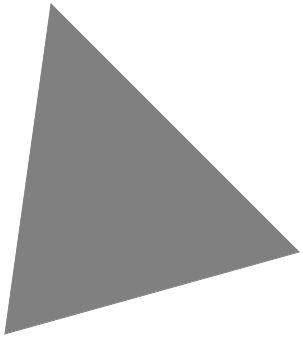
Quick Start



Borland®
Kylix™
Delphi™ for Linux®

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

Printed in the U.S.A.

HDB7010WW21000 1E0R0201
0001020304-9 8 7 6 5 4 3 2 1
PDF

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 desktop," 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 or shared object. Chapter 4, "Creating a text editor—a tutorial," takes you step by step through a tutorial to write a program for a text editor. Chapter 5, "Customizing the desktop" describes how you can customize the 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 32-bit Linux applications for Intel architecture with a minimum of manual coding. Kylix provides all the tools you need to develop, test, debug, and deploy applications, including a large library of reusable components, a suite of design tools, application templates, and programming wizards. These tools simplify prototyping and shorten development time.

Finding information

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

- Online Help
- Printed documentation
- Borland developer support services and 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 the core Help files listed in Table 1.1.

Table 1.1 Online Help files

Help file	Contents	Audience
Welcome to Kylix	Introduces the development environment and explains how to work with projects and forms. Discusses basic concepts of component-based object-oriented programming. Includes a step-by-step tutorial to help you learn Kylix.	All developers, people with questions about the IDE
Kylix Object and Component Reference (CLX)	Presents a detailed reference on CLX classes, global routines, types, and variables. Entries show the unit where each class is declared; its position in the hierarchy; a list of available properties, methods, and events; and code examples.	All Kylix developers using CLX
Programming with Kylix	Provides details about using CLX components and illustrates common programming tasks such as handling exceptions, creating toolbars and drag-and-drop controls, and using graphics.	All Kylix developers
Developing Database Applications	Explains design of client/server database applications, including database architecture, datasets, fields, tables, queries, and decision support.	Database developers
Writing Distributed Applications	Explains how to create distributed applications. Includes information on HTTP and sockets.	Developers writing client/server applications
Creating Custom Components	Provides information on writing custom Kylix components. Explains how to design, build, test, and install a component.	Developers writing Kylix components
Object Pascal Reference	Provides a formal definition of the Object Pascal language and includes topics on file I/O, string manipulation, program control, data types, and language extensions.	Developers who need Object Pascal language details

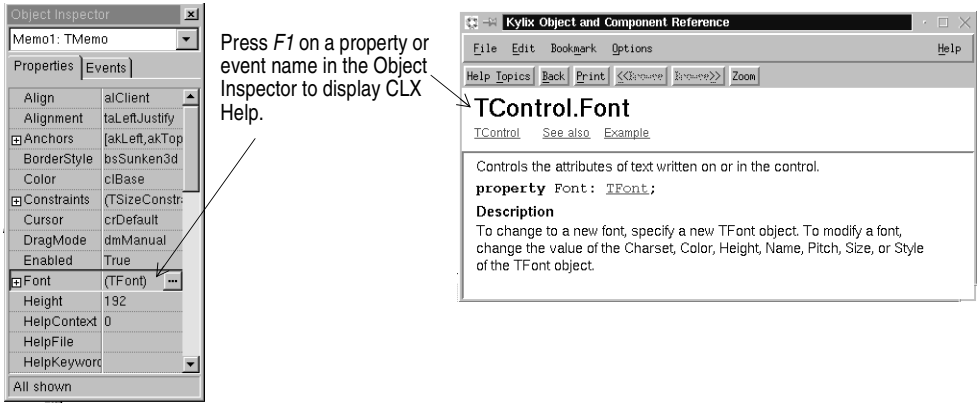
For more information...

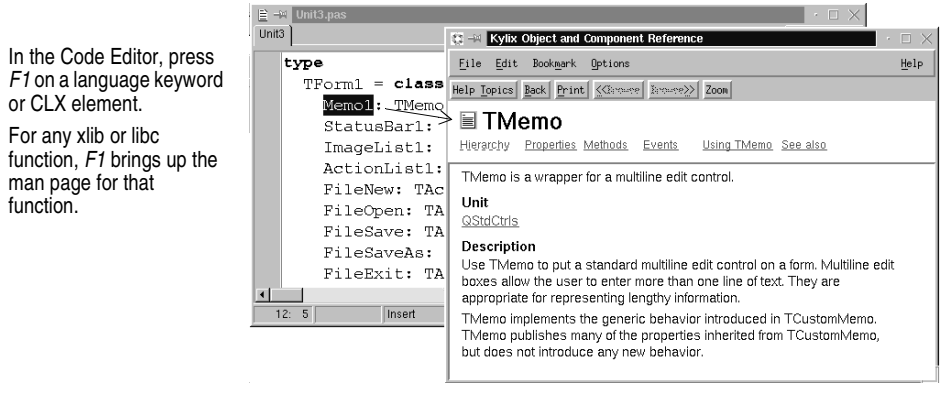
To open Help, choose Help | Kylix Help.

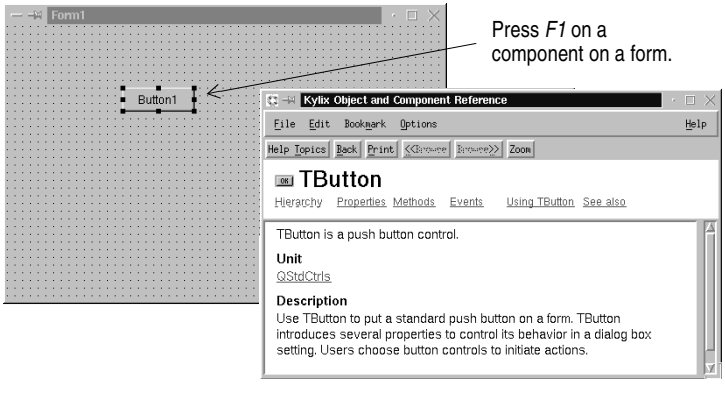
F1 Help

The online Help system provides extensive documentation on CLX and other parts of Kylix. You can get context-sensitive Help on any part of the development

environment, including menu items, dialog boxes, toolbars, and components by selecting the item and pressing *F1*.







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 Kylix. 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 Kylix developers exchange information, tips, and techniques. The site also includes a list of books about Kylix.

Typographic conventions

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

Table 1.2 Typographic conventions

Typeface	Meaning
`Monospace type`	Monospaced type represents text as it appears on screen or in code. It also represents anything you must type.
Boldface	Boldfaced words in text or code listings represent reserved words or compiler options.
Italics	Italicized words in text represent 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."

Chapter

2

A tour of the desktop

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

Starting Kylix

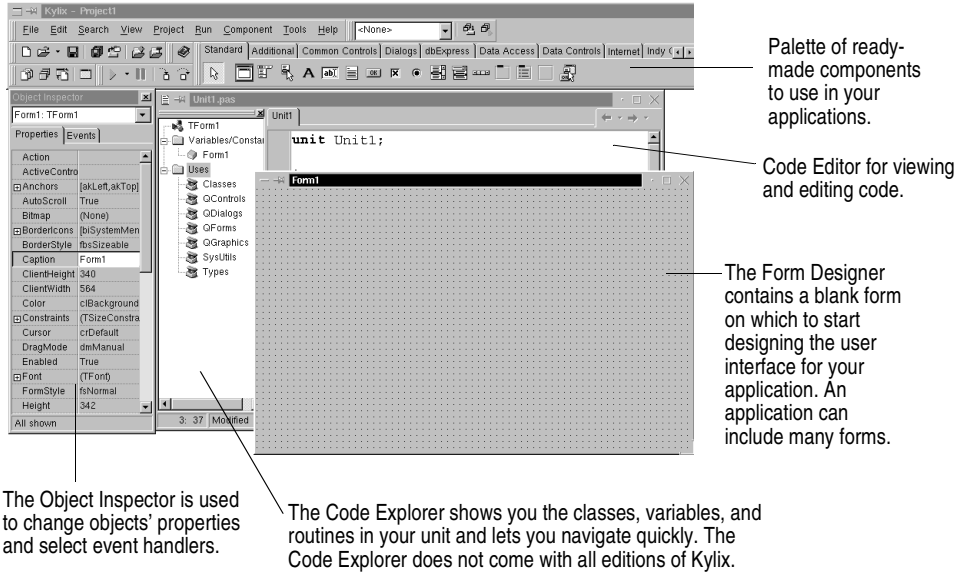
You can start Kylix in the following ways:

- From a shell window, enter `{install directory}/bin/startkylix`. For example, if your install directory is in the /root directory, enter: `/root/kylix/bin/startkylix`.
- From the Application Starter menu, choose Borland Kylix | Kylix or Personal | Borland Kylix | Kylix.

Getting started: The IDE

When you first start Kylix, you'll see some of the major tools in the IDE. In Kylix, the IDE includes the toolbars, menus, Component palette, Object Inspector, Code Editor, Code Explorer, Project Manager, and many other tools. The particular features and

components available to you will depend on which edition of Kylix you've purchased.



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 manage projects, design graphical interfaces, write code, compile, test, debug, and browse through class libraries without leaving the IDE.

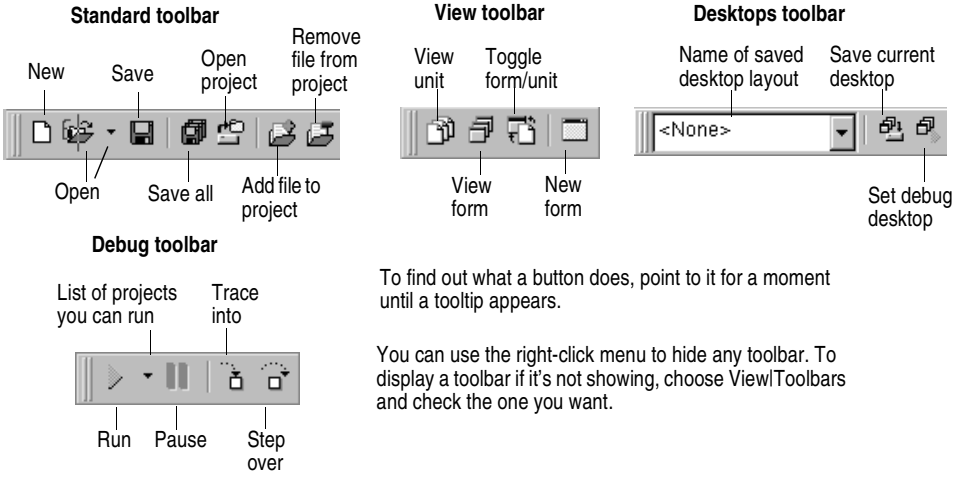
To learn about organizing and configuring the IDE, see Chapter 5, "Customizing the desktop."

Controlling Kylix: The menu and toolbars

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

Main window in its default arrangement.

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



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

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

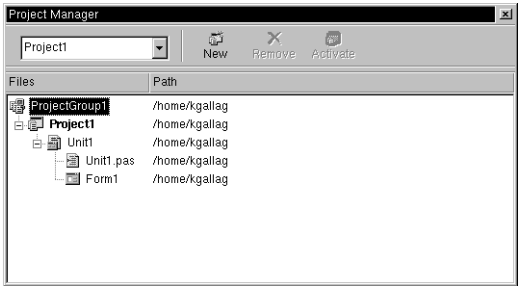
Many operations have keyboard shortcuts as well as toolbar buttons. When a keyboard shortcut is available, it is always shown next to the command on the drop-down menu.

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

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

Managing projects: The Project Manager

When you first start Kylix , it automatically opens a new project, as shown on page 2-2. 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 5-8.

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

Browsing project structure and elements: The Project Browser

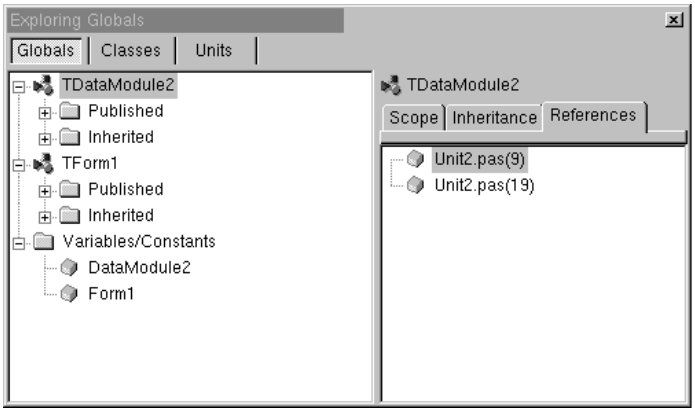
The Project Browser examines a project in detail. The Browser displays classes, units, and global symbols (types, properties, methods, variables, and routines) your project declares or uses in a tree diagram. Choose View | Browser to display the Project Browser.

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

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

Classes displays classes in a hierarchical diagram.

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



You can change the way the contents are grouped within the diagram by right-clicking in the Browser, 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.

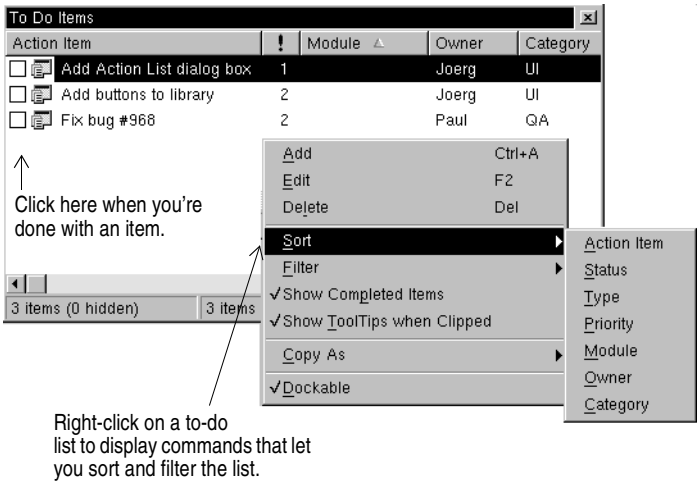
By default, the Project Browser displays the symbols from units in the current project only. You can change the scope to display all symbols available in Kylix. Choose Tools | Environment Options, and on the Explorer page, check All symbols (CLX included).

For more information...
See "Project Browser" in the Help index.

Creating to-do lists

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

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



For more information...

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

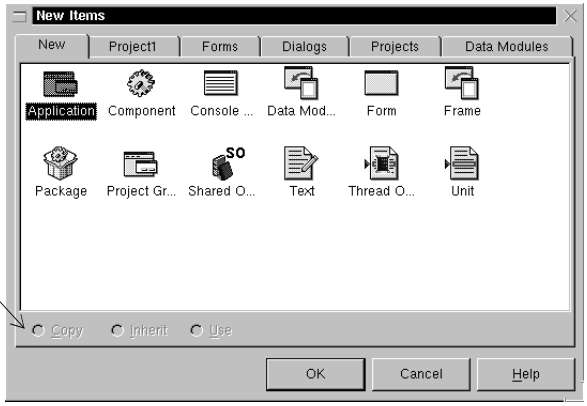
Adding items to your projects: The Object Repository

The Object Repository contains forms, dialog boxes, data modules, wizards, shared object files, sample applications, and other items that can simplify development. Choose File | New 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:

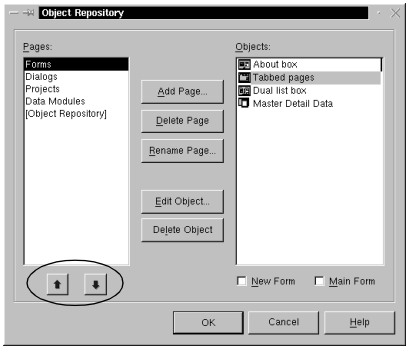
Copy (the default) creates a copy of the item in your project. *Inherit* means changes to the object in the Repository are inherited by the one in your project. *Use* means changes to the object in your project are inherited by the object in the Repository.



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.

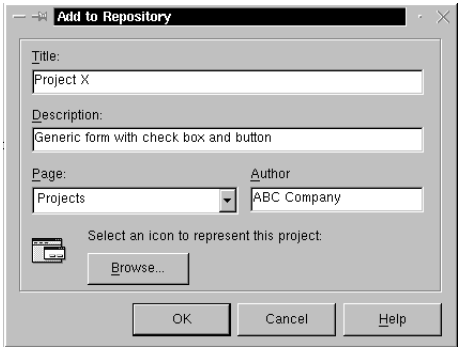
Click the arrows to change the order in which a tabbed page appears in the New Items dialog box.



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.

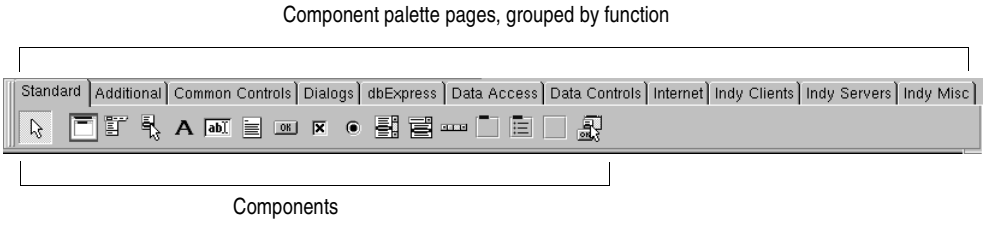
The next time you open the New Items dialog box, your project template will appear on the Projects page (or the page to which you had saved it). To make your template the default every time you open Kylix, see "Specifying project and form templates as the default" on page 5-9.

For more information...

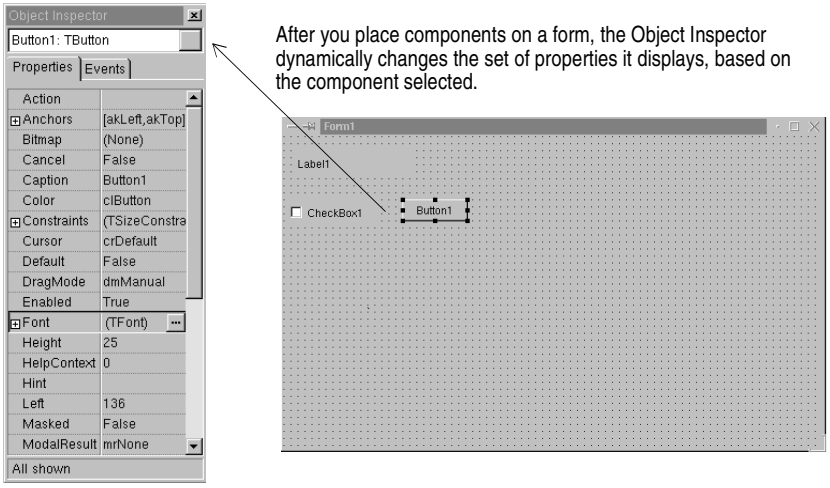
See "Object Repository" in the Help index. The objects available to you will depend on which edition of Kylix you purchased.

Building the user interface: The Form Designer, Component palette, and Object Inspector

The Component palette includes tabbed pages with groups of icons representing visual or nonvisual CLX components you use to design your application interface. The pages divide the components into various functional groups. For example, 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. Use the Form Designer to 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 3-3.



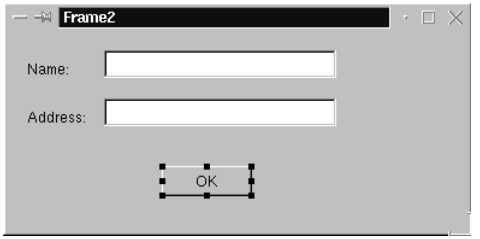
For more information...

See "Component palette" and "Object Inspector" in the 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.

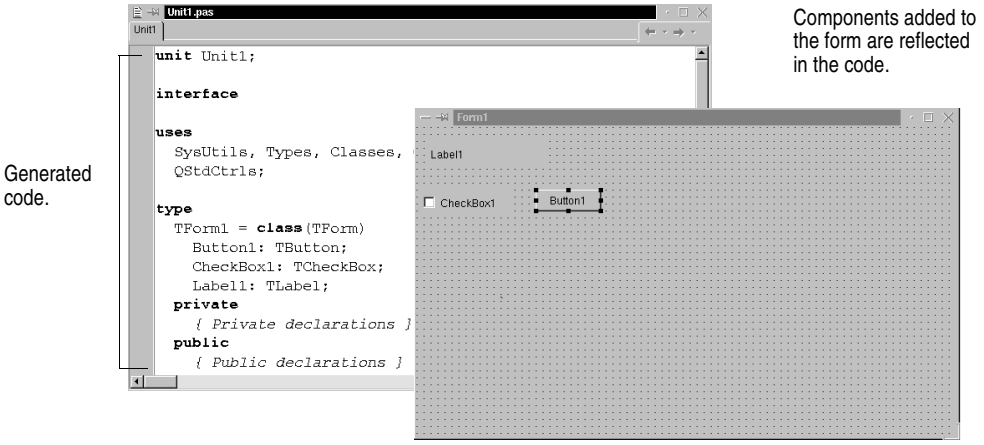
For more information...

See “frames” and “TFrame” in the Help index.

Viewing and editing code: The Code Editor and Code Explorer

As you design the user interface for your application, Kylix generates the underlying Object Pascal code. When you select and modify the properties of forms and components, 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.



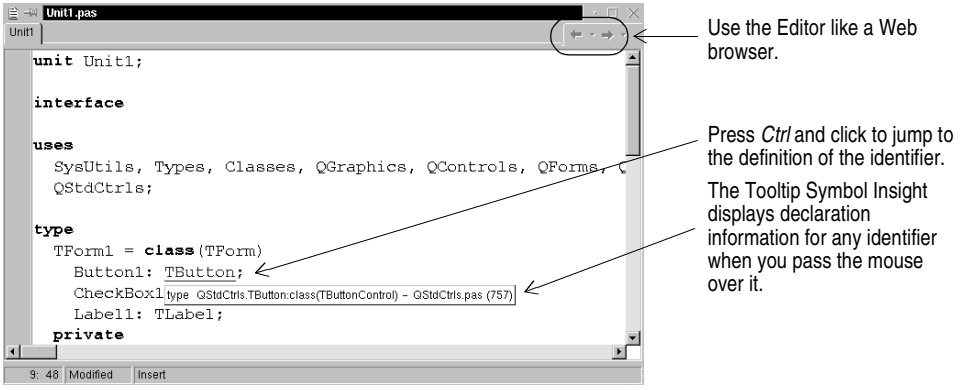
Generated code.

Components added to the form are reflected in the code.

To customize your code editing environment, see "Customizing the Code Editor" on page 5-10.

Browsing with the Code Editor

The Code Editor has forward and back buttons like the ones you've seen on Web browsers. You can use them to navigate through source code. First press *Ctrl* and point to any identifier. The cursor turns into a hand, and the identifier turns blue and is underlined. Click to jump to the definition of the identifier. Click the left, or back arrow, to return to the last place you were working in your code. Then click the right, or forward arrow, to move forward again.



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

For more information...

See "Code Editor" in the Help index.

Getting help with your code

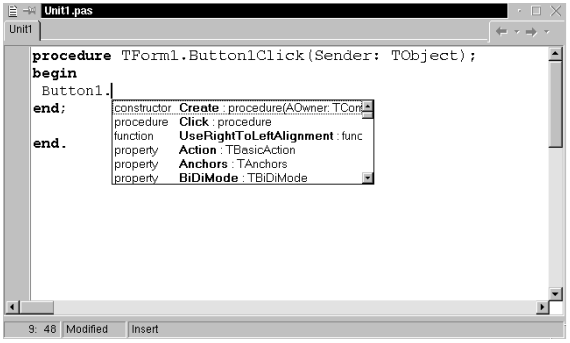
Kylix provides various aids to help you write code. The Code Insight tools display context-sensitive pop-up windows.

Table 2.1 Code Insight tools

Tool	How it works
Code Completion	Type a class name followed by a dot (.) to display a list of properties, methods, and events appropriate to the class. 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.

Table 2.1 Code Insight tools (continued)

Tool	How it works
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.
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.



When you type the dot in `Button1.` 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.

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

Class Completion

Class Completion generates skeleton code for classes. Place the cursor anywhere within a class declaration; then press *Ctrl+Shift+C*, or right-click and select Complete Class at Cursor. 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 configure Class Completion, choose Tools | Environment Options and click the Explorer tab.

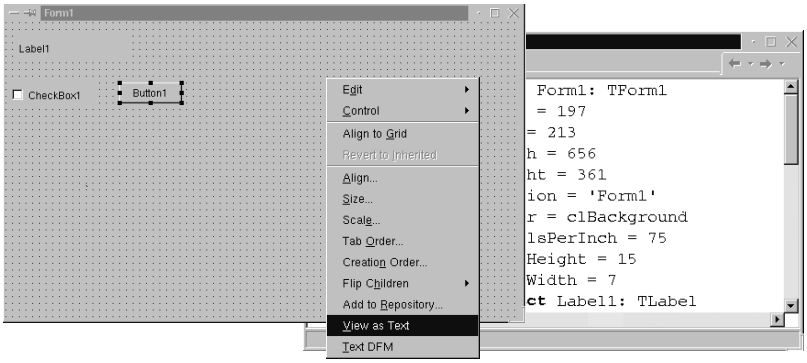
For more information...

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

Viewing and editing 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 (.xfm) 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. Use the Environment Options dialog box to designate which format to use for newly created forms.

For more information...

See "form files" in the Help index.

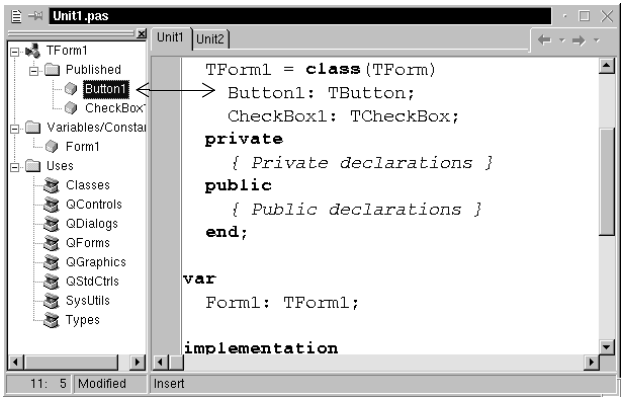
Viewing code with the Code Explorer

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

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

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

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



For more information...

See "Code Explorer" in the 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 programs.

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. It automatically generates a project file (Project1.dpr), unit file (Unit1.pas), and resource file (Unit1.xfm), among others.

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

For more information...
See "projects" in the Help index.

Types of projects

All editions of Kylix support general-purpose Linux programming for writing a variety of GUI applications, shared objects, packages, and other programs. Some editions support server applications such as distributed applications, Web-based applications, and database applications.

For more information...
See Chapter 4, "Building applications and shared objects," in the *Developer's Guide* and "distributed applications" in the Help index.

Database applications

For use in database applications, Kylix uses a new data access technology, *dbExpress*. dbExpress is a collection of drivers that applications use to access data in databases. Kylix has drivers for four SQL databases, including DB2, InterBase, MySQL, and Oracle, depending on which edition you have.

To access the data, you can add dbExpress components to data modules or forms. These components include a connection component, which controls information you need to connect to a database, and dataset components, which represent the data fetched from the server.

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

For more information...
See Part II, "Developing database applications," in the *Developer's Guide* and "database applications" in the Help index.

Web server applications

Web server applications extend the functionality of existing Web servers. A Web server application receives HTTP request messages from the Web server, performs any actions requested in those messages, and formulates responses that it passes back to the Web server. Any operation that you can perform with a Kylix application can be incorporated into a Web server application.

To create a Web server application, choose File | New and double-click the Web Server Application icon in the New Items dialog box. When the New Web Server Application dialog box appears, select one of two Web server application types: CGI stand-alone executable or an Apache Shared Module (DSO). Either option creates a new project with an empty Web module and is configured to use Internet components.

The Web server application tools are not available in all editions of Kylix.

For more information...
See Chapter 23, "Creating Internet server applications" in the in the *Developer's Guide* and "Web server applications" in the 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.

For more information...
See “shared objects” in the 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 see how to install components provided by a third party, see “Installing component packages” on page 5-7.

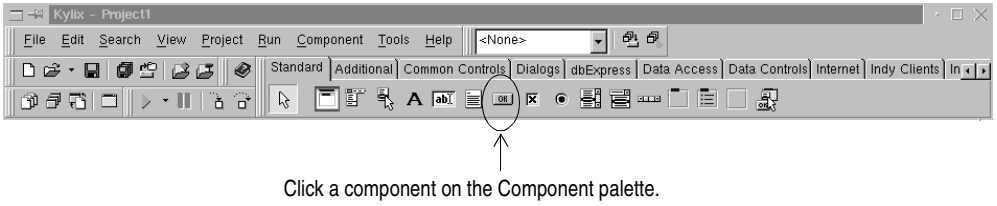
For more information...
See Part IV, “Creating custom components,” in the *Developer’s Guide* and “components, creating” in the Help index.

Building the user interface

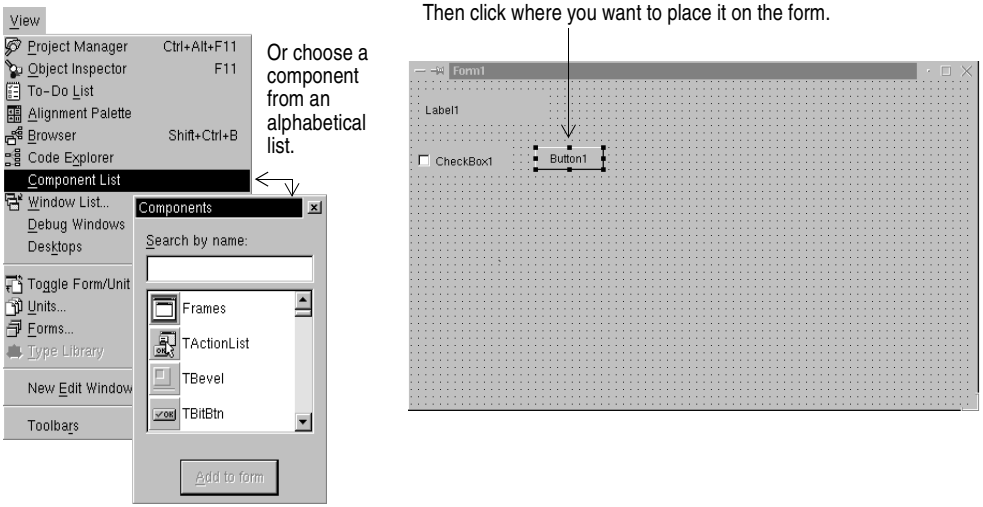
With Kylix, you first create a user interface (UI) by selecting components from the Component palette and placing them on the main form.

Placing components on a form

To place components on a form, either double-click the component or click the component once and then click the form where you want the component to appear.

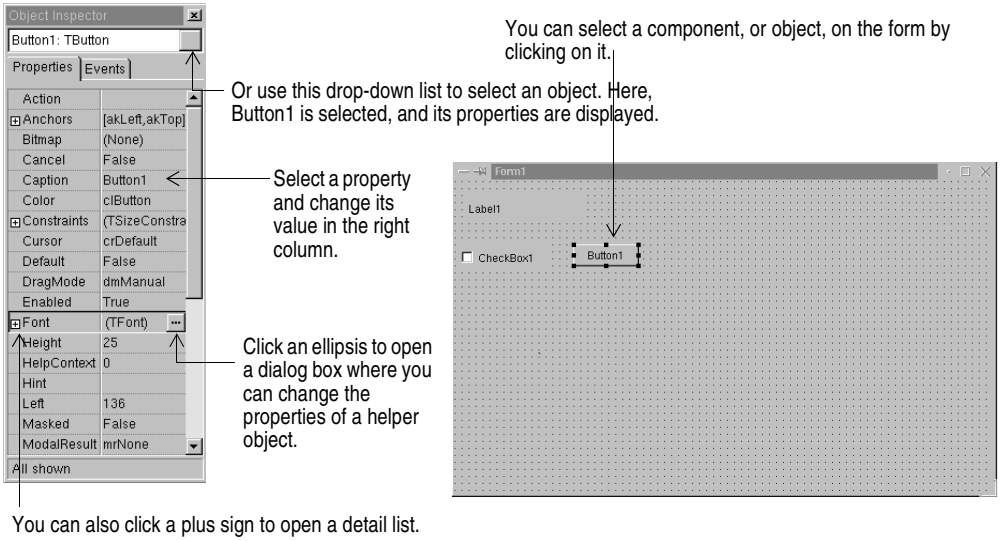


Select the component and drag it to wherever you want on the form.



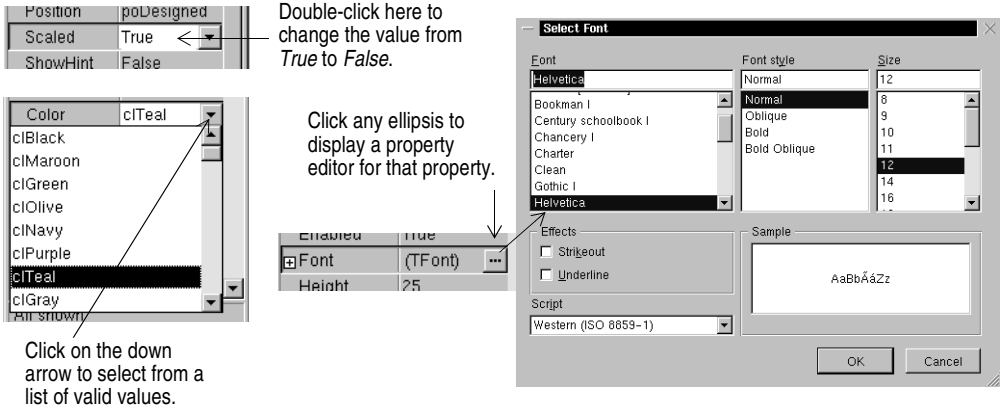
Setting component properties

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.

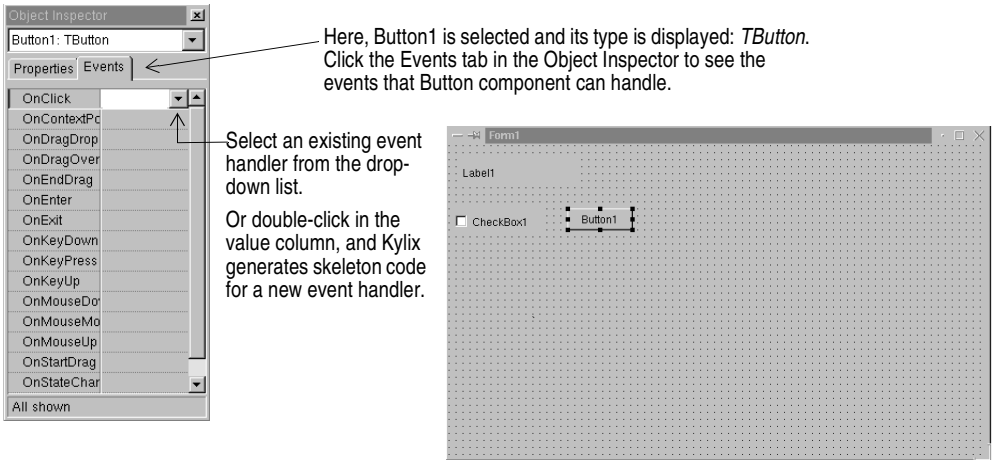
Writing code

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 prepackaged visual and nonvisual components, you will usually need to write event handlers and perhaps some of your own classes. To help you with this task, you can choose from Kylix's CLX class library of nearly 750 objects. To view and edit your source code, see "Viewing and editing code: The Code Editor and Code Explorer" on page 2-8.

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.



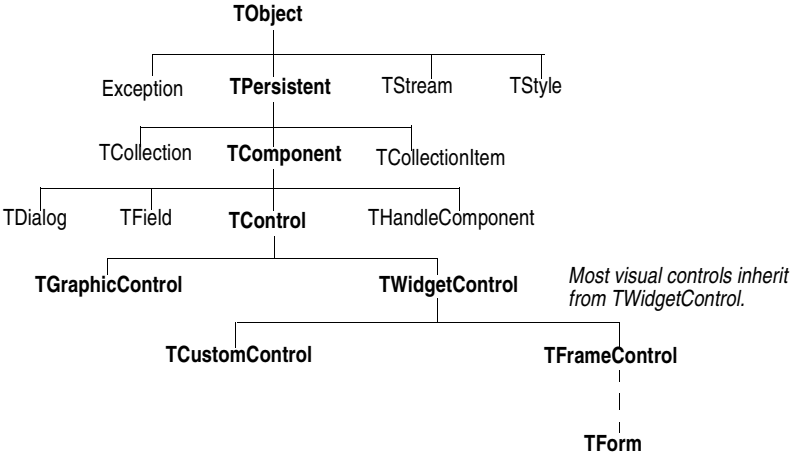
For more information...

See "events" in the Help index.

Using CLX classes

Kylix comes with a class library made up of objects, some of which are also components or controls, that you use when writing code. This class hierarchy, called the Borland Component Library for Cross Platform (CLX), includes objects that are visible at runtime—such as edit controls, buttons, and other user interface elements—as well as nonvisual controls like datasets and timers.

The diagram below shows some of the principal classes that make up CLX.



Objects descended from *TComponent* have properties and methods that allow them to be installed on the Component palette and added to Kylix forms. 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.

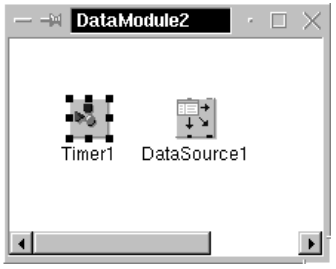
Most editions of Kylix come with complete CLX source code. In addition to supplementing the online documentation, CLX source code provides invaluable examples of Object Pascal programming techniques.

For more information...
See "Kylix Object and Component Reference" in the Help contents. See http://www.borland.com/kylix for open source and licensing options on CLX.

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 and in the Object Repository, double-click the Data Module icon. Kylix 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.



Click a nonvisual component from the Component palette and click in the data module to place the component.

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

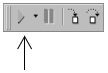
For more information...
See "data modules" in the Help index.

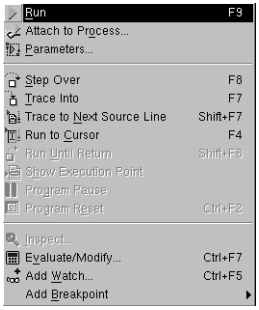
Compiling and debugging projects

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.

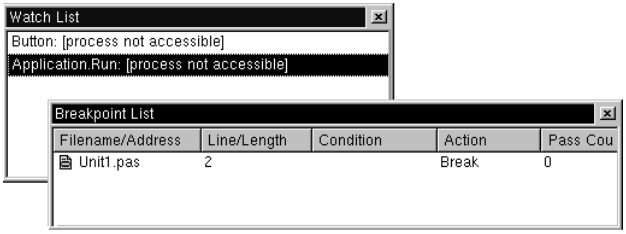
You can begin a debugging session in the IDE by choosing Run | Run or pressing F9.





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.



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

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

For more information...

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

Deploying programs

You can make your application available for others to install and run by deploying it. To deploy an application, create an installation package that includes not just the required files, such as the executables, but also any supporting files, such as shared object files, initialization files, package files, and helper applications.

For more information...

See “deploying” in the Help index.

Internationalizing applications

Kylix offers several features for internationalizing and localizing applications for different locales. The IDE and CLX provides support for input method editors (IMEs) and extended character sets. Once your application is internationalized, you can create localized versions for the different foreign markets into which you want to distribute it.

Kylix provides a tool called *resbind* that extracts the Borland resources from your application and creates a shared object file that contains the resources. You can then dynamically link the resources at runtime or let the application check the environment variable on the local system on which it is running. To get the maximum benefit from these features, start thinking about localization requirements as early as possible in the development process.

For more information...

See “international applications” in the Help index.

Chapter

4

Creating a text editor—a tutorial

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

Note This tutorial is for all editions of Kylix.

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 home directory.
2. Create a new project.

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

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

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

   Each form has its own unit (*Unit1.pas*) and form (*Unit1.xfm*) files. If you create a second form, a second unit (*Unit2.pas*) and form (*Unit2.xfm*) file are automatically created.
3. Choose File | Save All to save your files to disk. When the Save dialog box appears:
   - Navigate to your TextEditor folder.

- Save Unit1 using the default name Unit1.pas.
- Save the project using the name TextEditor.dpr. (The executable will be named the same as the project name without an extension.)

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

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

When you open a new project, Kylix 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.

You can run the form anytime by pressing *F9*.

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

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

To return to the design-time view of Form1, either:

- 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;
- Choose Run | Program Reset; or
- Choose View | Forms, select Form1, and click OK.

Setting property values

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

You can change the caption of *Form1* right away:

- 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.

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.

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

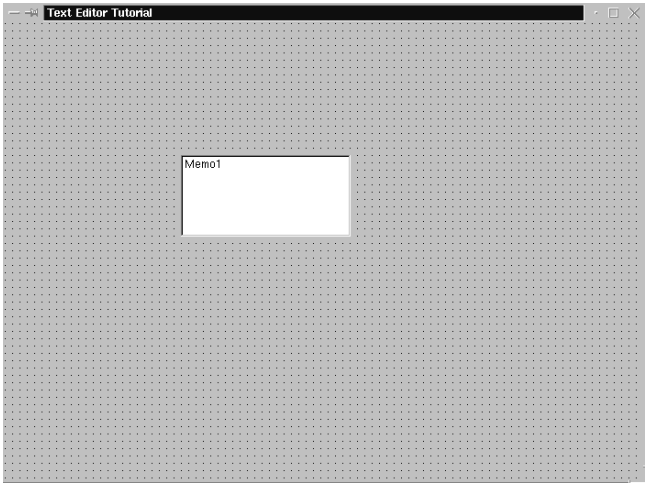
The text editor application requires an editing area, a status bar for displaying information such as the name of the file being edited, menus, and perhaps a toolbar with icons for easy access to commands. The beauty of designing the interface using Kylix is that you can experiment with different components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

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

1 To create a text area, first add a *Memo* component. To find the *Memo* component, on the Standard tab of the Component palette, point to an icon on the palette for a moment; Kylix displays a Help tooltip showing the name of the component.

When you find the *Memo* component, either:

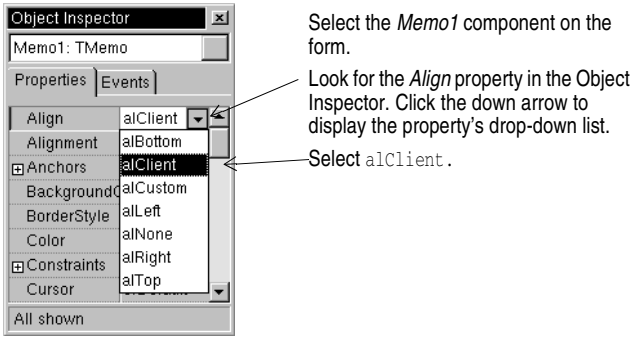
- Select the component on the palette and click on the form where you want to place the component; or
- Double-click it to place it in the middle of the form.



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

2 Set the *Align* property of *Memo1* to *alClient*.

To do this, click *Memo1* to select it on the form, then choose the *Align* property in the Object Inspector. Select *alClient* from the drop-down list.

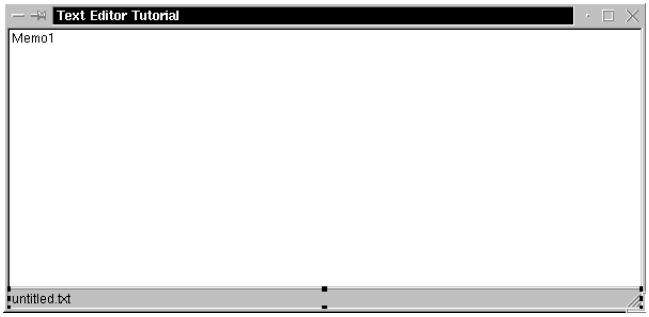


The *Memo* component now fills the entire form so you have a large text editing area. By choosing the *alClient* value for the *Align* property, the size of the *Memo* control will vary to fill whatever size window is displayed even if the form is resized.

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

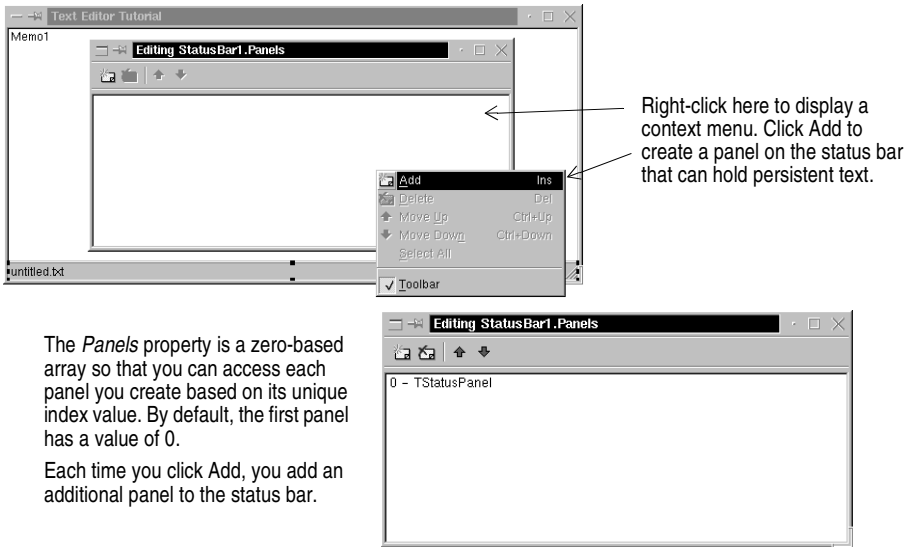
4 Next you want to create a place on the status bar to display the path and file name of the file being edited by your text editor. The easiest way is to provide one status panel.

- Change the *SimpleText* property to `untitled.txt`. If the file being edited is not yet saved, the name will be untitled.txt.



- Click the ellipse of the *Panels* property to open the Editing StatusBar1.Panels dialog box.

- Right-click the dialog box and click Add to add a panel to the status bar.



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

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

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

5 Click the **X** in the upper right corner 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, Kylix provides an *action list* to help centralize the code.

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

Table 4.1 Planning Text Editor commands

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

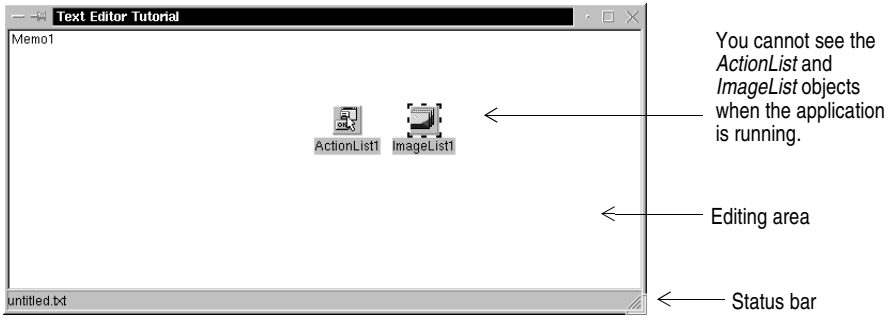
You can also centralize images to use for your toolbar and menus in an image list.

To add an *ActionList* and an *ImageList* component to your form:

1 On the Standard page of the Component palette, double-click the *ActionList* component to drop it onto the form.

2 On the Common Controls page, double-click the *ImageList* component to drop it onto your form. It drops on top of the *ActionList* component so drag it to another location on the form. Both the *ActionList* and *ImageList* components are nonvisual, so it doesn't matter where you put them on the form. They won't appear at runtime.

Your form should now resemble the following figure.



Adding actions to the action list

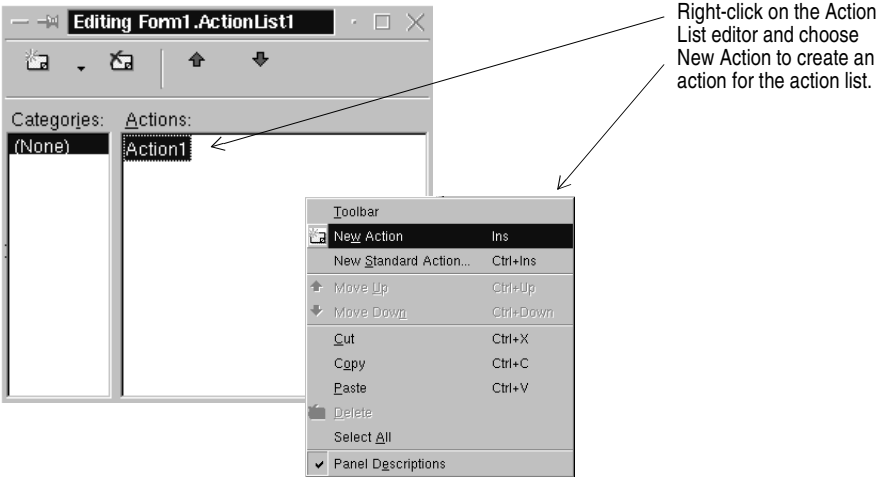
Next we'll add the actions to the action list.

Tip By convention, we'll name actions that are connected to menu items with the name of the top-level menu and the item name. For example, the FileExit action refers to the Exit command on the File menu.

1. Double-click the ActionList icon.

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

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

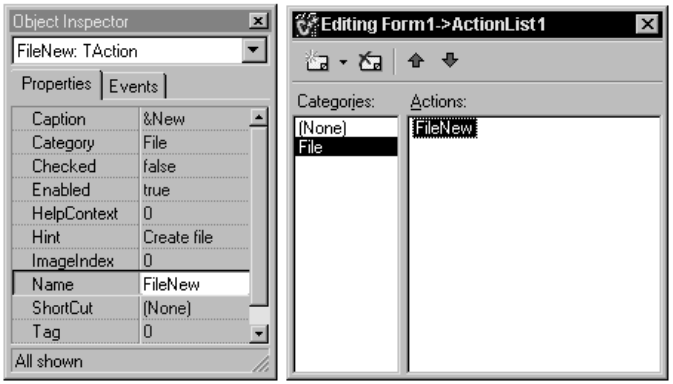


3. In the Object Inspector, set the following properties for the action:
   - After *Caption*, type `&New`. Note that typing an ampersand before one of the letters makes that letter a shortcut to accessing the command.
   - After *Category*, type `File` (this organizes the File commands in one place).
   - After *Hint*, type `Create file` (this will be the Help tooltip).
   - After *ImageIndex*, type `0` (this will associate image number 0 in your ImageList with this action).

- After *Name*, type `FileNew` (for the File | New command) and press *Enter* to save the change.

With the new action selected in the Action List editor, change its properties in the Object Inspector.

Caption is used in the menu, *Category* is the type of action, *Hint* is a Help tooltip, *ImageIndex* lets you refer to a graphic in the ImageList, and *Name* is what it's called in the code.



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

5 In the Object Inspector, set the following properties:

- After *Caption*, type `&Open`.
- Make sure *Category* says `File`.
- After *Hint*, type `Open file`.
- After *ImageIndex*, type `1`.
- After *Name*, enter `FileOpen` (for the File | Open command).

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

7 In the Object Inspector, set the following properties:

- After *Caption*, type `&Save`.
- Make sure *Category* says `File`.
- After *Hint*, type `Save file`.
- After *ImageIndex*, type `2`.
- After *Name*, enter `FileSave` (for the File | Save command).

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

9 In the Object Inspector, set the following properties:

- After *Caption*, type `Save &As`.
- Make sure *Category* says `File`.
- After *Hint*, type `Save file as`.
- No *ImageIndex* is needed. Leave the default value.
- After *Name*, enter `FileSaveAs` (for the File | Save As command).

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

11 In the Object Inspector, set the following properties:

- After *Caption*, type `E&xit`.
- Make sure *Category* says `File`.
- After *Hint*, type `Exit application`.

- After *ImageIndex*, type `3`.
- After *Name*, enter `FileExit` (for the File | Exit command).

12 Right-click on the Action List editor and choose New Action to create a Help | About command.

13 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

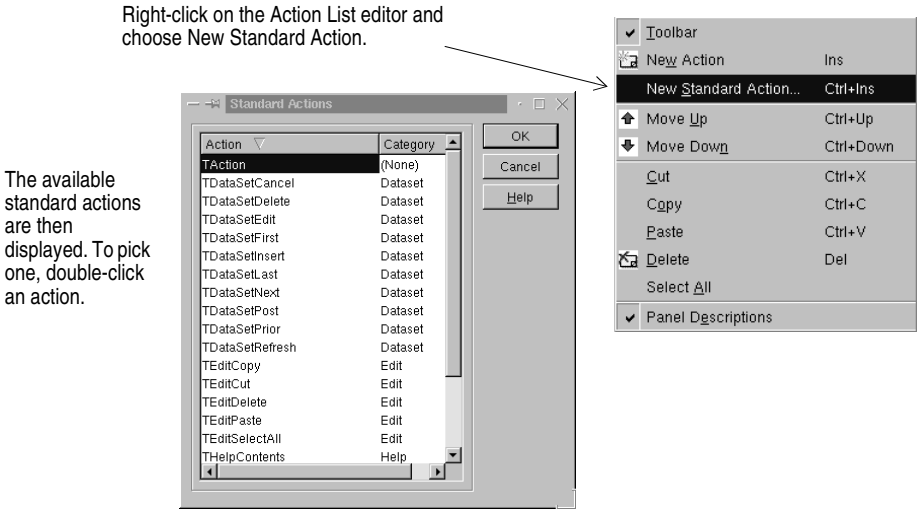
Kylix provides several standard actions that are often used when developing applications. Next we'll add the standard actions (cut, copy, and paste) to the action list.

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

To add standard actions to the action list:

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

The Standard Actions dialog box appears.



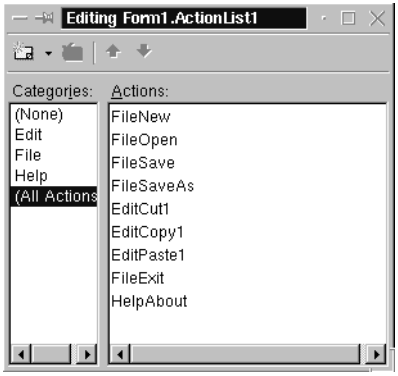
- Double-click TEditCut. The action is created in the Editing Form1.ActionList1 dialog box along with a new category called Edit. Select EditCut1.
- In the Object Inspector, set the *ImageIndex* property to `4`.

The other properties are set automatically.

2 Right-click on the Action List editor and choose New Standard Action.
   - Double-click TEditCopy.
   - In the Object Inspector, set the *ImageIndex* property to `5`.

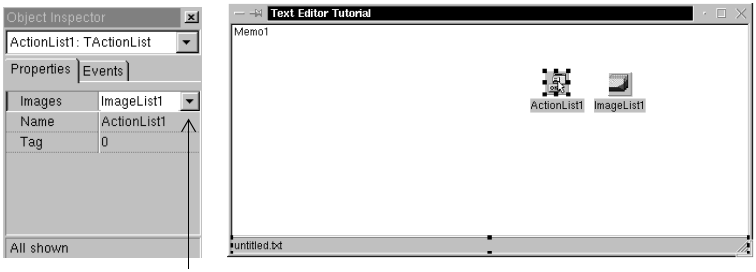
3 Right-click on the Action List editor and choose New Standard Action.
   - Double-click TEditPaste.
   - In the Object Inspector, set the *ImageIndex* property to `6`.

4 Now you've got all the actions that you'll need for the menus and toolbar. If you click the category All Actions, you can see all the actions in the list:



5 Click the **X** to close the Action List editor.

6 With the *ActionList* component still selected on the form, set its *Images* property to `ImageList1`.



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

Adding images to the image list

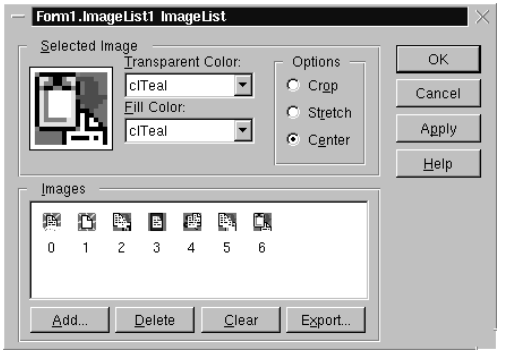
Previously, you added an ImageList object to your form. In this section, you'll add images to that list for use on the toolbar and on menus. Following are the images to use for each command:

Command	Icon image name	ImageIndex property
File	Open	Fileopen.bmp	0
File	New	Filenew.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	About	Help.bmp	7

To add the images to the image list:

1. Double-click the ImageList object on the form to display the Image List editor.
2. Click the Add button and navigate to the Buttons directory provided with the product. The default location is {install directory}/images/buttons. For example, if Kylix is installed in your /root directory, look in /root/kylix/images/buttons.
3. Double-click fileopen.bmp.
4. 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.
   - Click Add. Double-click filenew.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.

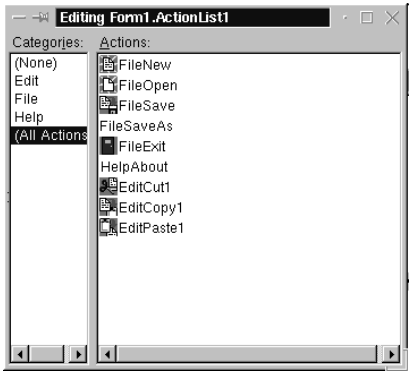


5 Click OK to close the Image List editor.

You've added 7 images to the image list and they're numbered 0-6 consistent with the ImageIndex numbers on 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.

6 To see the associated icons on the action list, double-click the ActionList object then select the All Actions category.



When you display the Action List editor now, you'll see the icons associated with the actions.

We didn't select icons for one of the commands because it will not be on the toolbar.

When you're done close the Action List editor. Now you're ready to add the menu and toolbar.

Adding a menu

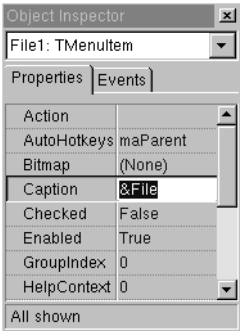
In this section, you'll add a main menu bar with three drop-down menus—File, Edit, and Help—and you'll add menu items to each one using the actions in the action list.

1 From the Standard page of the Component palette, drop a *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 bitmaps to the menu items.

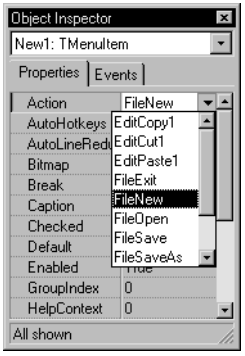
3 Double-click the main menu component to display the Menu Designer.

4 In the Object Inspector, after *Caption*, type `&File`, and press *Enter* to set the first top-level menu item.

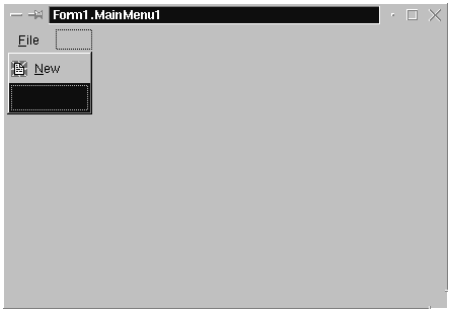


When you type *&File* and focus on the Menu Designer, the top-level File command appears ready for you to add the first menu item.

5 In the Menu Designer, select the File item you just created. You'll notice an empty item under it: select the empty item. In the Object Inspector, choose the *Action* property. The Actions from the action list are all listed there. Select `FileNew`.



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

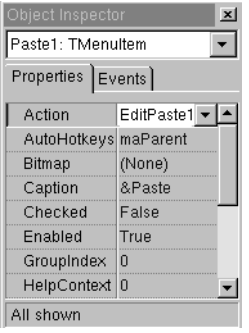


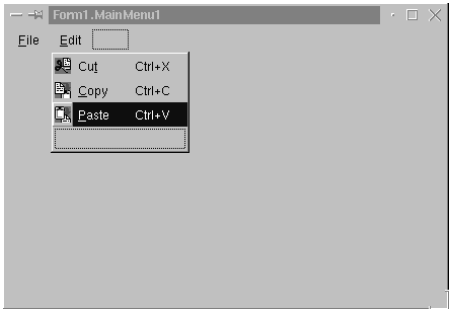
- Select the item under New and set its *Action* property to `FileOpen`.
- Select the item under Open and set its *Action* property to `FileSave`.

- Select the item under Save and set its *Action* property to `FileSaveAs`.
- Select the item under Save As, type a hyphen after its *Caption* property, and press *Enter*. This creates a separator bar on the menu.
- Select the item under the separator bar and set its *Action* property to `FileExit`.

6 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 under Cut and set its *Action* property to `EditCopy1`.
- Select the item under Copy and set its *Action* property to `EditPaste1`.





7 Next create the Help menu:

- Select the item to the right of the Edit command, set its *Caption* property to `&Help`, and press *Enter*.
- Select the item under Help and set its *Action* property to `HelpAbout`.

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

9 Choose File | Save to save changes in your project.

10 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 appears. The menus, text area, and status bar all appear on the form.

When you run your project, Kylix 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 we associated icons.

Though your program already has a great deal of functionality, there's still more to do to activate the commands. And we want to add a toolbar to provide easy access to the commands.

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

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

Clearing the text area

When you ran your program, the name *Memo1* appeared in the text area. You can remove that text using the Strings List Editor. If you don't clear the text now, the text should be removed when initializing the main form in the last step.

To clear the text area:

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

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

3 Select and delete the text you want to remove in the String List editor, and click OK.

4 Save your changes and trying running the program again.

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

Adding a toolbar

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 Common Controls page of the Component palette, double-click the *ToolBar* component to add it to the form.

A blank toolbar is added under the main menu. With the toolbar still selected, change the following properties in the Object Inspector:

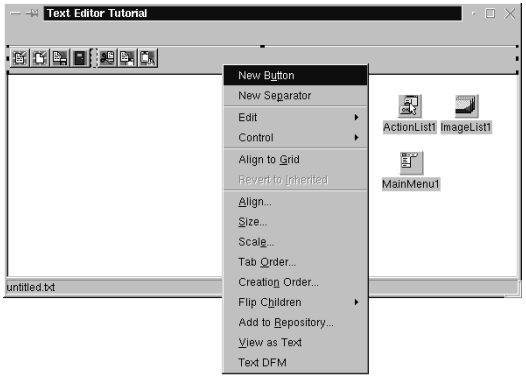
- Set the toolbar's *Indent* property to 4. (This indents the icons 4 pixels from the left of the toolbar.)
- Set its *Images* property to `ImageList1`.
- Set *ShowHint* to *True*. (**Tip:** Double-click *False* to change it to *True*.)

2 Add buttons and separators to the toolbar:

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

- Right-click and choose New Button three more times.

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



The toolbar object is added under the menus by default.

To add buttons or separators, select the toolbar, right-click, and choose New Button or New Separator. Then assign actions from the action list.

3 Assign actions from the action list to the first set of buttons.

- Select the first button and set its *Action* property to `FileExit`.
- Select the second button and set its *Action* property to `FileNew`.
- Select the third button and set its *Action* property to `FileOpen`.
- Select the fourth button and set its *Action* property to `FileSave`.

4 Assign actions to the second set of buttons.

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

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

Your text editor already has lots of functionality. You can type in the text area. Check out the toolbar. If you select text in the text area, the Cut, Copy, and Paste buttons should work.

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

Writing event handlers

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

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

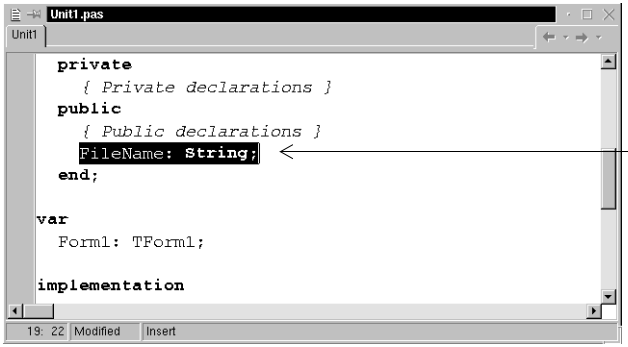
Creating an event handler for the New command

To create an event handler for the New command:

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

   ```
   FileName: String;
   ```

   Your screen should look like this:

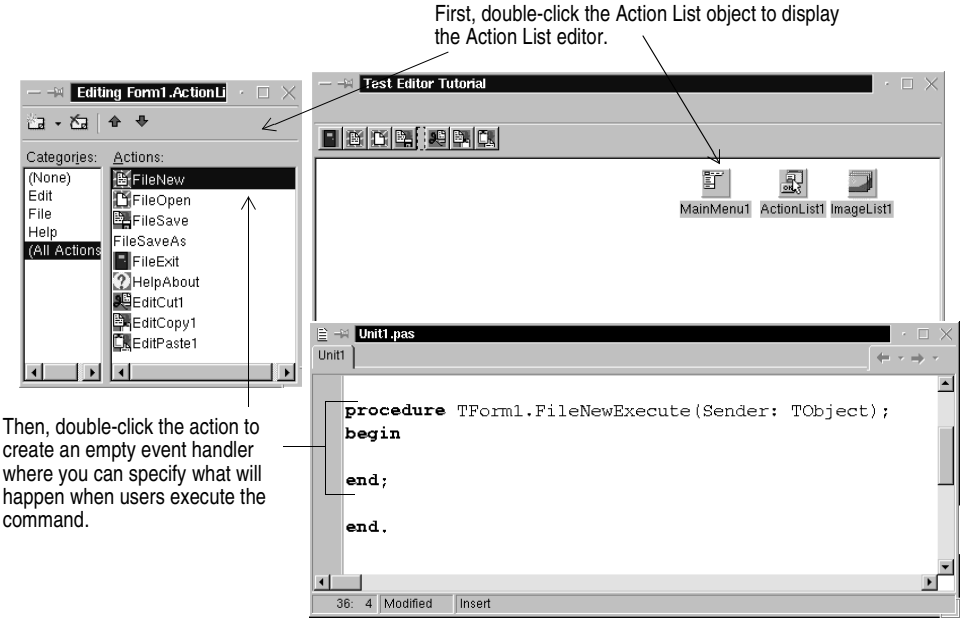


3. Press *F12* to go back to the main form.

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

4. Double-click the ActionList icon on the form to display the Action List editor.
5. In the Action List editor, select the File category and then double-click the FileNew action.

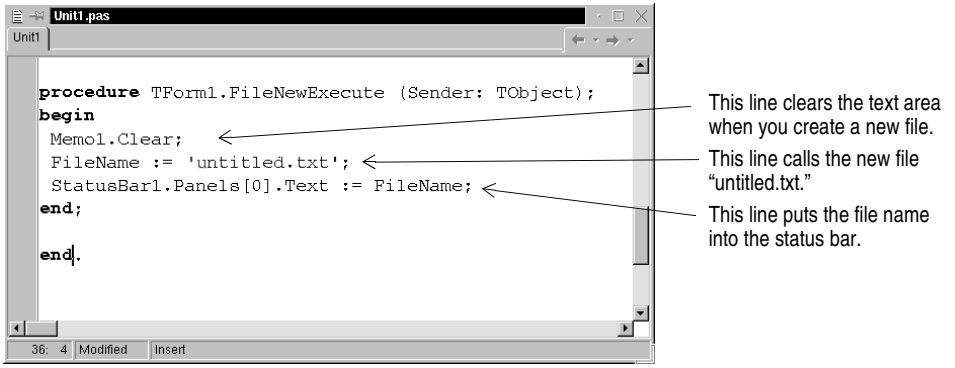
The Code Editor opens with the cursor inside the event handler.



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

```
Memo1.Clear;
FileName := 'untitled.txt';
StatusBar1.Panels[0].Text := FileName;
```

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



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

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

Creating an event handler for the Open command

When you open a file, a File Open dialog box is automatically displayed. To attach it to the Open command, drop a *TOpenDialog* object on the main editor form. Then you can write the event handler for the command.

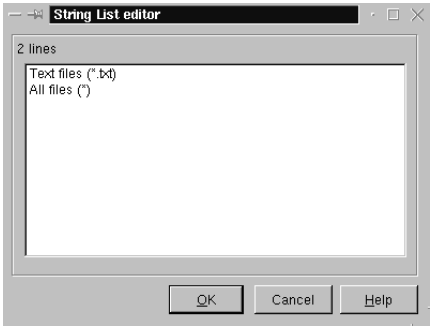
To create an Open dialog box and an event handler for the Open command:

1 Locate the main form (select View | Forms and choose Form1 to quickly find it).

2 On the Dialogs page on the Component palette, double-click an *OpenDialog* component to add it to the form. This is a nonvisual component, so it doesn't matter where you place it. Kylix names it *OpenDialog1* by default. (When *OpenDialog1*'s *Execute* method is called, it invokes a standard dialog box for opening files.)

3 In the Object Inspector, set the following properties of *OpenDialog1*:

- Set *DefaultExt* to `txt`.
- Double-click the text area next to *Filter* to display the String List editor. In the first line, type `Text files (*.txt)`. "Text files" is the filter name and "(*.txt)" is the filter. On the second line, type `All files (*)`. Click OK.



Use the String List editor to define filters for the *OpenDialog* and *SaveDialog* components.

- Set *Title* to `Open File`.

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

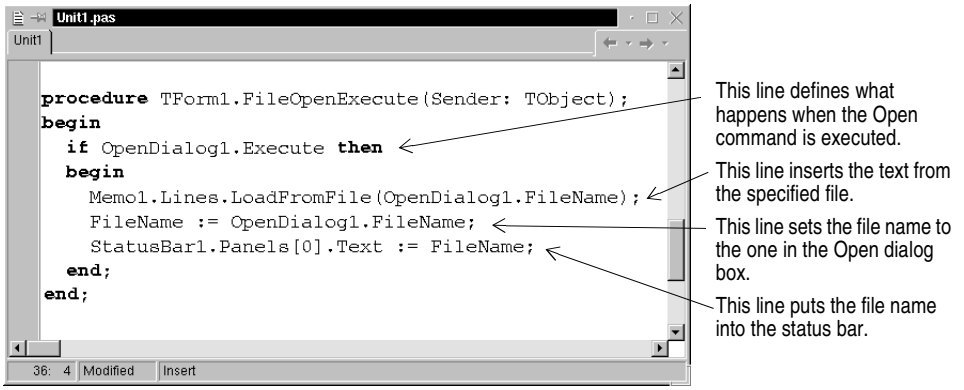
5 In the Action List editor, double-click the FileOpen action.

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

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

```
if OpenDialog1.Execute then
begin
 Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
 FileName := OpenDialog1.FileName;
 StatusBar1.Panels[0].Text := FileName;
end;
```

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:

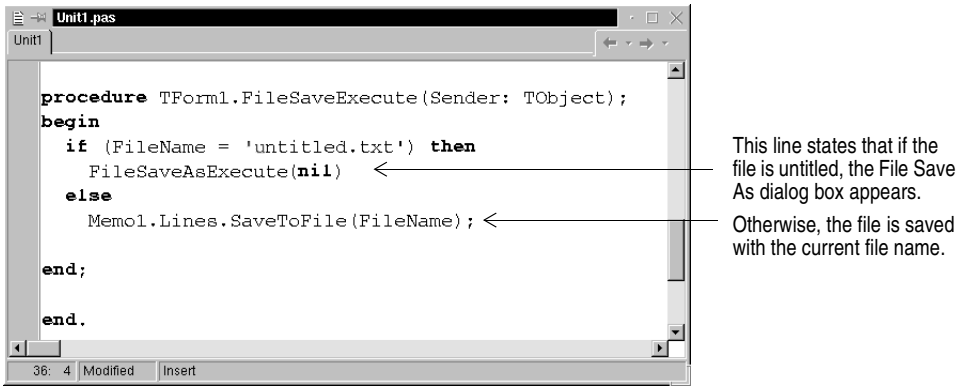
1. The Action List editor should still be displayed. If it's not, double-click the ActionList icon on the form.
2. On the Action List editor, double-click the FileSave action.

   The Code Editor opens with the cursor inside the event handler.
3. Right where the cursor is positioned in the Code Editor (between **begin** and **end**), type the following lines:

```
if (FileName = 'untitled.txt') then
 FileSaveAsExecute(nil)
else
 Memo1.Lines.SaveToFile(FileName);
```

   This code tells the text editor to display the SaveAs dialog box if the file isn't named yet so the user can assign a name to it. Otherwise, save the file using its current name. The SaveAs dialog box is defined in the event handler for the Save As command on page 4-21. FileSaveAsExecute 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

To create an event handler for the Save As command:

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

2 In the Object Inspector, set the following properties of *SaveDialog1*:

- Set *DefaultExt* to txt.
- Double-click the text area next to *Filter* to display the String List editor. In the Editor, specify filters for file types as in the Open dialog box. In the first line, type `Text files (*.txt)`; in the second line, type `All files (*)`. Click OK.
- Set *Title* to `Save As`.

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

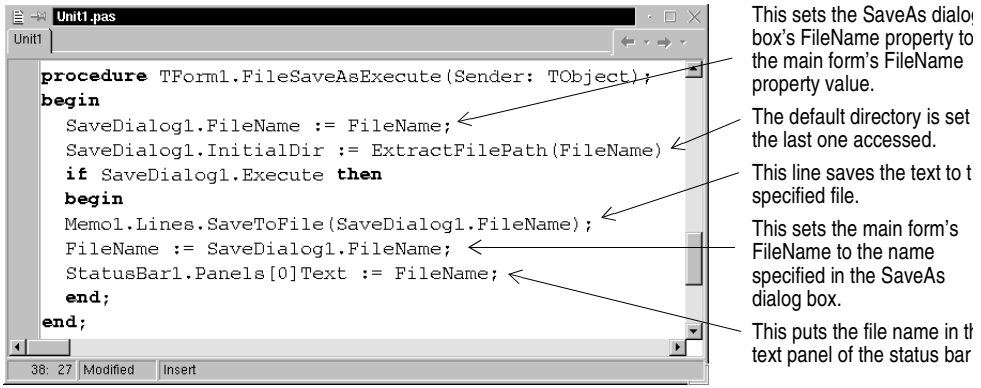
3 In the Action List editor, double-click the FileSaveAs action.

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

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

```
  SaveDialog1.FileName := FileName;
  SaveDialog1.InitialDir := ExtractFilePath(Filename);
if SaveDialog1.Execute then
begin
  Memo1.Lines.SaveToFile(SaveDialog1.FileName);
  FileName := SaveDialog1.FileName;
  StatusBar1.Panels[0].Text := FileName;
end;
```

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



That's it for the File | SaveAs command.

Creating an event handler for the Exit command

To create an event handler for the Exit command:

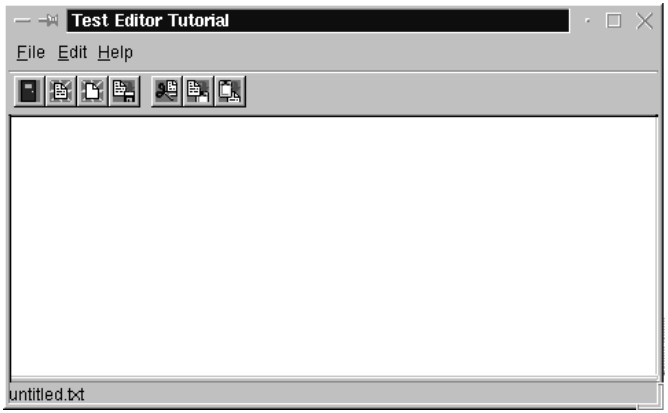
1. The Action List editor should still be displayed. If it's not, double-click the ActionList icon on the form.
2. On the Action List editor, double-click the FileExit action.

   The Code Editor opens with the cursor inside the event handler.
3. Right where the cursor is positioned in the Code Editor, type the following line:

   ```
   Close;
   ```

   This calls the close method of the main form. That's all you need to do for the File | Exit command.
4. Choose File | Save All to save your project.

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



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

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

5 To return to design mode, close the text editor application by clicking the **X**.

If you receive any error messages, click them to locate the error. Make sure you've followed the steps as described in the tutorial.

Creating an About box

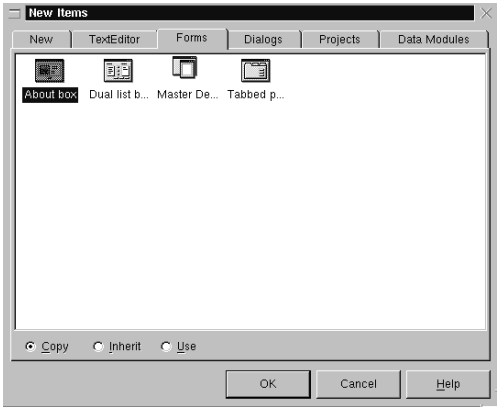
Many applications include an About box which displays information on the product such as the name, version, logos, and may include other legal information including copyright information.

We've already set up a Help About command on the action list.

To create an About box:

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

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



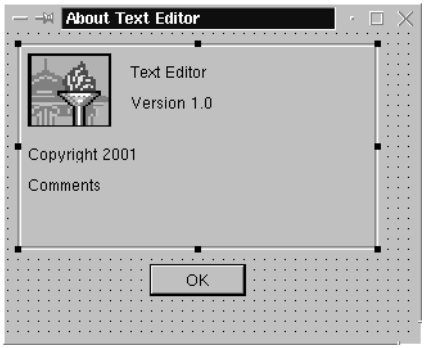
A new form is created that simplifies creation of an About box.

3 Select the following *TLabel* items on the About box and in the Object Inspector, change their *Caption* properties:

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

4 Select the form itself and in the Object Inspector, change its *Caption* property to `About Text Editor`.

Tip The easiest way to select the form is to click the grid portion.



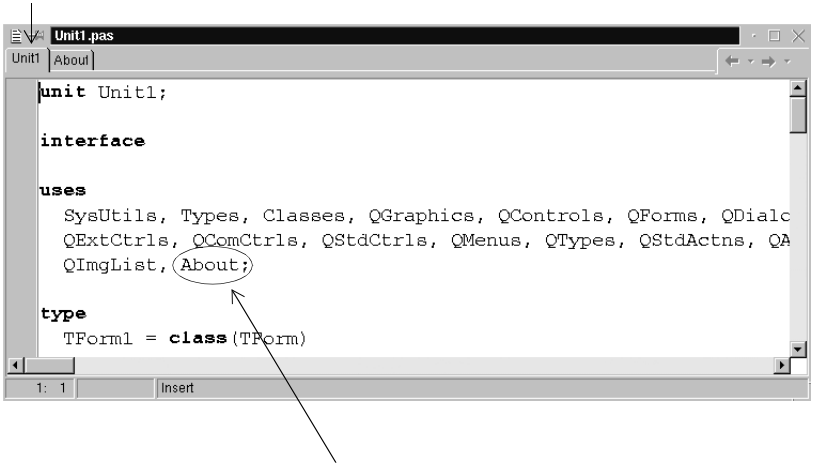
The Object Repository contains a standard About box that you can modify as you like to describe your application.

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

6 In the Kylix Code Editor, you should have two files displayed: Unit1 and About. Click the Unit1 tab to display Unit1.pas.

7 Add the new About unit to the **uses** section of Unit1, the main form: add the word About to the list of included units in the **uses** clause.

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



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

8 On the action list, double-click the HelpAbout action to create an event handler.

9 Right where the cursor is positioned in the Code Editor, type the following line:

```
AboutBox.ShowModal;
```

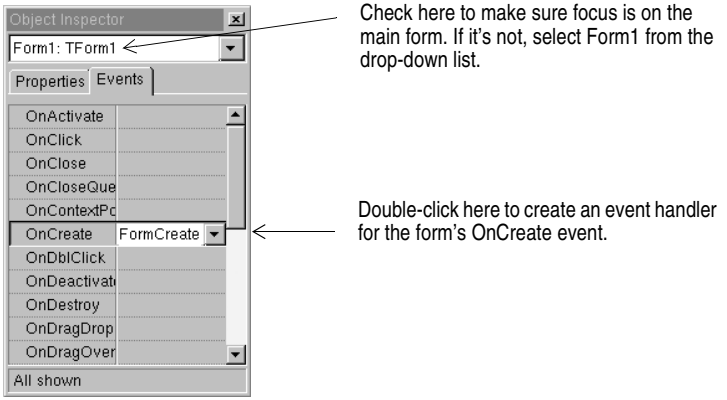
This code opens the About box when the user clicks Help | About. ShowModal opens the form in a modal state, a runtime state when the user can't do anything until the form is closed.

Completing your application

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

1 Locate the main form (press *F12* to quickly find it).

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



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

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

```
FileName := 'untitled.txt';
StatusBar1.Panels[0].Text := FileName;
Memo1.Clear;
```

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

5 Press *F9* to run the application.

You can test the text editor now to make sure it works. If errors occur, click the error message and you'll go right to the place in the code where the error occurred.

Congratulations! You're done.

Chapter

5

Customizing the desktop

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

Organizing your work area

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

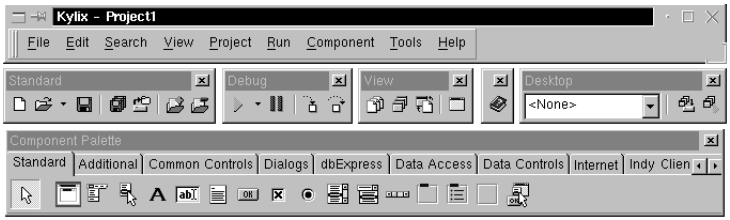
Arranging menus and toolbars

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

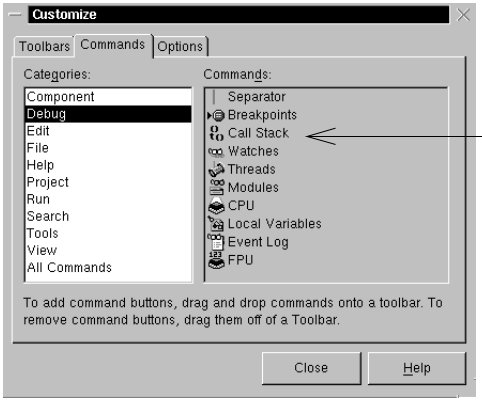
Main window organized differently.

You can 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.



Drag the grabber of an individual toolbar to move it.

You can add or delete tools from the toolbars by choosing View | Toolbars | Customize.



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

Docking tool windows

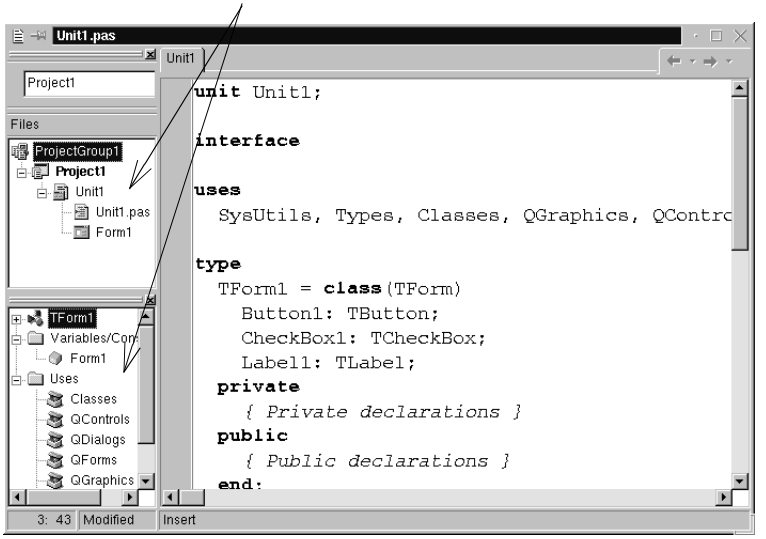
You can open and close individual tool windows and arrange them on the desktop as you wish. Many windows can also be *docked* to one another for easy management. Docking—which means either attaching windows to each other so that they move together or combining several windows into a tabbed "notebook"—helps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

From the View menu, you can bring up any tool window and then dock it directly to the Code Editor for use while coding and debugging. For example, when you first open Kylix in its default configuration, the Code Explorer is docked to the left of the

Code Editor. You can add the Project Manager to the first two to create three docked windows.

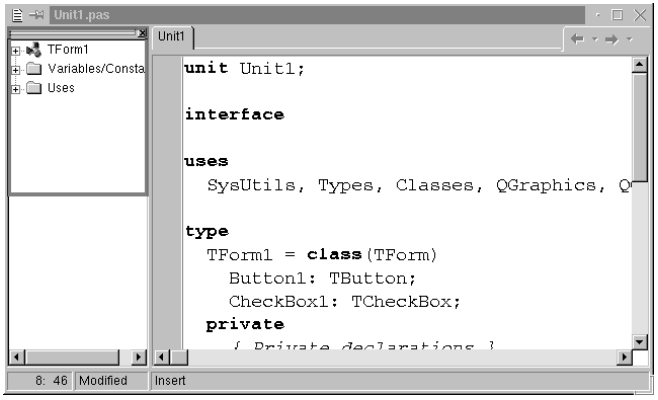
Here the Project Manager and Code Explorer are docked to the Code Editor.

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

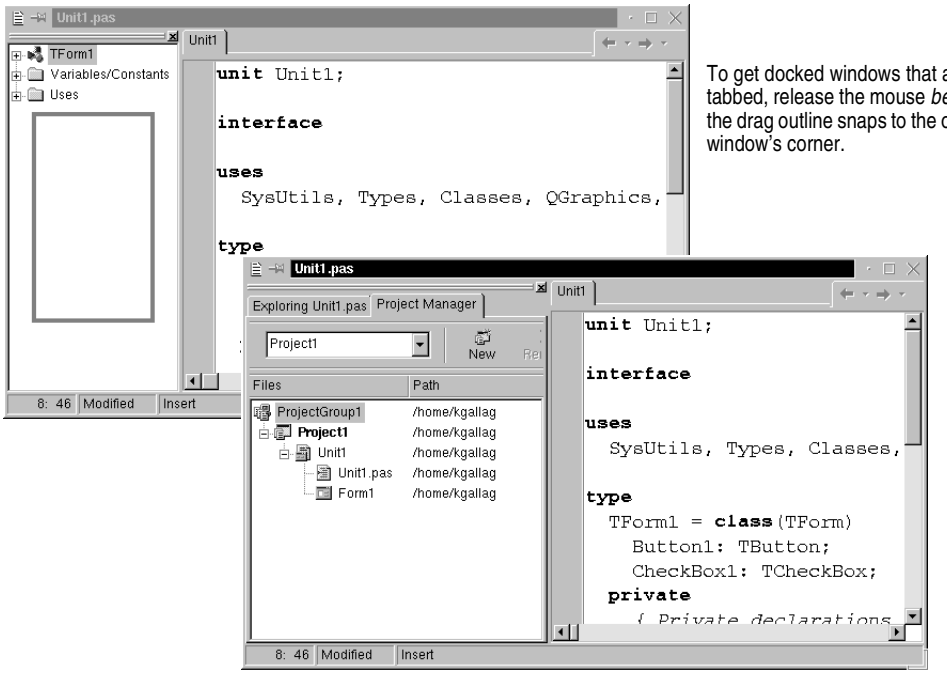


To dock a window, click its title bar and drag it over the other window. When the drag outline narrows into a rectangle and it snaps into a corner, release the mouse. The two windows dock together.

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 a tabbed window.



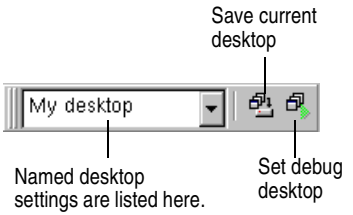
To get docked windows that are tabbed, release the mouse *before* the drag outline snaps to the other window's corner.

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

For more information...
See "docking" in the 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, and placing them where you want on the screen. 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 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:

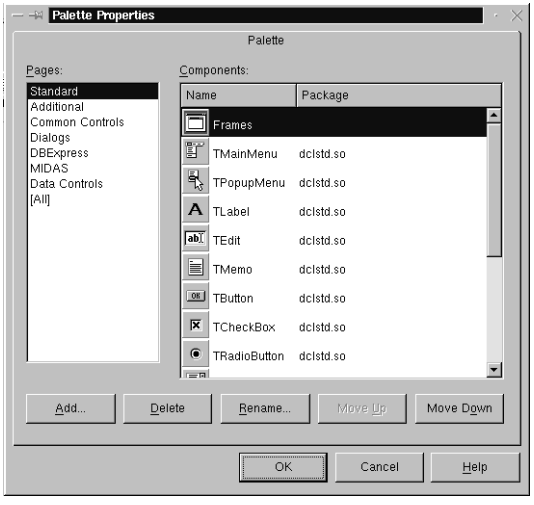
- Hiding or rearranging components.
- Adding, removing, rearranging, or renaming pages.
- Creating component templates and adding them to the palette.
- Installing new components.

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...

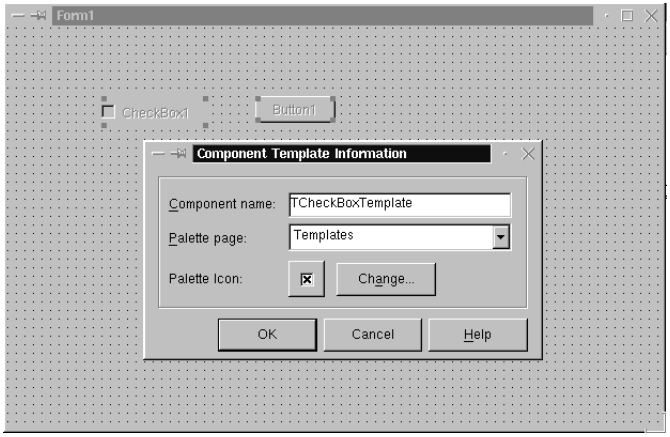
Click the Help button in the Palette Properties dialog box.

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...

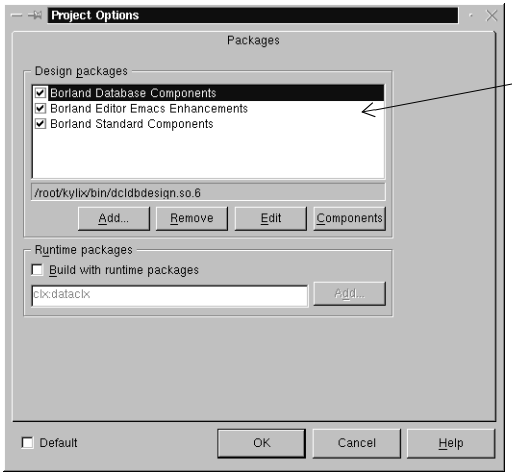
See “templates, component” in the Help index.

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.

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 appear in this list.

Click Components to see what components the package contains.

For more information...

See "installing components" and "packages" 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.

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 defprof.kof. To restore Kylix's original default settings, delete or rename the defproj.kof file, which is located in {home directory}/.borland.

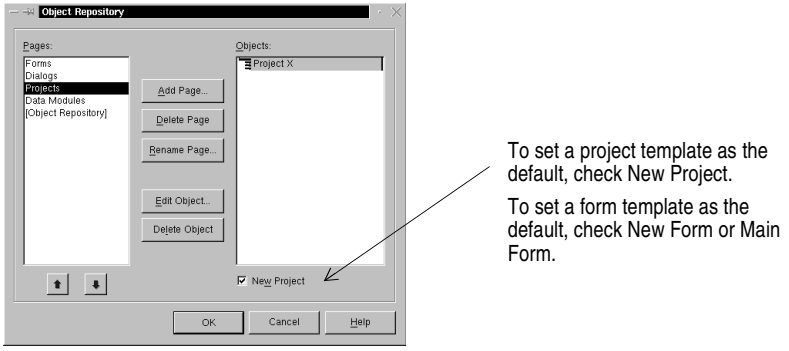
For more information...

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

Specifying project and form templates as the default

When you choose File | New Application, a new project opens in the IDE. Kylix creates a standard new application with an empty form, unless you specify a project *template* as your *default* project. Kylix does not come with predesigned project templates, but 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 items to your projects: The Object Repository" on page 2-5).

To specify your 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, and check New Project.



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

You can always override your default project or forms by choosing File | New 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 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 Component palette. 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 Help index, or click on the Help button on any page in the Environment Options dialog box.

Customizing the Form Designer

The Preferences page of the 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.

For more information...

In the Environment Options dialog box, click the Help button on the Preferences page.

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 "Getting help with your code" on page 2-9.

For more information...

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

Customizing the Code Explorer

When you start Kylix, the Code Explorer (described in "Viewing code with the Code Explorer" on page 2-11) opens automatically (unless the Code Explorer is not in the edition of Kylix you purchased). If you don't want Code Explorer to open automatically, choose Tools | Environment Options, click the Explorer tab, and uncheck Automatically show Explorer. Use the Explorer page to set other options for the Code Explorer.

For more information...

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

Index

P

R

S

T

U

W

X