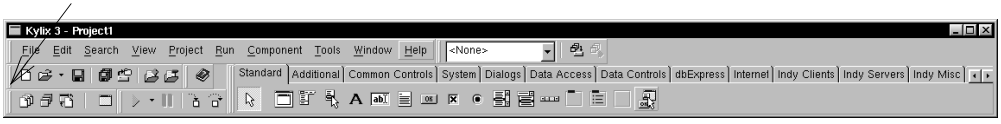
## Organizing your work area

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

### Arranging menus and toolbars

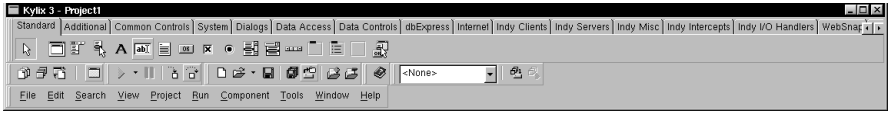
In the main window, you can reorganize the menu, toolbars, and Component palette by clicking the grabber on the left-hand side of each one and dragging it to another location.

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



Main window in its default arrangement.

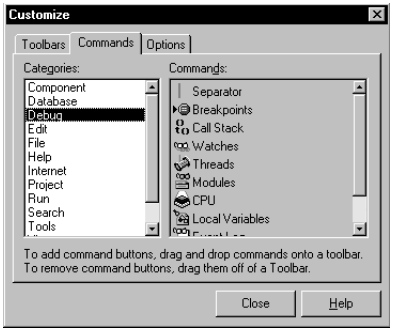

Main window organized differently




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 with parts separated.

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...*

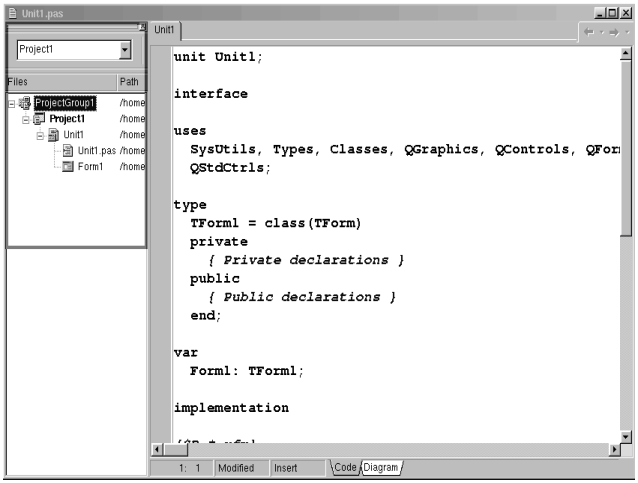
See “toolbars, customizing” in the online Help index.

## 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.

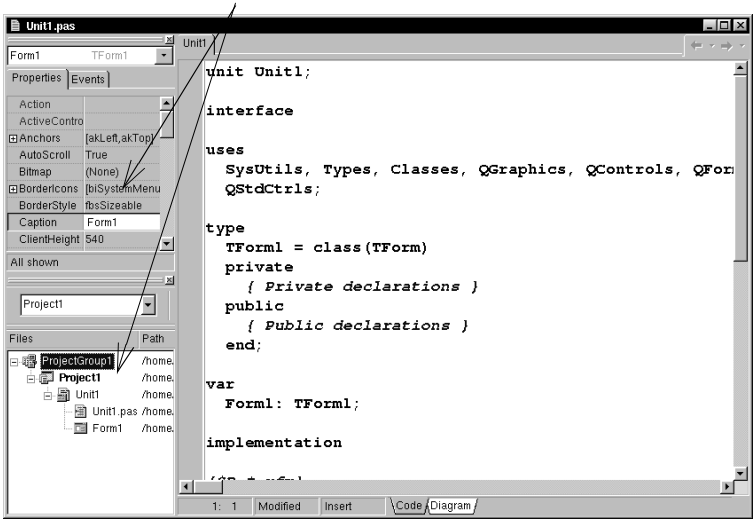
From the View menu, you can bring up any tool window and then dock it directly to another. For example, when you first open the Delphi Language IDE in its default configuration, the Code Explorer is docked to the left of the Code Editor. You can add the Project Manager to the first two to create three docked windows.

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



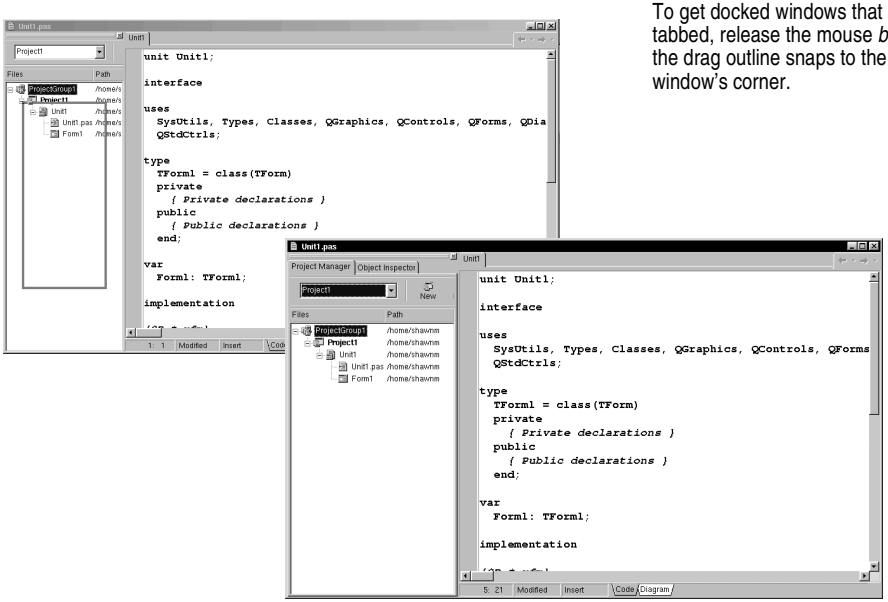
Here the Project Manager and Object Inspector are docked to the Code Editor.

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




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.You can also dock tools to form tabbed windows.

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



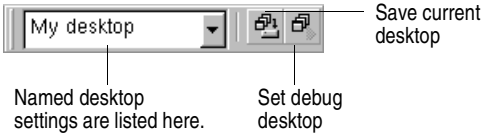

To undock a window, double-click its grabber or tab, or click and drag the tab outside of the docking area.

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.

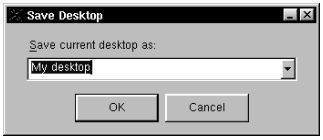
## Saving desktop layouts

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

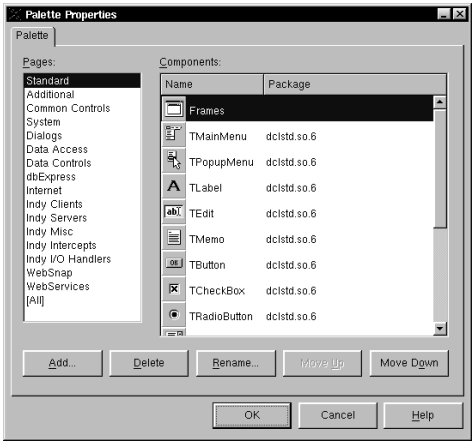
In its default configuration, the Component palette displays many useful CLX objects organized functionally onto tabbed pages. You can customize the Component palette by:

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

## Arranging the Component palette

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

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




You can rearrange the palette and add new pages.

***For more information...***

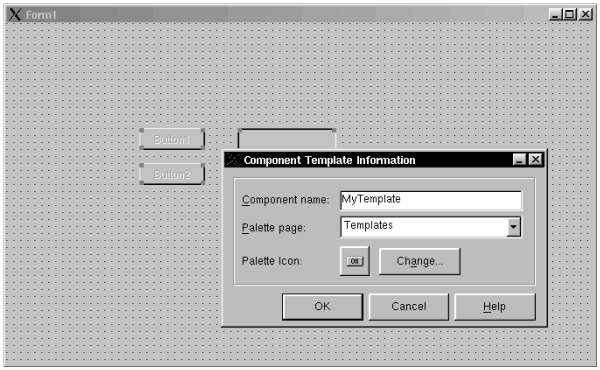
Click the Help button in the Palette Properties dialog box.

## Creating component templates

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

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

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




***For more information...***

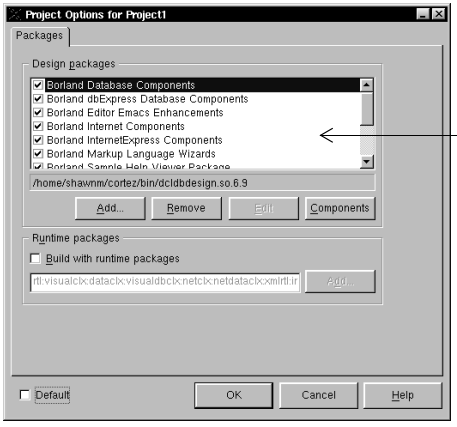
See “templates, component” in the online Help index.

## Installing component packages

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

A package is a special shared object containing code that can be shared among Kylix 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. Kylix 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 Kylix. When you install new components from third-party vendors, their package appears in this list.

Click Components to see what components the package contains.

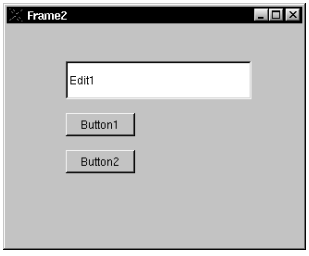
***For more information...***

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.

## 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.

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.

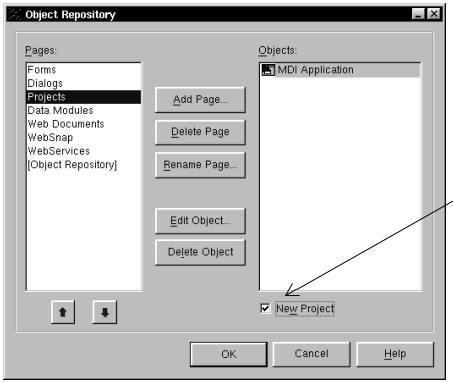
***For more information...***

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

# Specifying project and form templates as the default

When you choose File | New | Application, Kylix 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. Or you can choose from one of Kylix'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, Kylix 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, Kylix 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...*

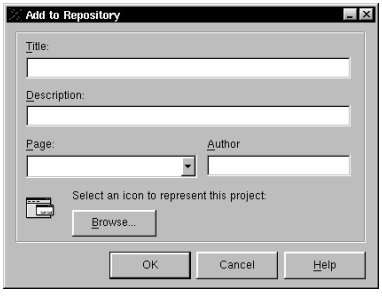
See "templates, adding to Object Repository," "projects, specifying default," and "forms, specifying default" in the online Help index.

## 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 Kylix, see "Specifying project and form templates as the default" on page 4-9.

***For more information...***

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

# Setting tool preferences

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

***For more information...***

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

## Customizing the Form Designer

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

***For more information...***

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

## Customizing the Code Editor

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

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

***For more information...***

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

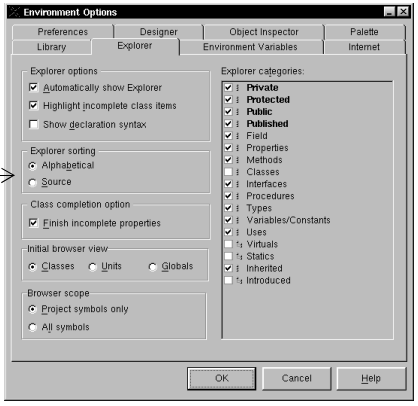
D

## Customizing the Code Explorer

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

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

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




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

***For more information...***

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

# Index