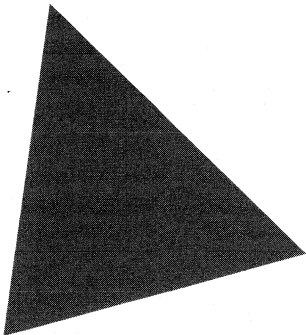




Quick Start



Borland®
Kylix™ 2
Delphi™ for Linux®

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

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

COPYRIGHT © 2001 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. All other marks are the property of their respective owners.

HDE7020WW21000 2E2R1001
0001020304-9 8 7 6 5 4 3 2 1
D3

Contents

Chapter 1		
Introduction	1-1	
What is Kylix?	1-1	
Requirements	1-2	
Installation	1-2	
The Kylix installer	1-2	
Root versus non-root install	1-2	
Installing Kylix	1-3	
Registering Kylix	1-4	
Starting Kylix	1-4	
Finding information	1-4	
Online Help	1-5	
Printed documentation	1-6	
Developer support services and Web site	1-6	
Uninstalling Kylix	1-7	
Typographic conventions	1-7	
Chapter 2		
A tour of the desktop	2-1	
The IDE	2-1	
The menu and toolbars	2-2	
The Component palette, Form Designer, and Object Inspector	2-3	
The Object Repository	2-4	
The Code editor	2-5	
Code Insight tools	2-6	
Class completion	2-7	
Form code	2-8	
The Code Explorer	2-8	
The Project Manager	2-9	
The Project Browser	2-10	
To-do lists	2-10	
Chapter 3		
Programming with Kylix	3-1	
Creating a project	3-1	
Designing the user interface	3-1	
Placing components on a form	3-2	
Setting component properties	3-2	
Writing code	3-4	
Writing event handlers	3-4	
Using CLX classes	3-4	
Adding data modules	3-5	
Compiling and debugging projects	3-6	
Deploying applications	3-7	
Internationalizing applications	3-7	
Types of projects	3-8	
Database applications	3-8	
Web server applications	3-9	
Web services	3-9	
Shared objects	3-10	
Custom components	3-10	
Chapter 4		
Creating a text editor—a tutorial	4-1	
Starting a new application	4-1	
Setting property values	4-2	
Adding components to the form	4-3	
Adding support for a menu and a toolbar	4-5	
Adding actions to the action list	4-7	
Adding standard actions to the action list	4-9	
Adding images to the image list	4-10	
Adding a menu	4-12	
Clearing the text area	4-15	
Adding a toolbar	4-15	
Writing event handlers	4-16	
Creating an event handler for the New command	4-17	
Creating an event handler for the Open command	4-19	
Creating an event handler for the Save command	4-20	
Creating an event handler for the Save As command	4-21	
Creating an event handler for the Exit command	4-22	
Creating an About box	4-23	
Completing your application	4-25	
Chapter 5		
Creating a database application— a tutorial	5-1	
Overview of database architecture	5-1	
Creating a new project	5-2	
Setting up data access components	5-2	
Setting up the database connection	5-3	
Setting up the unidirectional dataset	5-4	
Setting up the provider, client dataset, and data source	5-5	

Designing the user interface	5-5
Creating the grid and navigation bar	5-6
Adding support for a menu	5-7
Adding a menu	5-9
Adding a button	5-11
Displaying a title and an image	5-11
Writing an event handler	5-12
Writing the Update Now! command event handler	5-12
Writing the Exit command event handler	5-13
Writing the FormClose event handler	5-13

Chapter 6

Customizing the desktop	6-1
Organizing your work area	6-1
Arranging menus and toolbars	6-1
Docking tool windows	6-2
Saving desktop layouts	6-4

Customizing the Component palette	6-5
Arranging the Component palette.	6-5
Creating component templates	6-6
Installing component packages	6-7
Setting project options	6-8
Setting default project options	6-8
Specifying project and form templates as the default.	6-9
Adding templates to the Object Repository	6-9
Setting tool preferences.	6-10
Customizing the Form Designer.	6-10
Customizing the Code editor	6-11
Customizing the Code Explorer	6-11
Printing Help topics	6-12
Printing to a file	6-12
Printing directly to a printer	6-12

Index	I-1
--------------	------------

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.

This chapter describes how to install, register, and start Kylix, as well as how to find information. 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 text editor tutorial. Chapter 5, “Creating a database application—a tutorial,” takes you step by step through a database application tutorial. 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.

Kylix has three editions: Enterprise, Professional, and Open Edition. The particular features and components available to you depend on which edition of Kylix you have. To see what tools are available to you, refer to the feature list on <http://www.borland.com/kylix>.

Requirements

The following software must be installed to run Kylix:

- Linux kernel version 2.2 or higher.
- libgtk.so version 1.2 or higher (required for graphical installation only).
- libjpeg version 6.2 (libjpeg.so.62) or higher.
- an X11R6-compatible terminal server, such as XFree86.

Installation

Warning Do not use a package manager, such as RPM, kpackage, or gnorpm, to install or uninstall Kylix. Always use the install and uninstall programs provided by Borland.

Kylix must be installed on a file system that supports symbolic links, such as Linux's ext2. Do not install Kylix on FAT, FAT32, SMB (including Samba), or Novell file systems.

The Kylix installer

Kylix will install equally well on RPM and non-RPM systems. Although the Kylix installer will update the RPM package database to reflect the installed software, the installer can also work independently of RPM.

If the installation system has RPM, and Kylix is installed by the root user, several Kylix packages are added to the installed packages list. Uninstalling Kylix removes these packages. All package names begin with "kylix_." If your system does not have RPM, or if you install Kylix as a user other than root, the installer simply installs Kylix without adding entries to the RPM database.

Root versus non-root install

Kylix can be installed by any user who can provide the necessary disk space. If you do not have the password for the root account on your machine, you will have to install Kylix using a non-root account. If you have the root password, you need to decide whether to install Kylix as root or as another user.

The advantages of installing as a root user

- If the installation program has full access to the system, it can automatically resolve some installation problems, such as missing library links.
- All files are placed in standard, central locations, so that any user can run Kylix. Users can share CLX objects, using the Kylix Object Repository.
- If your system uses the RPM Package Manager, the installer adds a set of Kylix entries to the RPM database. This allows dependencies between Kylix and other software to be maintained automatically.

The advantages of installing as a standard user

- You own all Kylix files, and do not have to take any additional steps to work with the demo projects, modify the Object Repository, etc.
- If you omit some install options, and decide to add them later, the installer will automatically configure your `.borland/delphi60rc` file to make these options available. With a root install, the only way to update this file is to delete it, so that Kylix will regenerate it. Unfortunately, deleting this file causes all your IDE option changes to be lost.

Installing Kylix

To install Kylix, follow these steps:

- 1 Login as superuser (this step is optional if you do not need Kylix to be accessible by all users).
 - If you are using X Windows, open a terminal window.
- 2 Insert the CD-ROM in the drive, and mount the CD-ROM file system (some systems only allow the root user to mount the CD-ROM).
- 3 Change your working directory to the CD-ROM mount point.
- 4 Run the Kylix installer, `setup.sh`. If your CD-ROM file system is not configured to support executables, you will need to use the shell command "sh setup.sh."

The default installation location for Kylix is always beneath the home directory of the installing user. But Kylix should never be installed in `/root` (which is the home directory of the root user, and as such will be the default location when installing as root). That directory is not normally accessible to other users, and running Kylix as root is not recommended. When installing Kylix as root, always specify another installation location, such as `/usr/local/kylix2`.

- 5 Follow the installer prompts.

For example, on a system configured to mount the CD-ROM device at `/mnt/cdrom`, the following shell commands mount the CD-ROM file system and run the installer:

```
mount /mnt/cdrom
cd /mnt/cdrom
./setup.sh
```

Your system may use a mount point other than `/mnt/cdrom`. Substitute that mount point in the example above.

Note If run under X Windows and with the correct version of `libgtk.so`, the Kylix installer opens an X Windows dialog box. Otherwise, the Kylix installer uses a series of text console prompts. In either case, you will be prompted to specify an Install Path and a Link Path.

Note Even if you install Kylix as root, always run Kylix as an ordinary user. Installing Kylix as an ordinary user is acceptable, but only the installing user will be able to run Kylix in a non-root install.

For more information...

For more completed information on installing Kylix, see <http://www.borland.com/techpubs/kylix>.

Registering Kylix

This product must be activated with a serial number and authorization key. After activating Kylix, you must also register the product.

The Registration dialog box, which appears on first use after activation, offers three registration methods: online (direct registration through a secure connection), by telephone, or through a Web form.

Each of the registration options is driven by wizards, with Help provided where necessary.

For more information...

For more complete information on registering Kylix, see <http://www.borland.com/techpubs/kylix>.

Starting Kylix

You can start Kylix in the following ways:

- From a shell window, simply type `startkylix`. This will work if the link path you chose during installation was a directory that is in your path.
- Enter `{install path}/startkylix`. For example, if your install path is `/usr/local/kylix2`, enter: `/usr/local/kylix2/bin/startkylix`.
- Launch Kylix from the Application Starter menu in either Gnome or KDE.

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

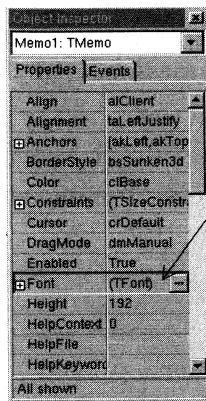
Online Help

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

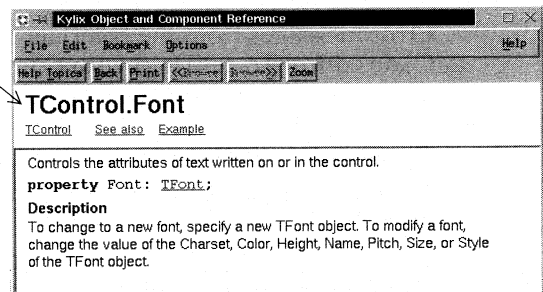
To view the table of contents, choose Help | Kylix Help and click the Contents tab. To find information on CLX objects or any other topic, click the Index or Find tab and enter your request.

F1 Help

The 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*.

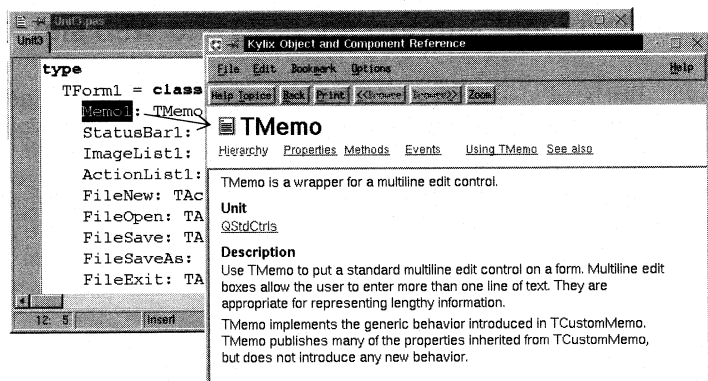


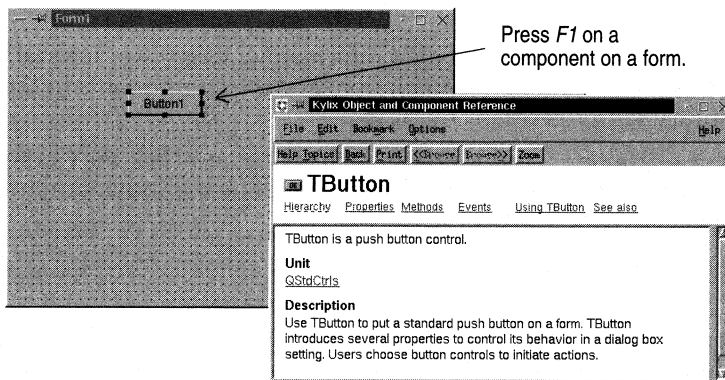
Press *F1* on a property or event name in the Object Inspector to display CLX Help.



In the Code editor, press *F1* on a language keyword or CLX element.

For any xlib or libc function, *F1* brings up the man page for that function.





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*.

For more information...

To configure your Help system to print Kylix Help topics, see “Printing Help topics” on page 6-12.

Printed documentation

This *Quick Start* is an introduction to Kylix. To order additional printed documentation, such as the *Developer’s Guide*, refer to <http://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.

Uninstalling Kylix

To uninstall Kylix run the uninstall program in the Kylix installation directory as the same user who installed Kylix. For instance, if Kylix was installed in `/usr/local` by the root user, execute `/usr/local/kylix2/uninstall` as the root user. Uninstalling removes files that were copied to your hard drive during installation, but does not restore previously existing configuration files that were modified by the installer.

Typographic conventions

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

Table 1.1 Typographic conventions

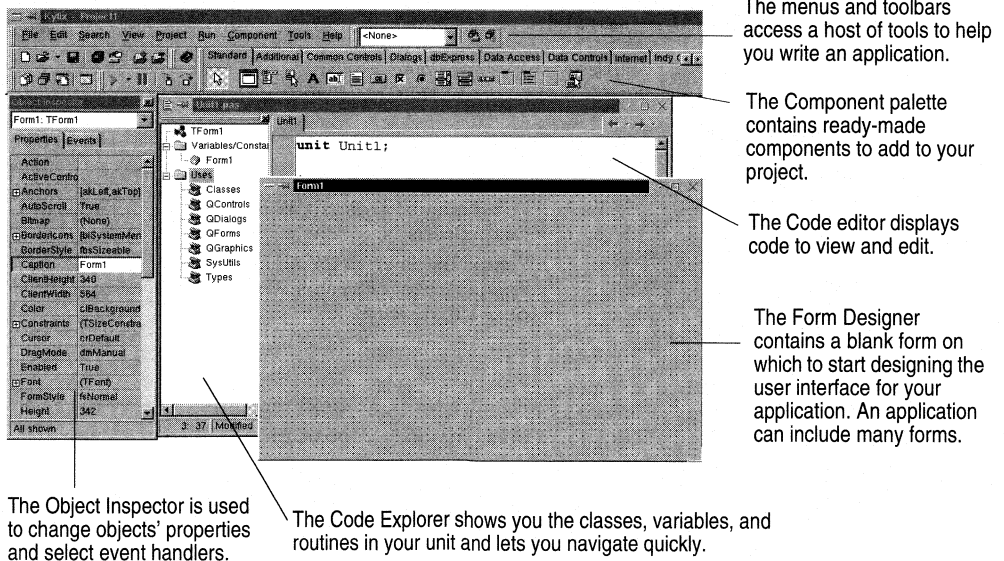
Typeface	Meaning
Monospace type	Monospaced type represents text as it appears on screen or in code. It also represents anything you must type.
Boldface	Boldfaced words in text or code listings represent reserved words or compiler options.
<i>Italics</i>	Italicized words in text represent Kylix identifiers, such as variable or type names. Italics are also used to emphasize certain words, such as new terms.
<i>Keycaps</i>	This typeface indicates a key on your keyboard. For example, "Press <i>Esc</i> to exit a menu."

A tour of the 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 development environment (IDE).

The IDE

When you first start Kylix, you'll see some of the major tools in the IDE. In Kylix, the IDE includes the menus, toolbars, Component palette, Object Inspector, 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 design graphical interfaces, browse through class libraries, write code, compile, test, debug, and manage projects without leaving the IDE.

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

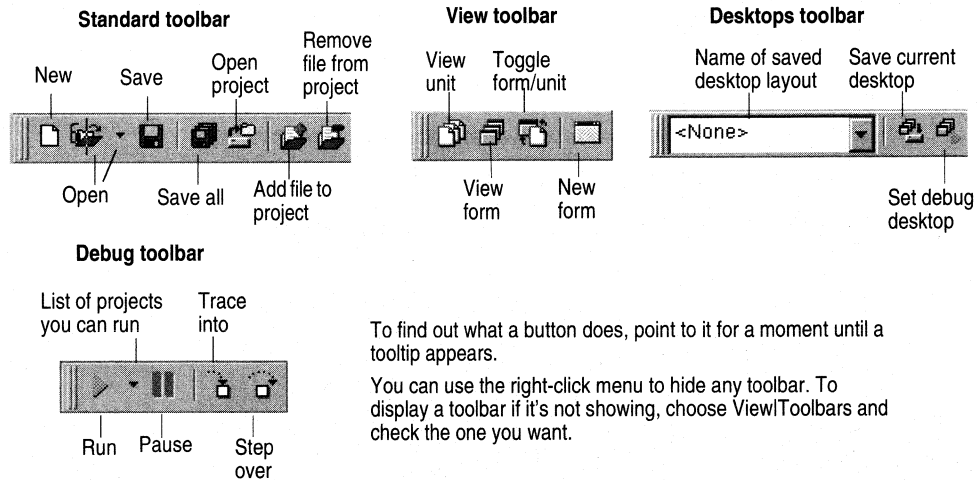
The menu and toolbars

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

For more information...

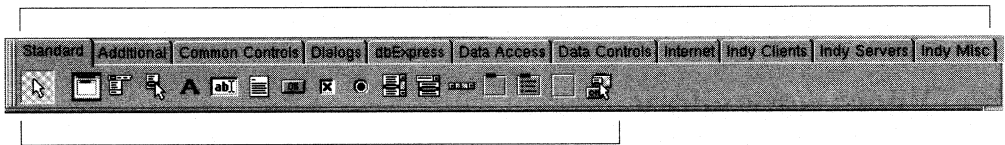
If you need help on any menu option, point to it and press *F1*.

The Component palette, Form Designer, and Object Inspector

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

The *Component palette* includes tabbed pages with groups of icons representing visual and nonvisual CLX components you use to design your 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.

Component palette pages, grouped by function

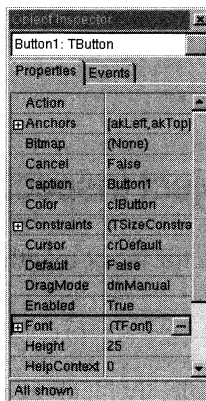


Components

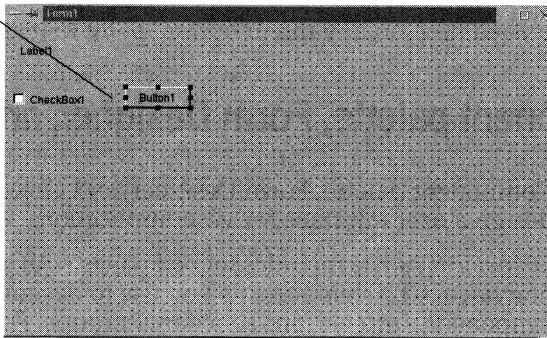
Each component has specific attributes—properties, events, and methods—that enable you to control your application.

After you place components on the form, or *Form Designer*, you can arrange components the way they should look on your user interface. For the components you place on the form, use the *Object Inspector* to set design-time properties, create event handlers, and filter visible properties and events, making the connection

between your application's visual appearance and the code that makes your application run. See "Placing components on a form" on page 3-2.



After you place components on a form, the Object Inspector dynamically changes the set of properties it displays, based on the component selected.



For more information...

See "Designing the user interface" on page 3-1, or "Component palette" and "Object Inspector" in the Help index.

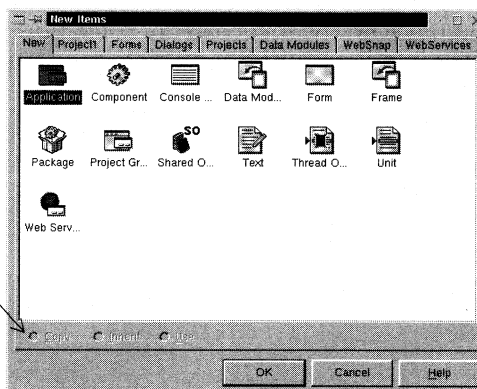
The Object Repository

The Object Repository—also known as the New Items dialog box—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. 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:

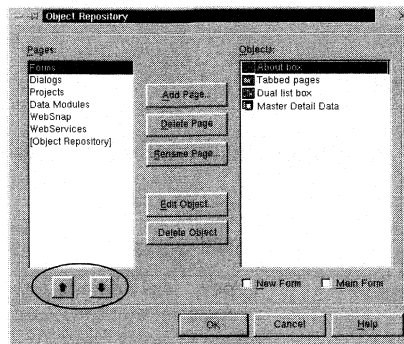
Copy (the default) creates a copy of the item in your project. *Inherit* means changes to the object in the Repository are inherited by the one in your project. *Use* means changes to the object in your project are inherited by the object in the Repository.



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

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

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



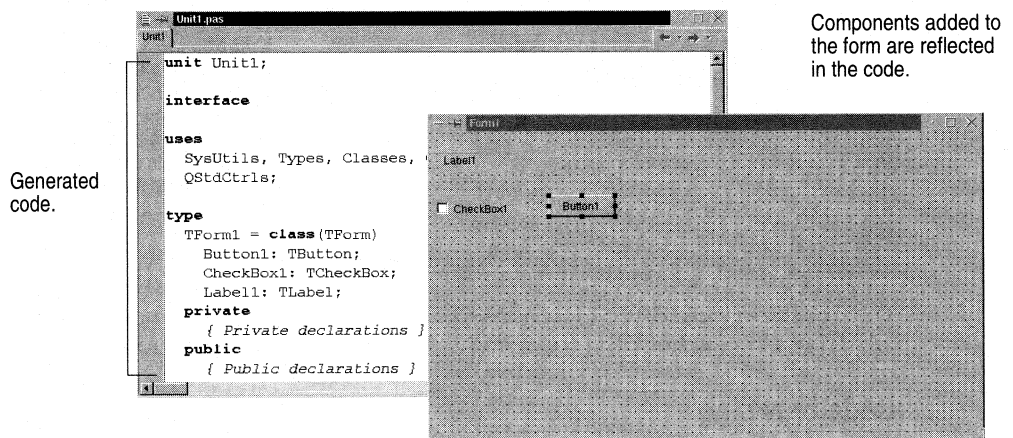
To add project and form templates to the Object Repository, see “Adding templates to the Object Repository” on page 6-9.

For more information...

See “Object Repository” in the Help index. The objects available to you depends on which edition of Kylix you purchased.

The Code editor

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.



The Code editor provides various aids to help you view and write code, including the Code Insight tools and class completion.

For more information...

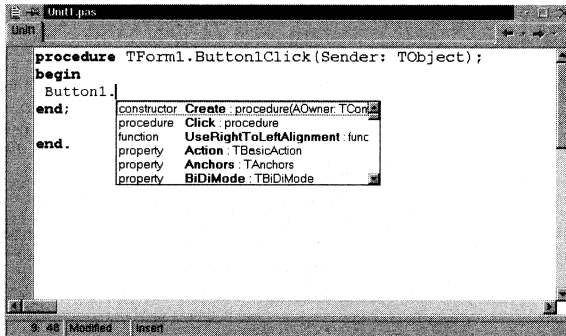
See “Code editor” in the Help index. To customize your code editing environment, see “Customizing the Code editor” on page 6-11.

Code Insight tools

The Code Insight tools displays the following context-sensitive pop-up windows.

Table 2.1 Code Insight tools

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



With code completion, when you 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.

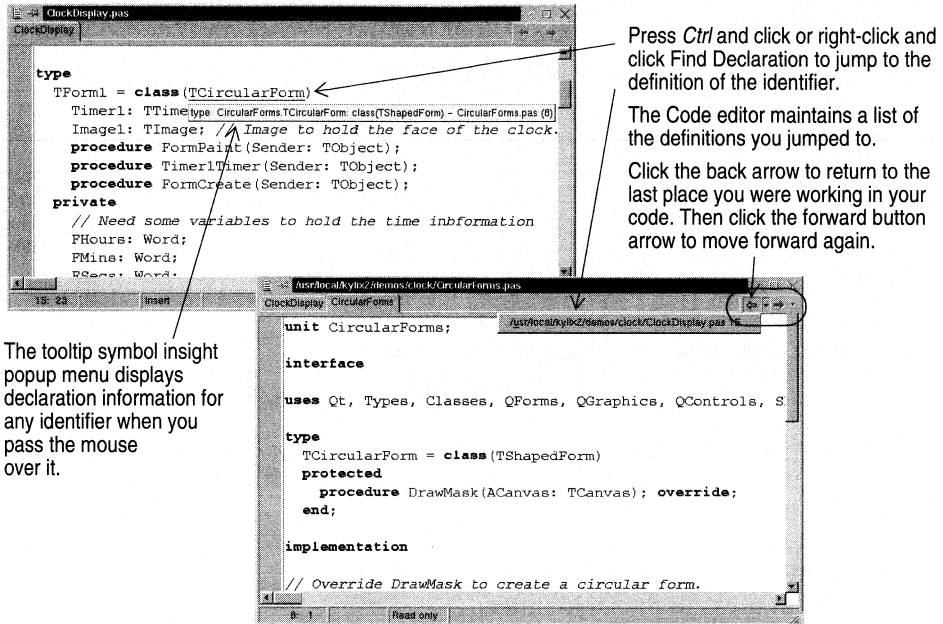
To turn these tools on or off, choose **Tools | Editor Options** and click the **Code Insight** tab. Check or uncheck the tools in the **Automatic features** section.

Code browsing

While passing the mouse over the name of any class, variable, property, method, or other identifier, the pop-up menu called tooltip symbol insight displays where the

identifier is declared. Press *Ctrl* and the cursor turns into a hand, the identifier turns blue and is underlined, and you can click to jump to the definition of the identifier.

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



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

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.

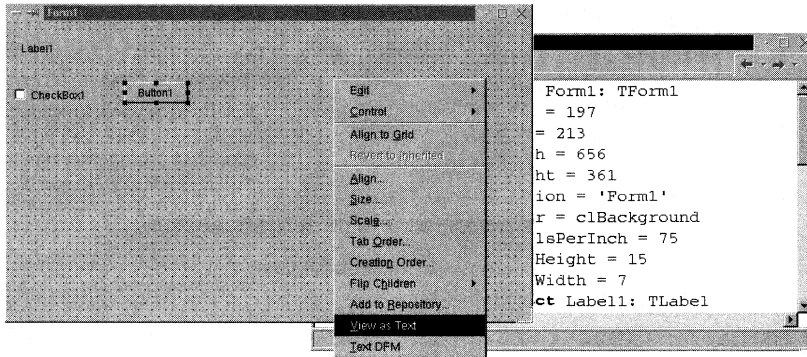
To configure class completion, choose Tools | Environment Options and click the Explorer tab.

For more information...

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

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.

The Code Explorer

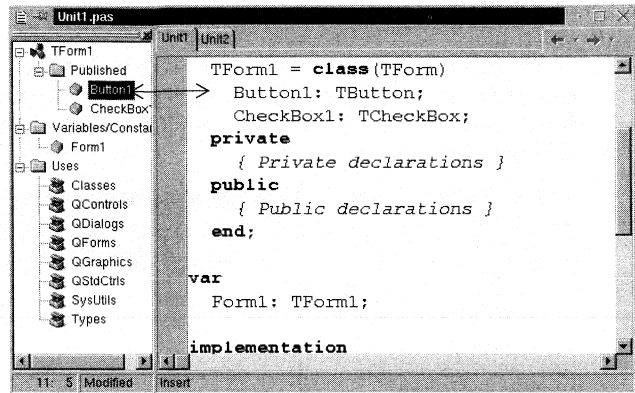
When you open Kylix, the Code Explorer is docked to the left of the Code editor window, depending on whether the Code Explorer is available in the edition of Kylix you have. The Code Explorer displays the table of contents as a tree diagram for the source code open in the Code editor, listing the types,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

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

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

Press Ctrl+Shift+E to move the cursor back and forth between the last place you were in the Code Explorer and the Code editor.

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



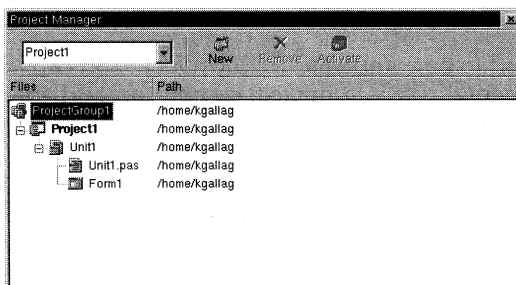
To configure how the Code Explorer displays its contents, choose Tools | Environment Options and click the Explorer tab. See "Customizing the Code Explorer" on page 6-11.

For more information...

See "Code Explorer" in the Help index.

The Project Manager

When you first start Kylix, it automatically opens a new project, as shown on page 2-1. 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 6-8.

For more information...

See "Project Manager" in the Help index.

The Project Browser

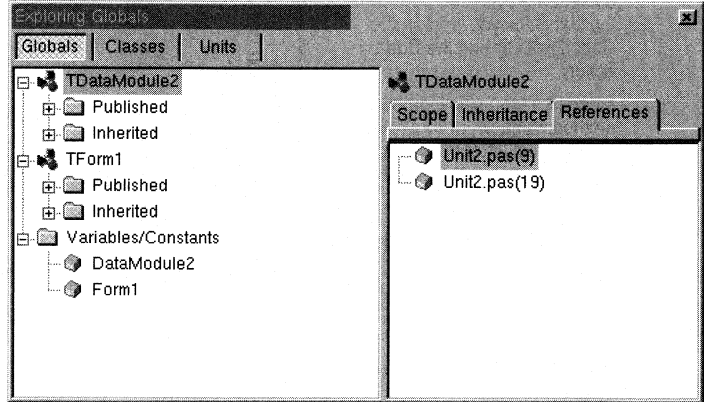
The Project Browser examines a project in detail. The Browser displays classes, units, and global symbols (types, properties, methods, variables, and routines) your project declares or uses in a tree diagram. Choose View | Browser to display the Project Browser.

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

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

Classes displays classes in a hierarchical diagram.

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



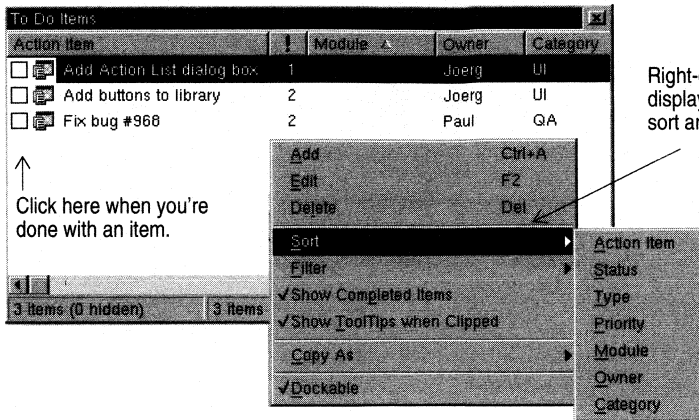
By default, the Project Browser displays the symbols from units in the current project only. You can change the scope to display all symbols available in Kylix. Choose Tools | Environment Options, and on the Explorer page, check All symbols (CLX included).

For more information...

See "Project Browser" in the Help index.

To-do lists

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



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

For more information...

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

Programming with Kylix

This chapter provides an overview of software development with Kylix, including creating a project, designing the user interface, writing code, and compiling, debugging, deploying, and internationalizing programs. The last section includes the different types of projects you can develop.

Creating a project

A project is a collection of files that are either created at design time or generated when you compile the project source code. When you first start Kylix, a new project opens. 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 “The Object Repository” on page 2-4.

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

For more information...

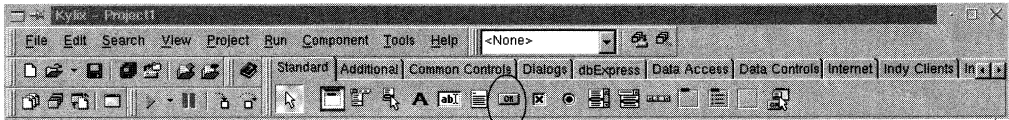
See “projects” in the Help index.

Designing the user interface

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

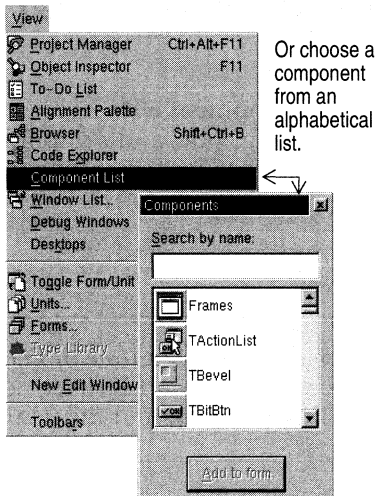
Placing components on a form

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.

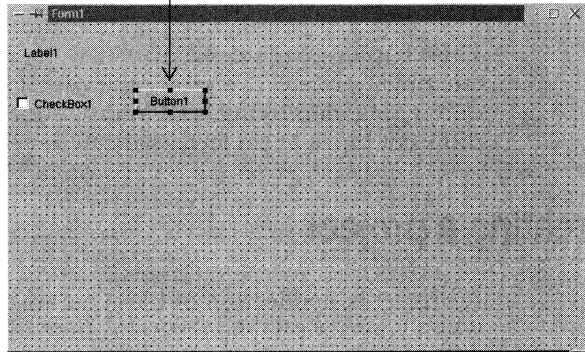


Click a component on the Component palette.

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



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



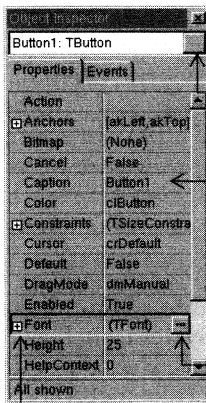
For more information...

See "Component palette" in the Help index.

Setting component properties

After you place components on a form, set their properties and code their event handlers. Setting a component's properties changes the way a component appears

and behaves in your application. When a component is selected on a form, its properties and events are displayed in the Object Inspector.

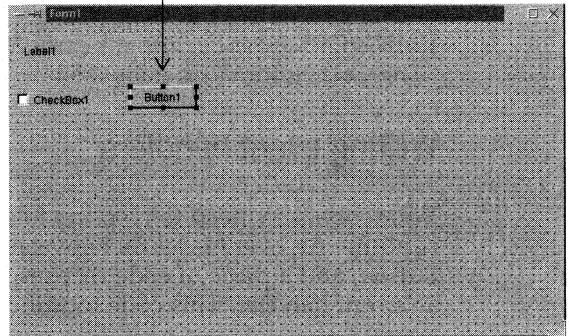


Or use this drop-down list to select an object. Here, Button1 is selected, and its properties are displayed.

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

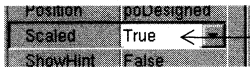
Click an ellipsis to open a dialog box where you can change the properties of a helper object.

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

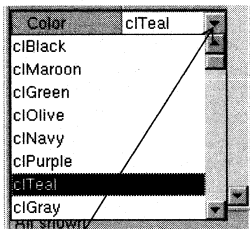


You can also click a plus sign to open a detail list.

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

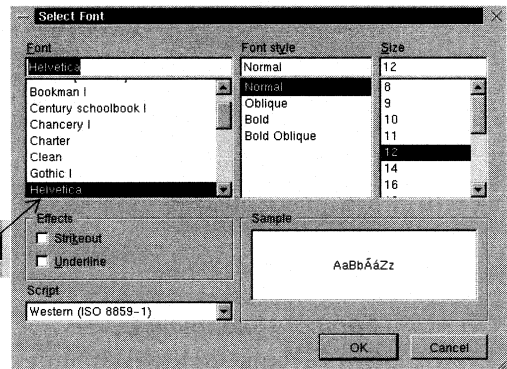
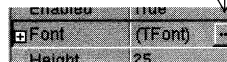


Double-click here to change the value from *True* to *False*.



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

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



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

For more information...

See "Object Inspector" in the Help index.

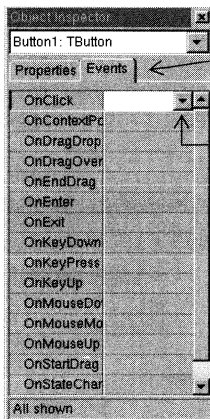
Writing code

An integral part of any application is the code behind each component. While Kylix's RAD environment provides most of the building blocks for you, such as 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 "The Code editor" on page 2-5.

Writing event handlers

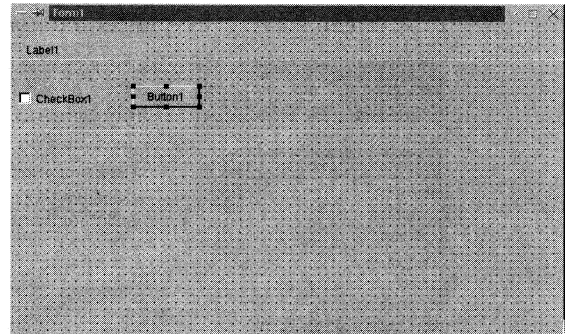
Your code may need to respond to events that might occur to a component at runtime. An event is a link between an occurrence in the system, such as clicking a button, and a piece of code that responds to that occurrence. The responding code is an event handler. This code modifies property values and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

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



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

Select an existing event handler from the drop-down list.
Or double-click in the value column, and Kylix generates skeleton code for a new event handler.



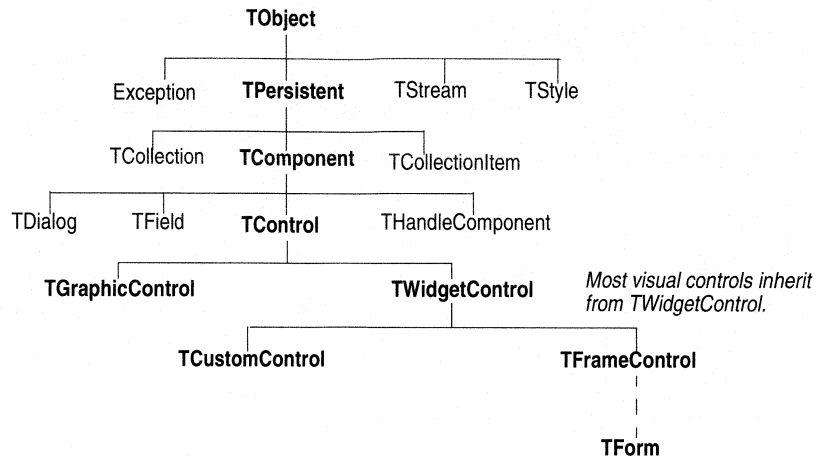
For more information...

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...

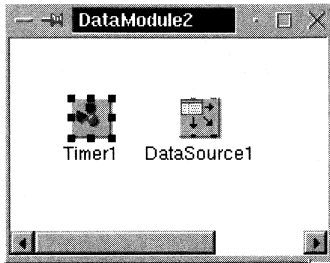
See "Kylix Object and Component Reference" in the Help contents. See <http://www.borland.com/kylix> for open source and licensing options on CLX.

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

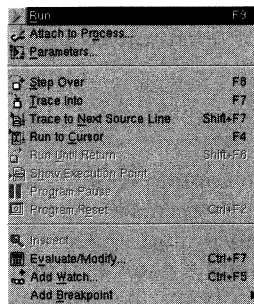
After you have written your code, you will need to compile and debug your project. With Kylix, you can either compile your project first and then separately debug it, or you can compile and debug in one step using the integrated debugger. To compile your program with debug information, choose Project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

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



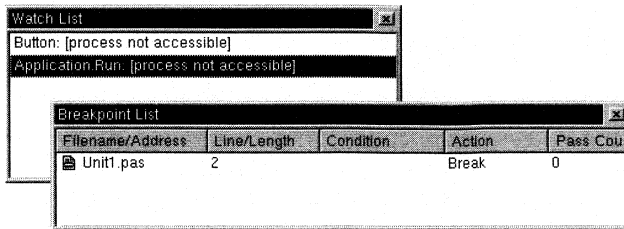
↑
Run button



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 6-2.

Once you set up your desktop as you like it for debugging, you can save the settings as the debugging or runtime desktop. This desktop layout will be used whenever you are debugging any application. For details, see “Saving desktop layouts” on page 6-4.

For more information...

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

Deploying applications

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.

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, database applications, and Web-based applications. To see what tools your edition supports, refer to the feature matrix on <http://www.borland.com/kylix>.

For more information...

See Chapter 5, “Building applications and shared objects,” in the *Developer’s Guide*.

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 several SQL databases, including DB2, Informix, InterBase, MySQL, and Oracle.

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. To use *dbExpress* and database components, click the Component palette *dbExpress* and *Data Access* pages. 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.

Client/server applications

To build multi-tiered, client/server database applications, you can use *DataSnap*. *DataSnap* defines the protocols and components that allow data-aware application servers and client applications to communicate. For the application server, you add a dataset and a dataset provider to a SOAP data module. The SOAP data module acts as a Web module in a Web services application to dispatch messages between client applications and the provider.

To build an application server, choose **File | New** and click the *WebServices* page in the *New Items* dialog box. Double-click the icon for the *SOAP Services Data Module*. To establish a connection with the server, click the Component palette *WebServices* page and select the *SoapConnection* component.

For more information...

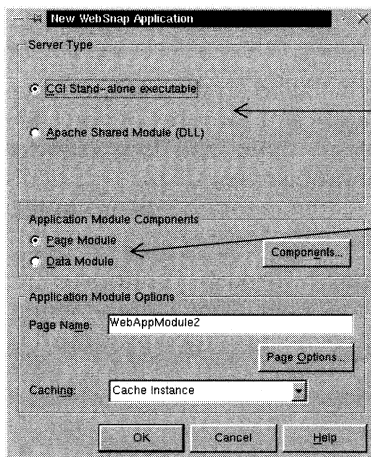
See “Using a multi-tiered architecture” in the *Developer’s Guide*.

Web server applications

Web server applications extend the functionality and capability 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. Many operations that you can perform with a Kylix application can be incorporated into a Web server application.

Kylix includes two different technologies, depending on which edition of Kylix you have. To create a Web server application with WebBroker, choose File | New and double-click the Web Server Application icon. You can add WebBroker components to your Web module from the Internet Component palette page.

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



To create a Web server application using WebSnap, you can choose from two different Web servers.

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

For more information...

See Part III, “Writing Internet applications,” in the *Developer’s Guide* and “Web server applications” in the Help index.

Web services

A Web service, such as an inventory tracking program, is an application that can be accessed over the Internet. Web services include well-defined interfaces that expose the service provided. The server implementation allows the client to use one or more communications methods and encoding schemes. Kylix’s BizSnap components are

designed to work with the HTTP and XML-based SOAP (Simple Object Access Protocol) protocols. With these components, you can write both servers that implement Web services and clients that call on those services.

To build a server using a wizard, choose File | New, click the WebServices page in the New Items dialog box, and double-click the Soap Server Application icon. To access components for the client, click the Component palette WebServices page.

For more information...

See “Using Web services” in the *Developer’s Guide*.

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 install components provided by a third party, see “Installing component packages” on page 6-7.

For more information...

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

Creating a text editor—a tutorial

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

Note This tutorial is for all editions of 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 Begin a new project by choosing `File | New Application` or use the default project that is already open when you started Kylix.

Each application is represented by a *project*. When you start Kylix, it creates a blank project by default, and 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.

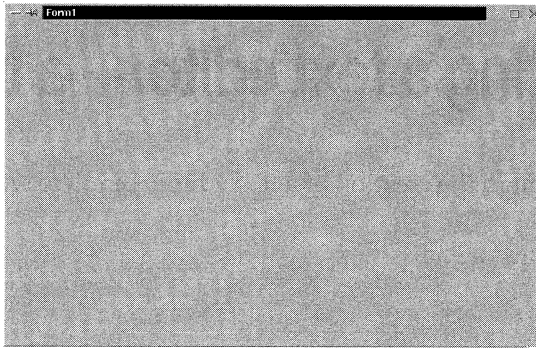
- 3 Choose `File | Save All` to save your files to disk. When the Save dialog box appears:
 - Navigate to your `TextEditor` folder.
 - Save `Unit1` using the default name `Unit1.pas`.

- 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.

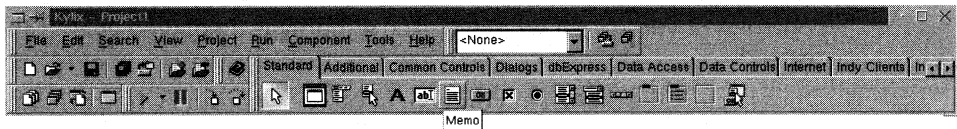
Kylix includes many components that represent parts of an application. For example, there are components (also called *objects*) on the Component palette that make it easy to program menus, toolbars, dialog boxes, and many other visual and nonvisual program elements.

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:

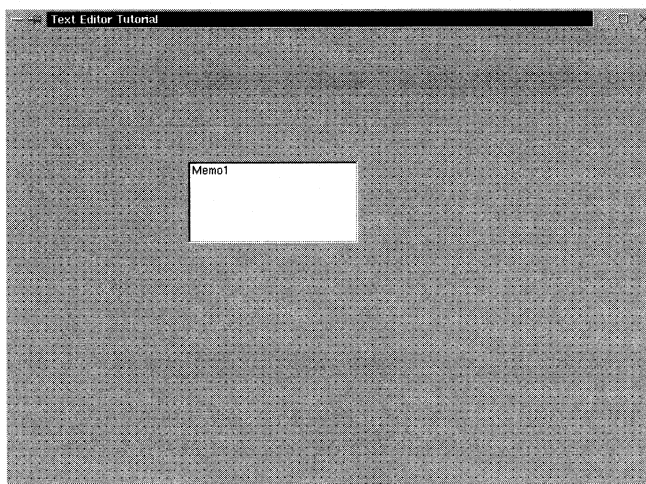


- 1 To create a text area, first add a *Memo* component. To find the *Memo* component, on the Standard tab of the Component palette, point to an icon on the palette for a moment; Kylix displays a Help tooltip showing the name of the component.



When you find the *Memo* component, either:

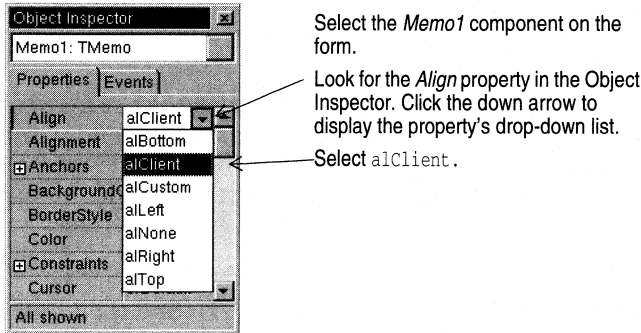
- Select the component on the palette and click on the form where you want to place the component; or
- Double-click it to place it in the middle of the form.




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.

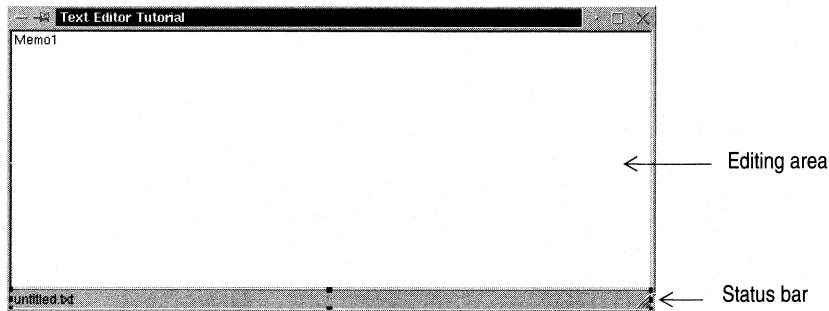
2 Set the *Align* property of *Memo1* to *alClient*.

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.



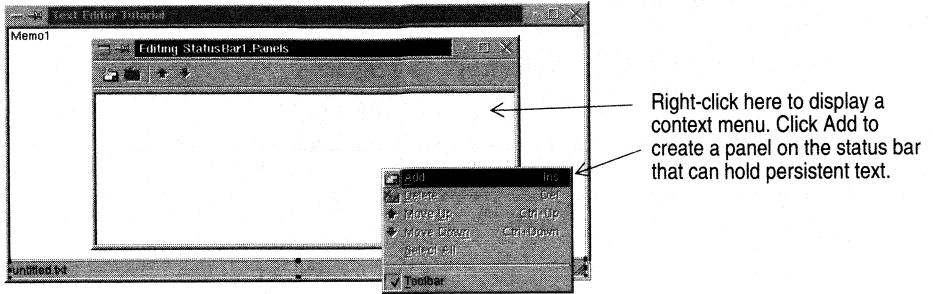
The *Memo* component now fills the entire form so you have a large text editing area. By choosing the *alClient* value for the *Align* property, the size of the *Memo* control will vary to fill whatever size window is displayed even if the form is resized.

-  **3** Double-click the *StatusBar* component on the Common Controls tab 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`.
 - Set *Simple Panel* to `True`.



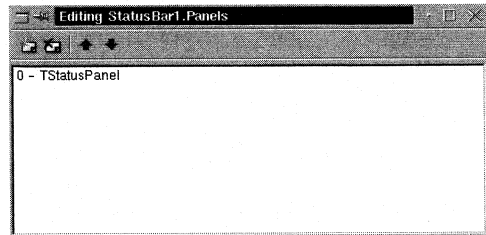
- Click the ellipse of the *Panels* property to open the `Editing StatusBar1.Panels` dialog box. You can stretch the dialog box to make it larger.

- 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:

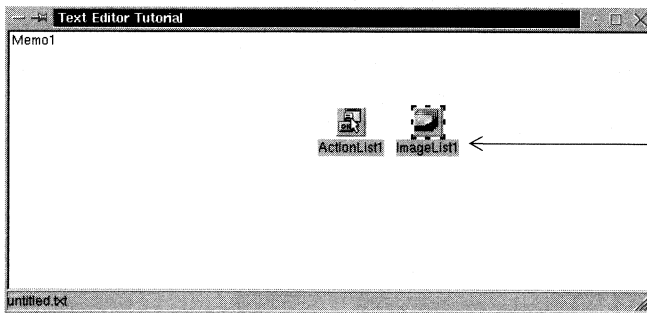


1 On the Standard tab of the Component palette, double-click the *ActionList* component to drop it onto the form.



2 On the Common Controls tab, 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.



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

Because the *ActionList* and *ImageList* components are nonvisual, you cannot see them when the application is running.

Adding actions to the action list

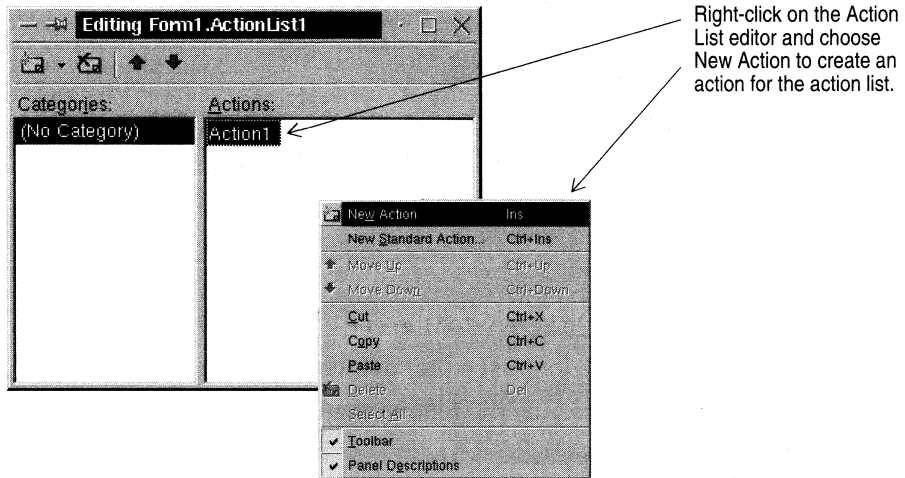
Next we'll add the actions to the action list.

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.



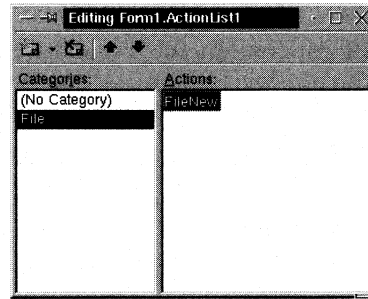
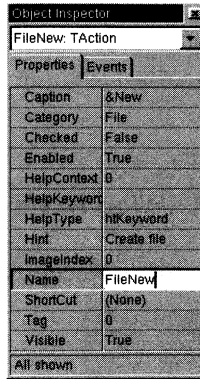
3 In the Object Inspector, set the following properties for the action:

- 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.

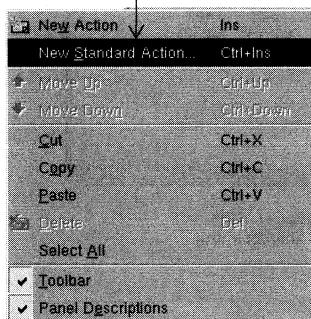
Note The Action List editor should still be displayed. If it's not, double-click the ActionList icon on the form.

To add standard actions to the action list:

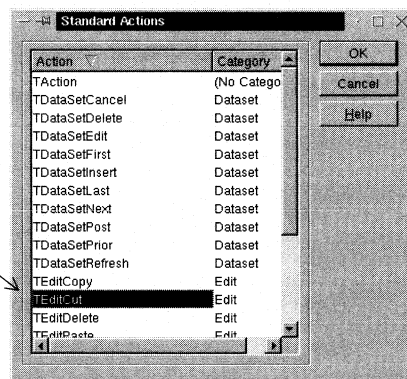
1 Right-click on the Action List editor and choose New Standard Action.

The Standard Actions dialog box appears.

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



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



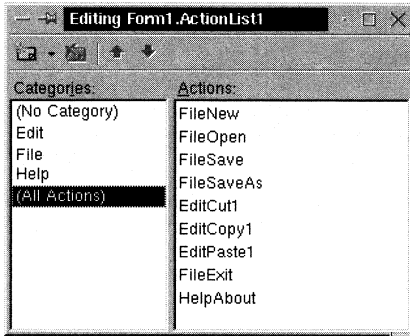
- Double-click TEditCut. The action is created in the Editing Form1.ActionList1 dialog box along with a new category called Edit. Select EditCut1.
- In the Object Inspector, set the *ImageIndex* property to 4.

The other properties are set automatically.

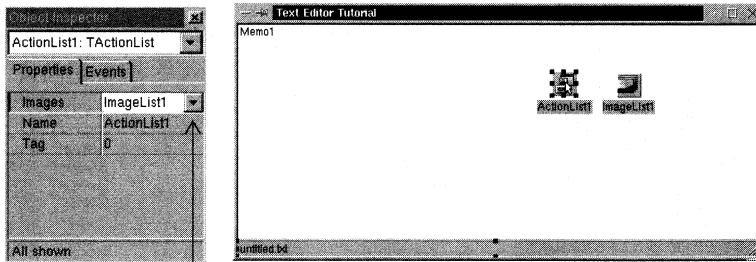
2 Right-click on the Action List editor and choose New Standard Action.

- 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.

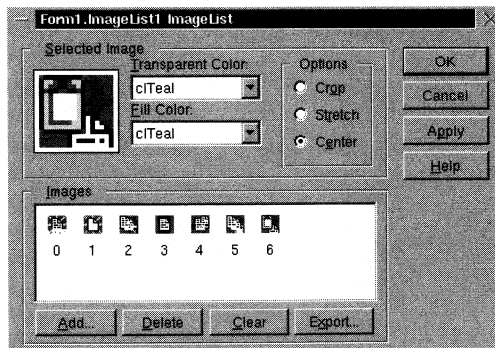
Adding images to the image list

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 New	Filenew.bmp	0
File Open	Fileopen.bmp	1
File Save	Filesave.bmp	2
File Exit	Doorshut.bmp	3
Edit Cut	Cut.bmp	4
Edit Copy	Copy.bmp	5
Edit Paste	Paste.bmp	6

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 /usr/local/kylix2 directory, look in /usr/local/kylix2/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.



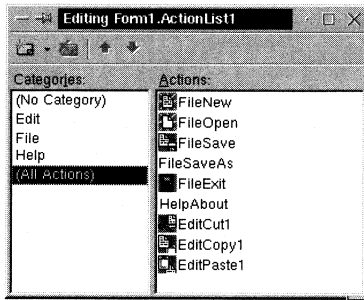
- 5 Click OK to close the Image List editor.

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.

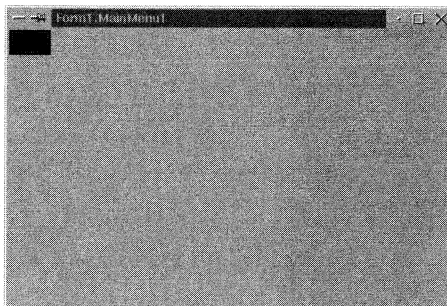
When you're done close the Action List editor. Now you're ready to add the menu and toolbar.

Adding a menu

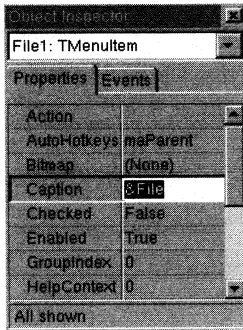
In this section, you'll add a main menu bar with three drop-down menus—File, Edit, and Help—and you'll add menu items to each one using the actions in the action list.



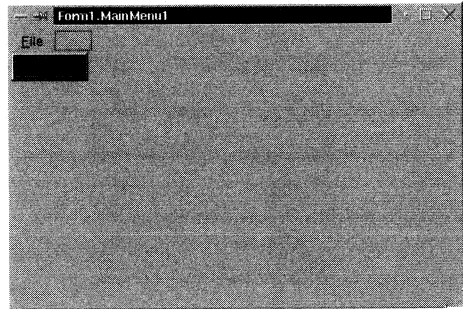
- 1 From the Standard tab of the Component palette, drop a *MainMenu* component onto the form. It doesn't matter where you place it.
- 2 Set the main menu's *Images* property to *ImageList1* so you can add the bitmaps to the menu items.
- 3 Double-click the main menu component to display the Menu Designer.



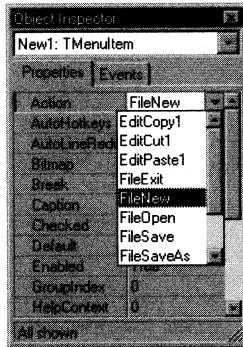
- 4 In the Object Inspector, after *Caption*, type *&File*, and press *Enter* to set the first top-level menu item.



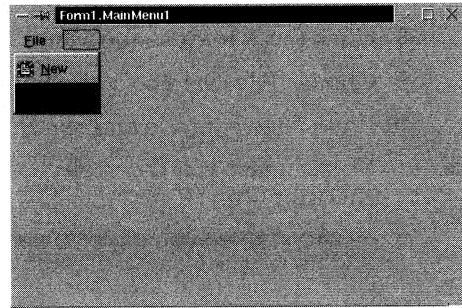
When you type *&File* and focus on the Menu Designer, the top-level File command appears ready for you to add the first menu item.



- 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*.

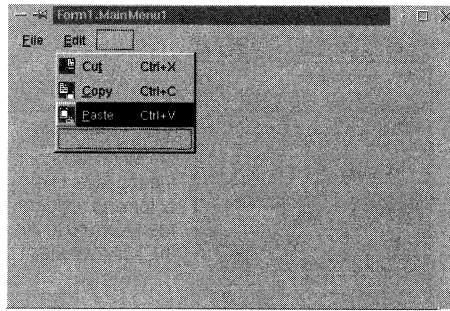
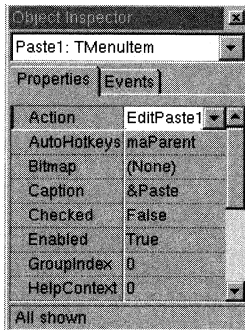


When you select *FileNew* from the *Action* property list, the New command appears with the correct Caption and ImageIndex.



- 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:

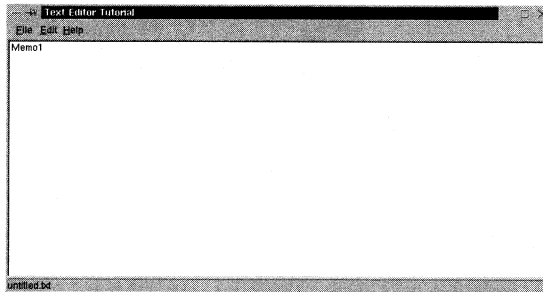
- Select the item to the right of the Edit command, set its *Caption* property to `&Help`, and press *Enter*.
- Select the item under Help and set its *Action* property to `HelpAbout`.

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 String List Editor. If you don't clear the text now, the text should be removed when initializing the main form in the last step.

To clear the text area:

- 1 On the main form, click the *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.

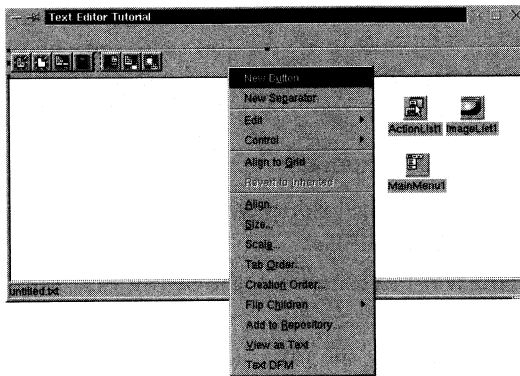


- 1 On the Common Controls tab of the Component palette, double-click the *ToolBar* component to add it to the form.

A blank toolbar is added under the main menu. 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.

You can create a skeleton handler, or an event handler without any code, in the Code editor. To create a skeleton handler, either double-click the component on the form or click the space to the right of an event in the Object Inspector. The skeleton handler is an empty event handler that includes the procedure name and a **begin** and **end** statement.

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

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

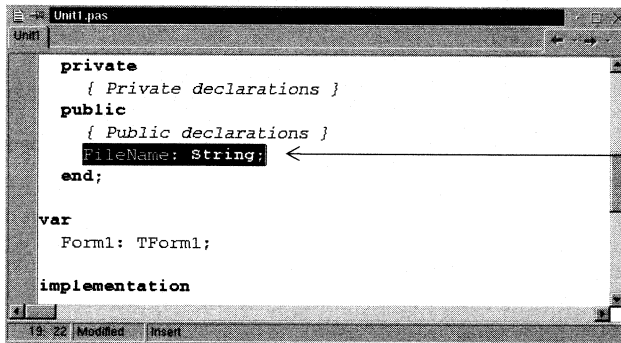
Creating an event handler for the New command

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:



```
Unit1.pas
Unit
private
  { Private declarations }
public
  { Public declarations }
  FileName: String;
end;

var
  Form1: TForm1;

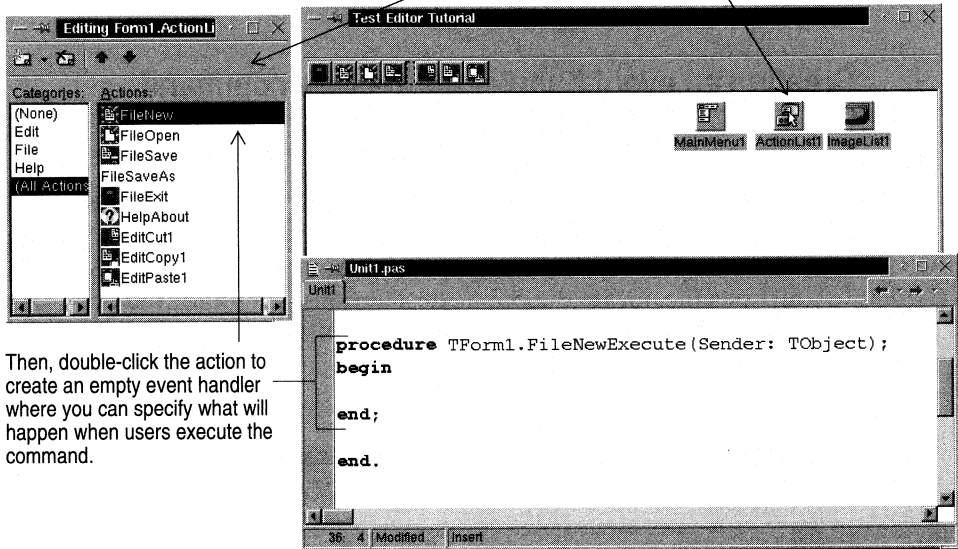
implementation
```

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

- 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.

First, double-click the Action List object to display the Action List editor.



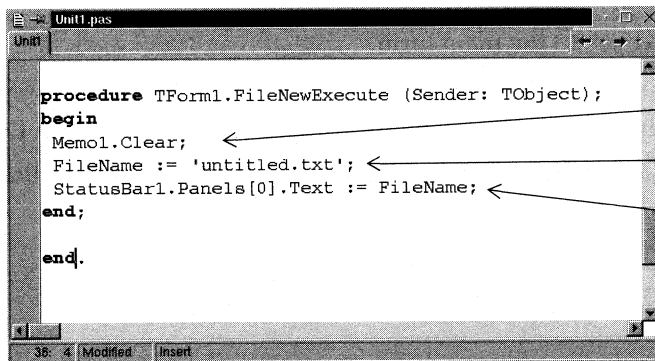
Then, double-click the action to create an empty event handler where you can specify what will happen when users execute the command.

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

```

Mem1.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.

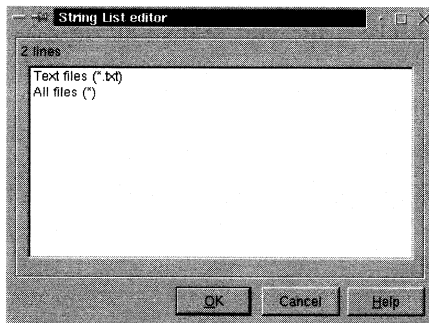
Tip You can resize the code portion of the window to reduce horizontal scrolling.

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 tab 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.

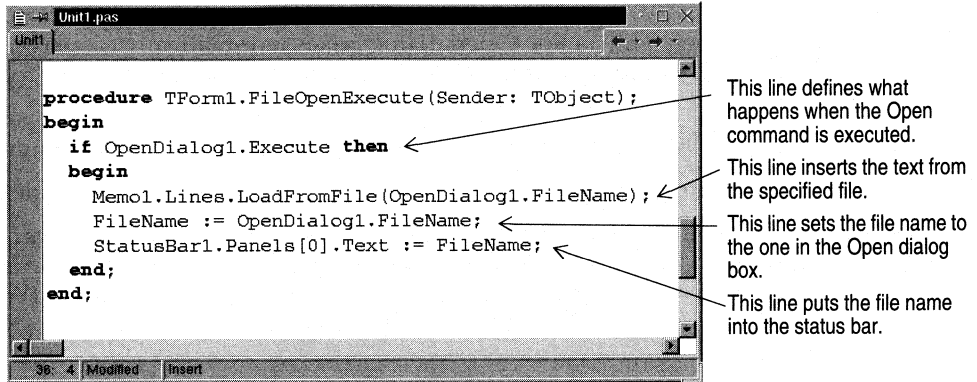


Use the String List editor to define filters for the *OpenDialog* and *SaveDialog* components.

- After *Title*, type *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:

```

procedure TForm1.FileSaveExecute(Sender: TObject);
begin
  if (FileName = 'untitled.txt') then
    FileSaveAsExecute(nil)
  else
    Mem1.Lines.SaveToFile(FileName);
end;
end.

```

This line states that if the file is untitled, the File Save As dialog box appears.
 Otherwise, the file is saved with the current file name.

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 tab 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.
 - Make sure *Title* is set 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
  Mem1.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:

```

Unit1.pas
Unit1

procedure TForm1.FileSaveAsExecute(Sender: TObject);
begin
  SaveDialog1.FileName := FileName;
  SaveDialog1.InitialDir := ExtractFilePath(FileName)
  if SaveDialog1.Execute then
  begin
    Mem1.Lines.SaveToFile(SaveDialog1.FileName);
    FileName := SaveDialog1.FileName;
    StatusBar1.Panels[0].Text := FileName;
  end;
end;

```

This sets the SaveAs dialog box's FileName property to the main form's FileName property value.

The default directory is set to the last one accessed.

This line saves the text to the specified file.

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

This puts the file name in the text panel of the status bar.

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.

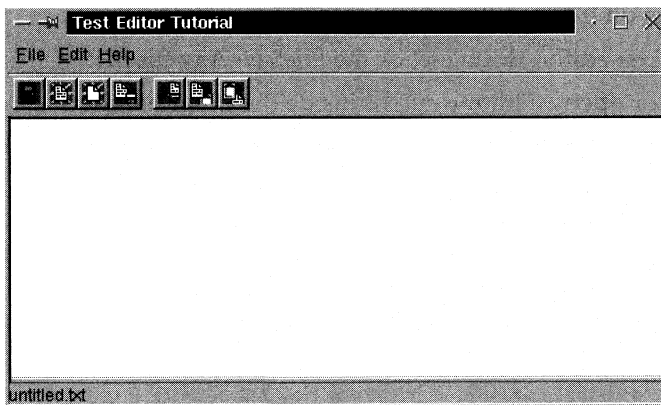
- 3 Right where the cursor is positioned in the Code editor, type the following line:

```
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.

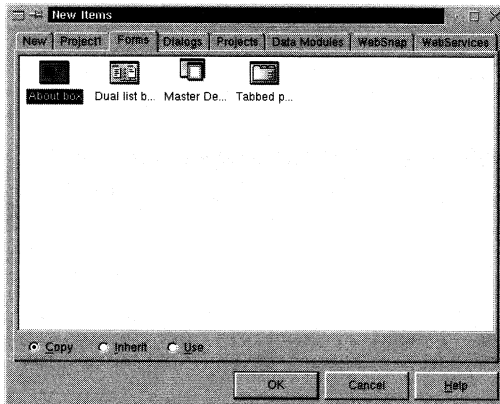
Creating an About box

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.

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

To create an About box:

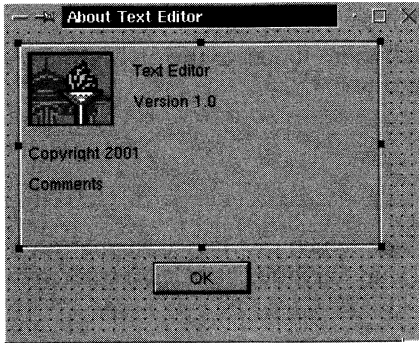
- 1 Choose File | New to display the New Items dialog box and click the Forms tab.
- 2 On the Forms tab, double-click About Box.



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.

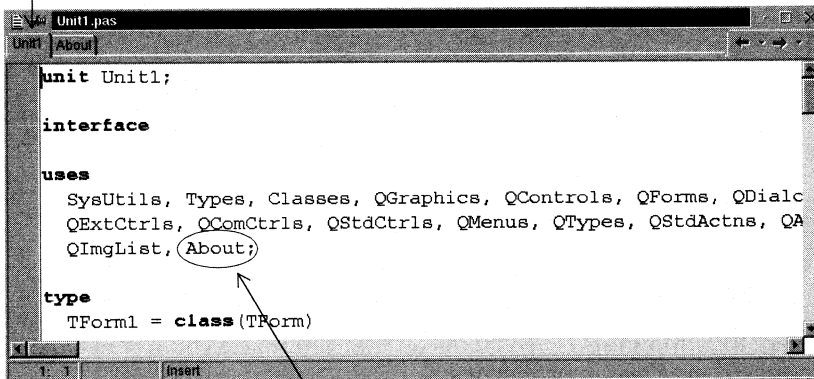
Tip The easiest way to select the form is to click the grid portion.



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 by typing the word About at the end of 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:

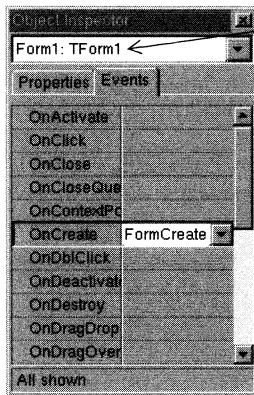
```
AboutBox.ShowModal;
```

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.)



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

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

- 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, double-click the error message and you'll go right to the place in the code where the error occurred.

You're done!

Creating a database application— a tutorial

This tutorial helps you create a database application with which you view and update a sample employee database.

Note This tutorial is written for the Kylix Enterprise and Professional editions, which include the database components. In addition, you must have InterBase installed to successfully complete this tutorial.

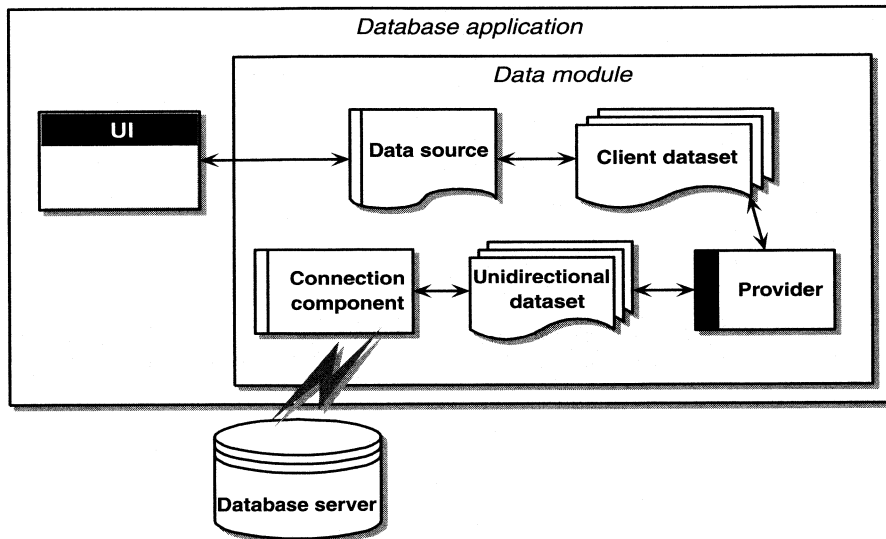
Overview of database architecture

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

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

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

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



Note For more information on database development, see "Designing database applications" in the *Developer's Guide* or online Help.

Creating a new project

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

- 1 Create a folder called Tutorial to hold the project files you'll create while working through this tutorial.
- 2 Use the default project already created when you start Kylix or begin a new project by choosing File | New Application.
- 3 Choose File | Save All to save your files to disk. When the Save As dialog box appears, navigate to your Tutorial folder and save each file using its default name.

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

Setting up data access components

Data access components are components that represent data (datasets), and the components that connect these datasets to other parts of your application. Each of

these data access components point to the next lower component. For example, the data source points to the client dataset, the client dataset points to the provider, and so forth. Therefore, when you set up your data access components, you add the components in the order that they are pointed to.

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

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

Setting up the database connection

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

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

To add a dbExpress connection component:



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

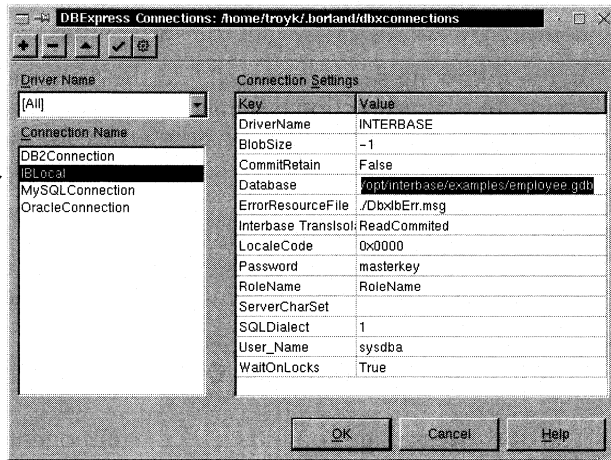
The *TSQLConnection* component is nonvisual, so it doesn't matter where you put it. However, for this tutorial, line up all the nonvisual components at the top of the form.

- Tip** To display the captions for the components you place on a form, choose Tools | Environment Options and click Show component captions.
- 2 In the Object Inspector, set its *ConnectionName* property to `IBLocal` (it's on the drop-down list).
 - 3 Set the *LoginPrompt* property to `False`. (By setting this property to `False`, you won't be prompted to log on every time you access the database.)
 - 4 Double-click the *TSQLConnection* component to display the Connection editor. You use the Connection editor to select a connection configuration for the *TSQLConnection* component or edit the connections stored in the `dbxconnections` file.
 - 5 In the Connection editor, specify the pathname of the database file called `employee.gdb` on your system. In this tutorial you will connect to a sample

InterBase database, `employee.gdb`, that is provided with Kylix. By default, the InterBase installation places `employee.gdb` in `/opt/interbase/examples`.

You can choose from several database drivers to connect to your database and then edit the connection settings.

You can add, delete, rename, and test your connections.



- 6 Check the `User_Name` and `Password` fields for acceptable values. If you have not altered the default values, you do not need to change the fields. If database access is administered by someone else, you may need to get a username and password to access the database.
- 7 When you are done checking and editing the fields, click **OK** to close the Connection editor and save your changes.

These changes are written to the `dbxconnections` file and the selected connection is assigned as the value of the `SQLConnection` component's `ConnectionName` property.

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

Setting up the unidirectional dataset

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

To add the unidirectional dataset:



- 1 From the dbExpress page, drop `TSQLDataSet` at the top of the form.
- 2 In the Object Inspector, set its `SQLConnection` property to `SQLConnection1` (the database connection created previously).
- 3 Set the `CommandText` property to `"Select * from sales"` to specify the command that the dataset executes. You can either type the Select statement in the Object Inspector or click the ellipsis to the right of `CommandText` to display the `CommandText` editor, where you can build your own query statement.
- 4 Set `Active` to `True` to open the dataset.
- 5 Choose **File | Save All** to save the project.

Setting up the provider, client dataset, and data source

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

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

To add the provider:



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

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

To add the client dataset:



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

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

To add the data source:



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

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

Designing the user interface

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

Creating the grid and navigation bar

To create the interface for the application:



- 1 You can start by adding a grid to the form. From the Data Controls page, drop a *TDBGrid* component onto the form.
- 2 Set *DBGrid's* properties to anchor the grid. Click the **+** next to *Anchors* in the Object Inspector to display *akLeft*, *akTop*, *akRight*, and *akBottom*; set them all to *True*. The easiest way to do this is to double-click *False* next to each property in the Object Inspector.
- 3 Align the grid with the bottom of the form by setting the *Align* property to *alBottom*. You can also enlarge the size of the grid by dragging it or setting its *Height* property to 400.
- 4 Set the grid's *DataSource* property to *DataSource1*. When you do this, the grid is populated with data from the employee database. If the grid doesn't display data, make sure you've correctly set the properties of all the objects on the form, as explained in previous instructions.

So far your application should look like this:

PO_NUMBER	CUST_NO	SALES_REP	ORDER_STATUS	ORDER_DATE	SHIP_DATE
V92F3004	1012	11	shipped	10/15/1992	01/16/1993
V92J1003	1010	61	shipped	07/26/1992	08/04/1992
V9320630	1001	127	open	12/12/1993	
V9324200	1001	72	shipped	08/09/1993	08/09/1993
V9324320	1001	127	shipped	08/16/1993	08/16/1993
V9333005	1002	11	shipped	02/03/1993	03/03/1993
V9333006	1002	11	shipped	04/27/1993	05/02/1993
V9336100	1002	11	waiting	12/27/1993	01/01/1994
V9345139	1003	127	shipped	09/09/1993	09/20/1993
V9345200	1003	11	shipped	11/11/1993	12/02/1993
V9346200	1003	11	waiting	12/31/1993	
V93B1002	1014	134	shipped	09/20/1993	09/21/1993
V93C0120	1006	72	shipped	03/22/1993	05/31/1993
V93C0990	1006	72	shipped	06/09/1993	09/02/1993
V93F0020	1009	61	shipped	10/10/1993	11/11/1993
V93F2030	1012	134	open	12/12/1993	
V93F2051	1012	134	waiting	12/18/1993	
V93F3086	1012	134	shipped	08/27/1993	09/08/1993

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



- 5 From the Data Controls page, drop a *TDBNavigator* control onto the form. A database navigator is a tool for moving through the data in a dataset (using next and previous arrows, for example) and performing operations on the data.

- 6 Set the navigator bar's *DataSource* property to `DataSource1` so the navigator is looking at the data in the client dataset.
- 7 Set the navigator bar's *ShowHint* property to `True`. (Setting *ShowHint* to `True` allows Help hints to appear when the cursor is positioned over each of the items on the navigator bar at runtime.)
- 8 Choose **File | Save All** to save the project.
- 9 Press **F9** to compile and run the project. You can also run the project by clicking the **Run** button on the **Debug** toolbar, or by choosing **Run** from the **Run** menu.



PO_NUMBER	CUST_NO	SALES_REP	ORDER_STATUS	ORDER_DATE	SHIP_DATE	DA
V93F3004	1012	11	shipped	10/15/1992	01/16/1993	01/
V92J1003	1010	61	shipped	07/26/1992	08/04/1992	09/
V9320630	1001	127	open	12/12/1993		12/
V9324200	1001	72	shipped	08/09/1993	08/09/1993	08/
V9324320	1001	127	shipped	08/16/1993	08/16/1993	09/
V9333005	1002	11	shipped	02/03/1993	03/03/1993	
V9333006	1002	11	shipped	04/27/1993	05/02/1993	05/
V9336100	1002	11	waiting	12/27/1993	01/01/1994	01/
V9345139	1003	127	shipped	09/09/1993	09/20/1993	10/
V9345200	1003	11	shipped	11/11/1993	12/02/1993	12/
V9346200	1003	11	waiting	12/31/1993		01/
V93B1002	1014	134	shipped	09/20/1993	09/21/1993	09/
V93C0120	1006	72	shipped	03/22/1993	05/31/1993	04/
V93C0990	1006	72	shipped	08/09/1993	09/02/1993	
V93F0020	1009	61	shipped	10/10/1993	11/11/1993	11/
V93F2030	1012	134	open	12/12/1993		
V93F2051	1012	134	waiting	12/18/1993		03/
V93F3088	1012	134	shipped	08/27/1993	09/08/1993	

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



Tip

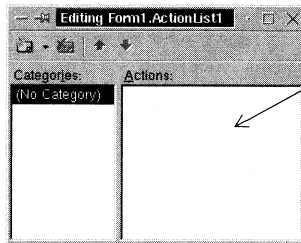
If you should encounter an error while testing an early version of your application, choose **Run | Program Reset** to return to the design-time view.

Adding support for a menu

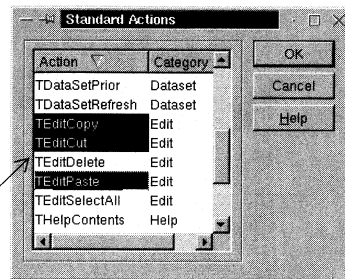
Though your program already has a great deal of functionality, it still lacks many features usually found in GUI applications. For example, most applications implement menus and buttons to make them easy to use.

In this section, you'll add an *action list*. While you can create menus, toolbars, and buttons without using action lists, action lists simplify development and maintenance by centralizing responses to user commands.

- 1 If the application is still running, click the **X** in the upper right corner to close the application and return to the design-time view of the form.
- 2  From the Common Controls page of the Component palette, drop an *ImageList* component onto the form. Line this up next to the other nonvisual components. The *ImageList* will contain icons that represent standard actions like cut and paste.
- 3  From the Standard page of the Component palette, drop an *ActionList* component onto the form. Set the action list's *Images* property to *ImageList1*.
- 4 Double-click the action list to display the Action List editor.
- 5 Right-click the Action List editor and choose New Standard Action. The Standard Actions list box appears.

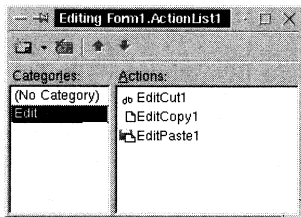


Right-click in the editor and choose New Standard Action to display the Standard Actions list box. Select the actions you want and click OK. Press *Ctrl* to select multiple actions.



- 6 Select the following actions: *TEditCopy*, *TEditCut*, and *TEditPaste*. (Use the *Ctrl* key to select multiple items.) Then click OK.

These standard actions appear in the Action List editor with default images already associated with them.



You've added three standard actions that come with the product. You'll use these on a menu.

Note To change the associated images, see "Adding images to the image list" on page 4-10 of the text editor tutorial.

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

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

- 8 Click (No Category), right-click and choose New Action to add another action. Action2 is added. Set its *Caption* property to *E&xit*.

- 9 Click the **X** (in the upper right corner) to close the Action List editor.

You've added three standard actions plus two other actions that you'll connect to event handlers later.

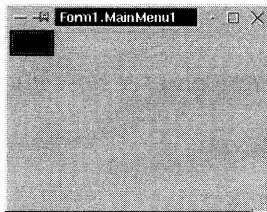
- 10 Choose File | Save All to save the project.

Adding a menu

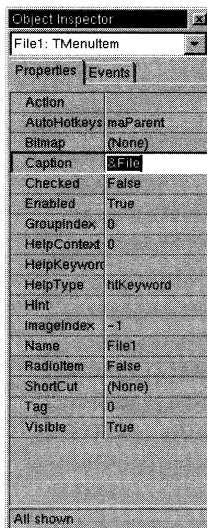
In this section, you'll add a main menu bar with two drop-down menus—File and Edit—and you'll add menu items to each one using the actions in the action list.



- 1 From the Standard page of the Component palette, drop a *TMainMenu* component onto the form. Drag it next to the other nonvisual components.
- 2 Set the main menu's *Images* property to *ImageList1* to associate the image list with the menu items.
- 3 Double-click the *TMainMenu* component to display the Menu Designer.

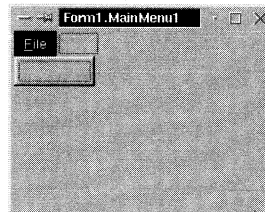


- 4 Type **&File** to set the *Caption* property of the first top-level menu item and press *Enter*.



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

The ampersand before a character activates an accelerator key.



- 5 Select the blank menu item below the File menu. Set the blank menu item's *Action* property to *Action2*. An Exit menu item appears under the File menu.

- 6 Click the second top-level menu item (to the right of File). Set its *Caption* property to &Edit and press *Enter*. Select the blank menu item that appears under the Edit menu.
- 7 In the Object Inspector, set the *Action* property to `EditCut1` and press *Enter*. The item's caption is automatically set to *Cut* and a default cut bitmap appears on the menu.
- 8 Select the next blank menu item (under *Cut*) and set its *Action* property to `EditCopy1` (a default copy bitmap appears on the menu).
- 9 Select the next blank menu item (under *Copy*) and set its *Action* property to `EditPaste1` (a default paste bitmap appears on the menu).
- 10 Select the next blank menu item (under *Paste*) and set its *Caption* property to a hyphen (-) to create a divider line in the menu. Press *Enter*.
- 11 Select the next blank menu item (under the divider line) and set its *Action* property to `Action1`. The menu item displays *Update Now!*
- 12 Click the **X** to close the Menu Designer.
- 13 Choose *File | Save All* to save the project.
- 14 Press *F9* or *Run* on the toolbar to run your program and see how it looks.

PO_NUMBER	CUST_NO	SALES_REP	ORDER_STATUS	ORDER_DATE
V93E3004	1012	11	shipped	10/15/1992
V92J1003	1010	61	shipped	07/26/1992
V9320630	1001	127	open	12/12/1993
V9324200	1001	72	shipped	08/09/1993
V9324320	1001	127	shipped	08/16/1993
V9333005	1002	11	shipped	02/03/1993
V9333006	1002	11	shipped	04/27/1993
V9336100	1002	11	waiting	12/27/1993
V9345139	1003	127	shipped	09/09/1993
V9345200	1003	11	shipped	11/11/1993
V9346200	1003	11	waiting	12/31/1993
V93B1002	1014	134	shipped	09/20/1993
V93C0120	1006	72	shipped	03/22/1993
V93C0990	1006	72	shipped	08/09/1993
V93F0020	1009	61	shipped	10/10/1993
V93F2030	1012	134	open	12/12/1993
V93F2051	1012	134	waiting	12/18/1993
V93F3088	1012	134	shipped	08/27/1993

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

Adding a button

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

To add a button:



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

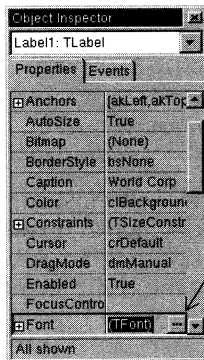
The button's caption changes to Update Now! When you run the application, it will be grayed out until an event handler is added to make it work.

Displaying a title and an image

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

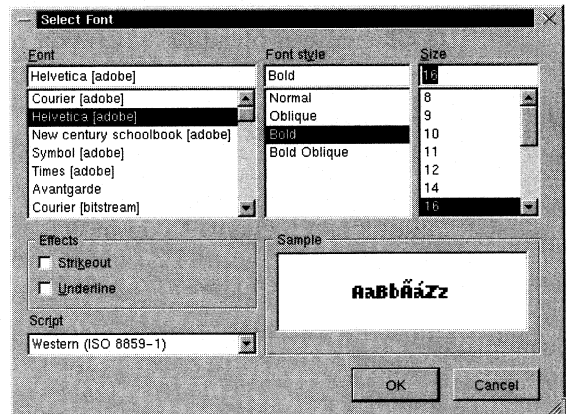


- 1 From the Standard page of the Component palette, drop a *TLabel* component onto the form. Kylix names this *Label1* by default.
- 2 In the Object Inspector, change the label's *Caption* property to *World Corp* or another company name.
- 3 Change the company name's font by clicking the *Font* property. Click the ellipsis that appears on the right and in the Font dialog box, change the font to Helvetica Bold, 16-point type. Click OK.



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

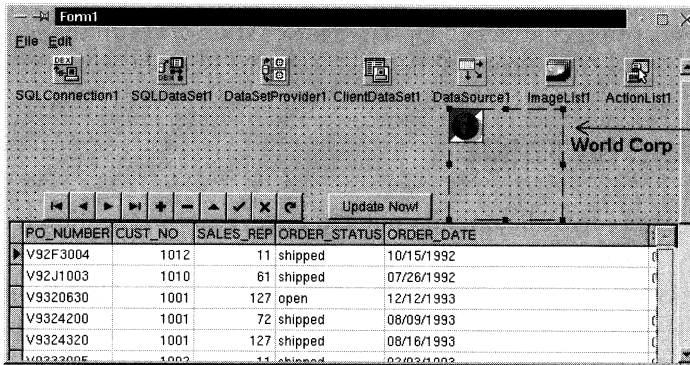
Click the ellipsis to display a standard font dialog box. →



- 4 Position the label in the upper right corner.
- 5 From the Additional Component palette page, drop a *TImage* component next to the label. Kylix names the component *Image1* by default.



- 6 To add an image to the *Image1* component, click the *Picture* property. Click the ellipsis to display the Picture editor.
- 7 In the Picture editor, choose Load and navigate to the icons directory provided with the product. The default location is {install directory}/images/icons. For example, if Kylix is installed in your /usr/local/kylix2 directory, look in /usr/local/kylix2/images/icons.
- 8 Double-click earth.ico. Click OK to load the picture and to close the Picture editor.
- 9 Size the default image area to the size of the picture. Place the image next to the label.



You can set the size of the *Image1* component to match the size of the picture inside it in two ways: drag the edge of *Image1*, or change the *Width* and *Height* properties in the Object Inspector.

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

Writing an event handler

Most components on the Component palette have events, and most components have a default event. A common default event is *OnClick*, which gets called whenever a component, such as *TButton*, is clicked. If you select a component on a form and click the Object Inspector's Events tab, you'll see a list of the component's events.

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

Writing the Update Now! command event handler

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

- 1 Double-click the *ActionList* component to display the Action List editor.
- 2 Select (No Category) to see Action1 and Action2.

- 3 Double-click Action1. In the Code editor, the following skeleton event handler appears:

```
procedure TForm1.Action1Execute(Sender: TObject);
begin

end;
```

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

```
if ClientDataSet1.State in [dsEdit, dsInsert] then ClientDataSet1.Post;
ClientDataSet1.ApplyUpdates(-1);
```

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

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

Writing the Exit command event handler

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

- 1 Double-click the *ActionList* component to display the Action List editor if it is not already displayed.
- 2 Click (No Category) so you see Action2.
- 3 Double-click Action2. The Code editor displays the following skeleton event handler:

```
procedure TForm1.Action2Execute(Sender: TObject);
begin

end;
```

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

```
Close;
```

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

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

Writing the FormClose event handler

Finally, you'll write another event handler that is invoked when the application is closed. The application can be closed either by using File | Exit or by clicking the **X** in the upper right corner. Either way, the program checks to make sure that there are no

pending updates to the database and displays a message window asking the user what to do if changes are pending.

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

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

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin

end;
```

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

```
Action := caFree;
if ClientDataSet1.State in [dsEdit, dsInsert] then
  ClientDataSet1.Post;
if ClientDataSet1.ChangeCount > 0 then
begin
  Option := Application.MessageBox('You have pending updates. Do you want to write them
  to the database?', 'Pending Updates', [smbYes, smbNo, smbCancel],
  smsWarning, smbYes);
  case Option of
    smbYes: ClientDataSet1.ApplyUpdates(-1);
    smbCancel: Action := caNone;
  end;
end;
```

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

- 4 You need to declare the variable used within the procedure. On a line between **procedure** and **begin** type:

```
var
  Option: TMessageButton;
```

5 Check that the whole procedure looks like this:

```

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var
    Option: TMessageButton;
begin
    Action := caFree;
    if ClientDataSet1.State in [dsEdit, dsInsert] then
        ClientDataSet1.Post;
    if ClientDataSet1.ChangeCount > 0 then
        begin
            Option := Application.MessageBox('You have pending updates. Do you want to write them
                to the database?', 'Pending Updates', [smbYes, smbNo, smbCancel],
                smsWarning, smbYes);
            case Option of
                smbYes: ClientDataSet1.ApplyUpdates(-1);
                smbCancel: Action := caNone;
            end;
        end;
    end;
end;

```

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

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

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

Customizing the desktop

This chapter explains some of the ways you can customize the tools in 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 a new way your desktop looks.

Arranging menus and toolbars

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

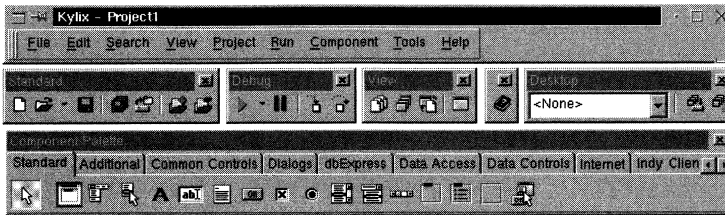
You can move toolbars and menus within the main window. Drag the grabber (the double bar on the left) of an individual toolbar to move it.



Main window organized differently.

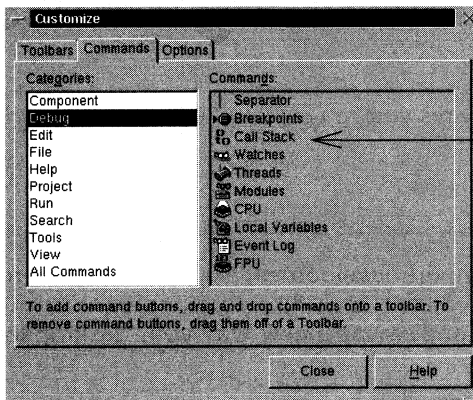
Organizing your work area

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



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.

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

For more information...

See “toolbars, customizing” in the Help index.

Docking tool windows

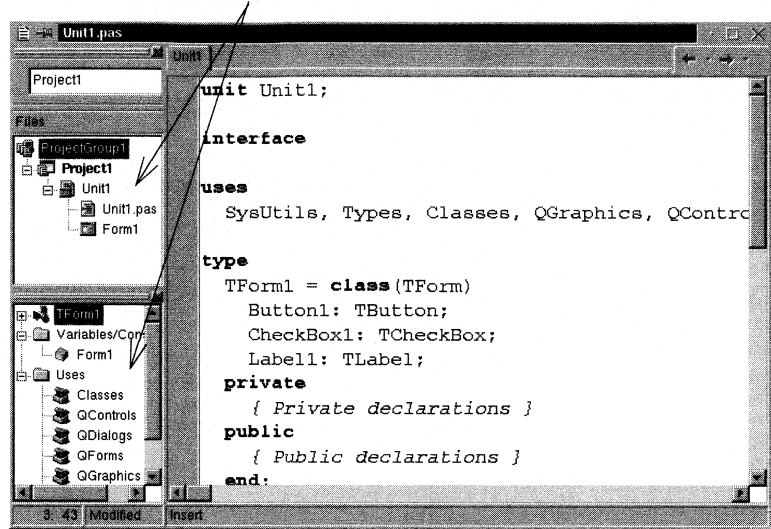
You can open and close individual tool windows and arrange them on the desktop as you wish. Many windows can also be *docked* to one another for easy management. Docking—which means attaching windows to each other so that they move together—helps you use screen space efficiently while maintaining fast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

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

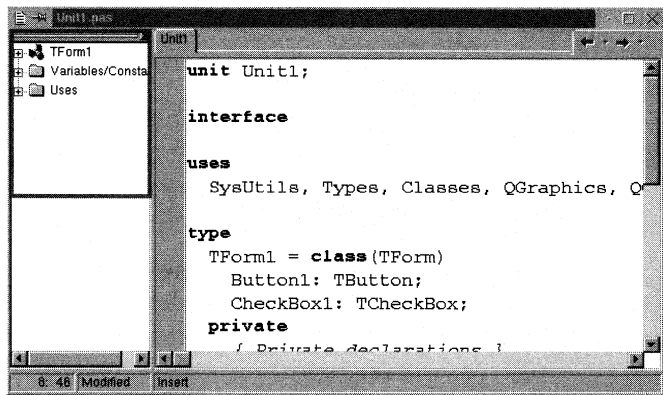
Here the Code Explorer and Project Manager are docked to the Code editor.

You can combine, or "dock" windows with either grabbers, as on the left, or tabs, as on page 6-4.

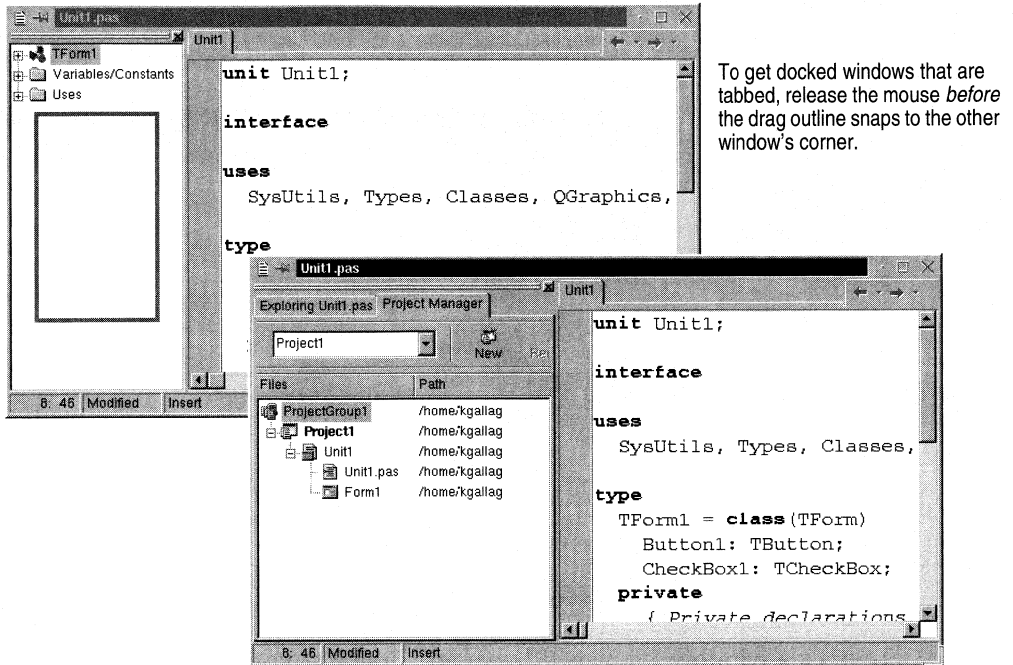


To dock a window, click its title bar and drag it over the other window. When the drag outline narrows into a rectangle and it snaps into a corner, release the mouse. The two windows snap together.

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



You can also dock tools to form a tabbed window.



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

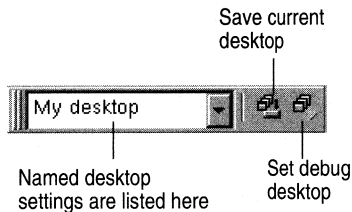
To turn off automatic docking, press the *Ctrl* key while moving windows around the screen.

For more information...

See “docking” in the Help index.

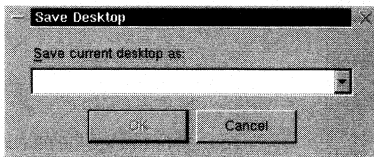
Saving desktop layouts

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:

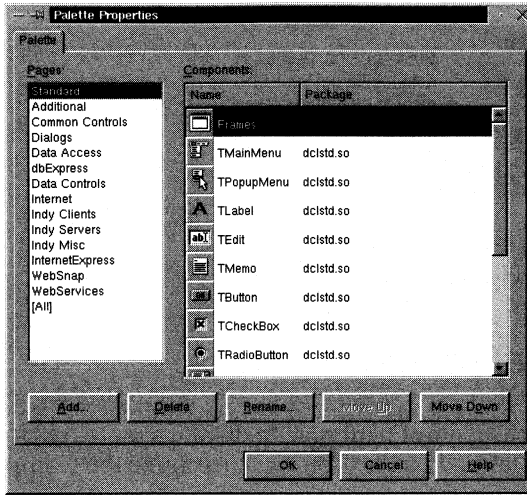
- Hiding or rearranging components.
- Adding, removing, rearranging, or renaming pages.
- Creating component templates and adding them to the palette.
- Installing new components.

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...

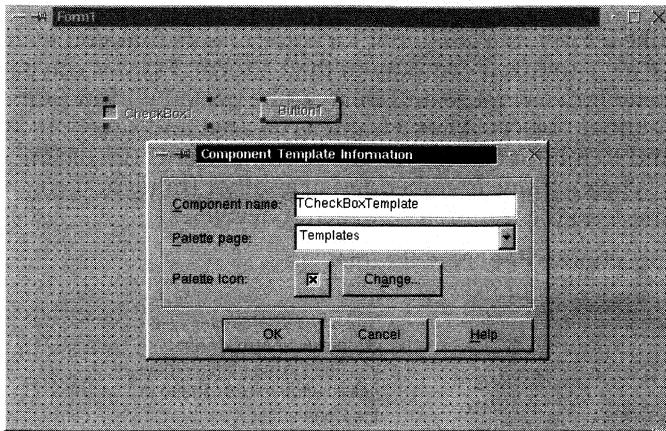
Click the Help button in the Palette Properties dialog box.

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...

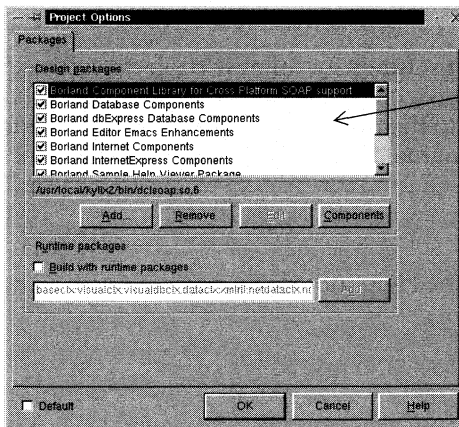
See “templates, component” in the Help index.

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 appears in this list.

Click Components to see what components the package contains.

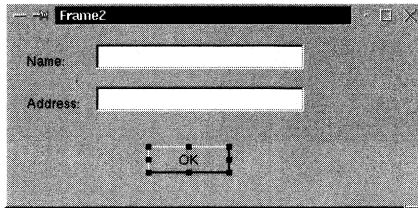
For more information...

See “installing components” and “packages” 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.

Setting project options

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

Setting default project options

To save your selections as the default settings for all new projects, in the lower-left corner of the Project Options dialog box, check Default. Checking Default writes the current settings from the dialog box to the options file `defproj.kof`. To restore Kylix’s original default settings, delete or rename the `defproj.kof` file, which is located in `{home directory}/.borland`.

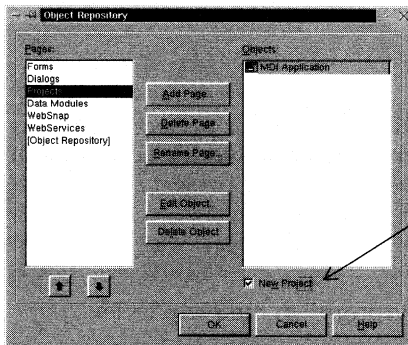
For more information...

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 templates to the Object Repository” below).

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, check New Project, and click OK.



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

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

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

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

In the same way that you specify a default project, you can specify a *default 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...

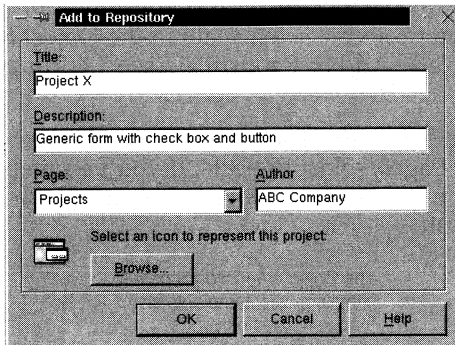
See “templates, adding to Object Repository,” “projects, specifying default,” and “forms, specifying default” in the Help index.

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 6-9.

For more information...

See "templates, adding to the Object Repository" 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; you can also display or hide the names, or *captions*, of nonvisual components you place on your form.

For more information...

In the Environment Options dialog box, click the 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 “Code Insight tools” on page 2-6.

For more information...

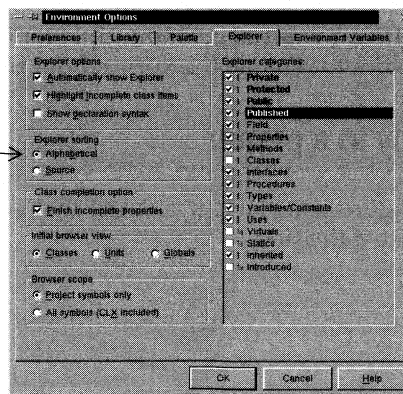
In the Editor Options dialog box, click the Help button on the General, Display, Key Mappings, Color, and Code Insight pages.

Customizing the Code Explorer

When you start Kylix, the Code Explorer (described in “The Code Explorer” on page 2-8) 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.

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

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



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

For more information...

See “Code Explorer, Environment options” in the Help index.

Printing Help topics

This section covers printing Kylix Help topics and configuring Kylix to work with your printer. You can print a Kylix Help topic in two ways. You can either print a topic to a PostScript file and then to a printer, or you can configure your system to print directly to a printer.

Note Your Linux machine must be configured to print. If your Linux machine is not already configured, edit the `/etc/printcap` file. To configure this file, check either your distribution-specific documentation, the Printing-HOWTO file, or the Printing-Usage-HOWTO file at <http://www.linuxdoc.org>.

Printing to a file

By default, the Kylix Help system is configured to print to a PostScript file. You can print a Help topic to a PostScript file and then print that file to a printer by doing the following:

- 1 Choose Help, select a Help menu item below Kylix Help, and click Print. The Topics dialog box appears.
- 2 Select the topic you want to print and click OK.

The Help system produces a PostScript file in the user's home directory. After a topic prints to file, a message box appears, stating that Output is in `/home/username/xprinter.out`.

- 3 If you now want to print the file to your printer and your printer supports PostScript, from a shell, go to your home directory and type `lpr xprinter.out`.

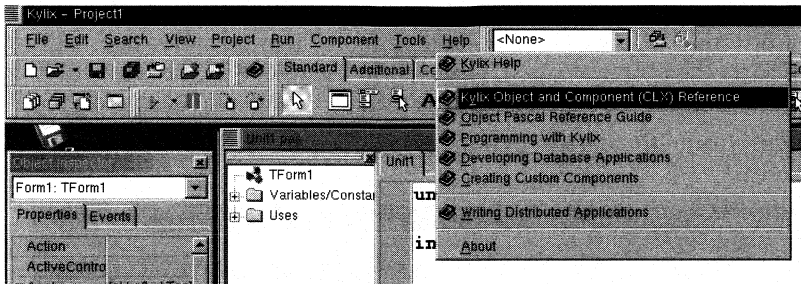
If your printer does not support PostScript, you can still print the `xprinter.out` file using a utility such as Ghostview.

Note When printing to a file, Kylix overwrites the existing `xprinter.out` file, assuming it already exists. If you want to print multiple topics to files before sending them to the printer, be sure to rename the `xprinter.out` file after printing each topic.

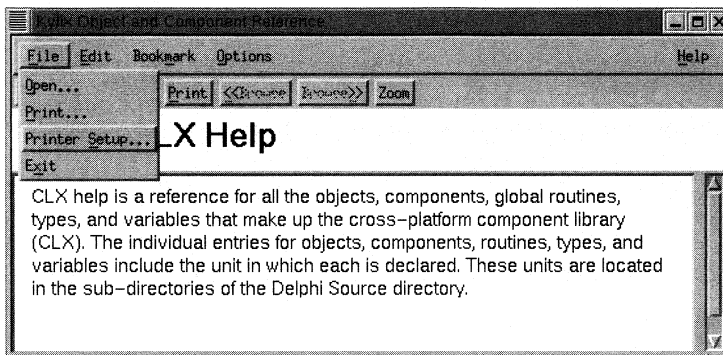
Printing directly to a printer

You can also configure your Help system so that you always print directly to the printer instead of to a file first.

- 1 Open the Help system by choosing Help and selecting a Help menu topic below Kylix Help.

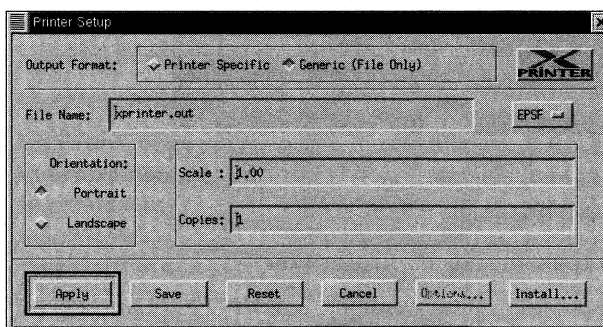


- 2 Once in the Help system, choose File | Printer Setup.



The Printer Setup dialog box appears.

- 3 To add a specific printer to your system, check the Printer Specific check box and click the Install button.

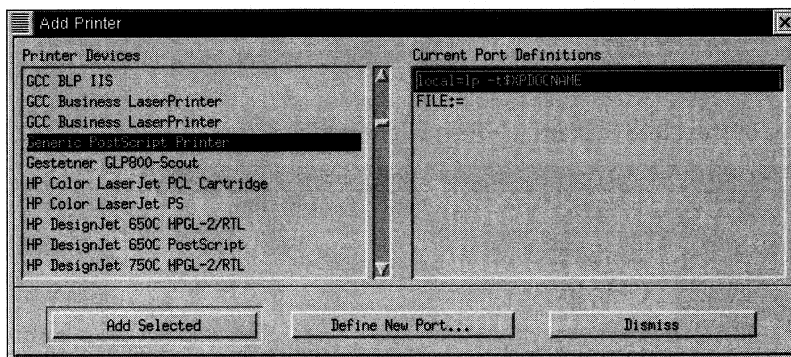


- 4 From the Printer Installation dialog box, click Add Printer. The Add Printer dialog box appears.
- 5 Below the Printer Devices column, select your printer from the list. If your printer supports PostScript printing but is not listed, select the Generic PostScript Printer.

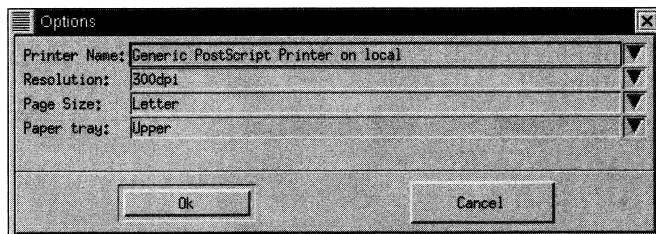
- 6 Under the Current Port Definitions column, select `local=lp -t$XPDOCNAME`.

Note Depending on which print service software is included with your Linux distribution, you may need to change the command syntax under Current Port Definitions. For instance, Red Hat version 7.0 and higher uses LPRng (which supports the above syntax), while earlier versions of Red Hat include a version of lpr that requires you to change the port definition to `local=lpr -T$XPDOCNAME`.

Tip To determine the correct syntax, check the man page for lp and lpr on your system.



- 7 Once you select the correct printer device and port definition, click Add Selected and then Dismiss. This connects the print command to the correct printer.
- 8 Click Dismiss again.
- 9 In the Printer Setup dialog box, click Options to open the Options dialog box.



- 10 In the Printer Name field, select the newly configured local printer and click OK.
- 11 In the Printer Setup dialog box, click Apply.
You are now ready to print directly to a printer.
- 12 Choose Help, select a Help menu item below Kylix Help, and click Print. The Topics dialog box appears.
- 13 Select the topic you want to print and click OK.

Index

A

- About box, adding 4-23
- actions, adding to an application 4-7, 4-9
- adding
 - components to a form 4-3, 4-12
 - items to Object Repository 2-4
- applications
 - client/server 3-8
 - compiling and debugging 3-6, 4-14
 - creating 3-1
 - database 3-8, 5-1
 - deploying 3-7
 - designing user interface 3-1
 - internationalizing 3-7
 - Web server 3-9
 - Web services 3-9

B

- bitmaps, adding to an application 4-10
- BizSnap 3-9

C

- character sets, extended 3-7
- class completion 2-7
- classes, using CLX 3-4
- client/server applications, creating 3-8
- CLX
 - class library 3-4 to 3-5
 - components 2-3
- code
 - help in writing 2-6
 - using the Code editor 2-5
 - viewing and editing 2-5
 - writing 3-4, 4-16 to 4-23, 5-12 to 5-15
- Code editor
 - combining with other windows 6-2
 - customizing 6-11
 - using 2-5 to 2-6
- Code Explorer
 - customizing 6-11
 - using 2-8
- Code Insight tools 2-6
- combining windows (docking) 6-2
- compiling programs 3-6, 4-14, 5-7
- Component Library for Cross Platform (CLX)
 - defined 3-4
 - diagram 3-5

- Component palette
 - adding custom components 3-10
 - adding pages 6-5
 - customizing 6-5 to 6-7
 - defined 2-3
 - using 3-2
- component templates, creating 6-6
- components
 - adding to a form 3-2, 4-3
 - adding to Component palette 6-5
 - arranging on the Component palette 6-5
 - CLX class library 3-5
 - creating 3-10
 - customizing 3-1, 6-6
 - installing 3-1, 6-7
 - setting properties 3-2, 4-2
- context menus, accessing 2-2
- customizing
 - Code editor 6-11
 - Code Explorer 6-11
 - Component palette 2-2
 - Form Designer 6-10
 - IDE 6-1 to 6-11

D

- data modules 2-4, 3-5
- database applications
 - accessing 5-3 to 5-4
 - creating 5-1 to 5-15
 - overview 3-8, 5-1
- database drivers 3-8
- database tutorial 5-1 to 5-15
- DataSnap 3-8
- dbExpress 3-8, 5-1
- debugging programs 3-6
- default
 - project and form templates 6-9
 - project options 6-8
- deploying programs 3-7
- design-time view 4-2
- desktop
 - layouts, saving 6-4
 - organizing 6-1 to 6-5
- developer support 1-6
- dialog boxes, templates for 2-4
- distributed applications 3-8
- docking windows 6-2 to 6-4
- documentation, ordering 1-6
- .dpr files 4-1

E

Editor Options dialog box 2-6, 6-11
Environment Options dialog box 2-7, 6-10
error messages 4-23, 4-25
event handlers
 creating 4-16 to 4-23, 5-12 to 5-15
 defined 3-4
events 4-16, 5-12
Events page (Object Inspector) 3-4, 5-12
executables 3-7

F

files
 configuration 4-2
 form 2-8, 4-1
 project 4-1
 saving 4-1
 types 4-1, 4-2
 unit 4-1
Form Designer
 customizing 6-10
 defined 2-1
 using 2-3
form files
 defined 4-1
 viewing code 2-8
forms
 adding components to 3-1, 4-3
 closing 4-2
 in Object Repository 2-4
 main 4-2, 6-9
 specifying as default 6-9
frames 6-8

G

global symbols 2-10
GUIs, creating 3-1

H

Help system
 accessing 1-5 to 1-6
 printing 6-12
Help tooltips 4-3

I

IDE
 customizing 6-1 to 6-11
 defined 1-1
 organizing 6-1
 tour of 2-1
images, adding to an application 4-10

initialization files, deploying programs 3-7
input method editors (IMEs) 3-7
Installing 1-3
installing
 custom components 6-7
 Kylix 1-2 to 1-4
integrated debugger 3-6
integrated development environment (IDE)
 customizing 6-1 to 6-11
 tour of 2-1
internationalizing applications 3-7

K

keystroke mappings 6-11
Kylix
 customizing 6-1 to 6-11
 installing 1-2
 overview 1-1
 printing Help topics 6-12
 programming 3-1
 registering 1-4
 starting 1-4
 uninstalling 1-7

L

localizing applications 3-7

M

menus
 adding to an application 4-12
 context 2-2
 in Kylix 2-2
 organizing 2-2 to 2-3, 6-1
messages, error 4-23, 4-25
modules, data 3-5
multi-tier applications, creating 3-8

N

New Items dialog box
 saving templates to 6-9, 6-10
 using 2-4, 4-23
newsgroups 1-6

O

Object Inspector 3-4
 defined 2-3
 using 3-2, 3-4, 4-2
Object Repository
 adding templates to 2-5, 6-9
 defined 2-4
 using 2-4 to 2-5, 3-1

- objects
 - adding to a form 4-4
 - defined 3-4, 4-3
- online Help
 - accessing 1-5 to 1-6
 - printing 6-12
- options
 - Code editor 6-11
 - Code Explorer 6-11
 - default project 6-8

P

- packages
 - defined 6-7
 - installing component 6-7
- .pas files 4-1
- printing Kylix Help 6-12
- programming with Kylix, overview 3-1
- programs
 - adding data modules 3-5
 - compiling and debugging 3-6, 4-14
 - database 3-8
 - deploying 3-7
 - internationalizing 3-7
 - Web server applications 3-9
 - Web services 3-9
 - writing code 3-4
- Project Browser 2-10
- project groups 2-9
- Project Manager 2-9
- Project Options dialog box 6-8
- projects
 - adding items to 2-4, 2-5
 - adding templates 6-9
 - creating 3-1
 - custom components 3-10
 - default files 4-1
 - managing 2-9
 - saving 4-1
 - setting options as default 6-8
 - shared objects 3-10
 - specifying as default 6-9
 - types 3-8 to 3-10
- properties, setting 3-2, 4-2, 4-7

R

- registering Kylix 1-4
- requirements, system 1-2
- resbind, localizing 3-7
- resource files (.res) 4-2
- right-click menus 2-2
- Run button 5-7
- running applications 3-6, 4-14, 5-7

S

- saving
 - desktop layouts 6-4
 - projects 4-1
- setting properties 3-2, 4-2, 4-7
- shared objects 3-10
- source code
 - CLX 3-5
 - files 4-1
 - help in writing 2-6
- standard actions, adding to an application 4-9
- starting Kylix 1-4
- support services 1-6
- system requirements 1-2

T

- technical support 1-6
- templates
 - adding to Repository 2-5, 6-9
 - specifying as default 6-9
- text editor tutorial 4-1 to 4-25
- to-do lists 2-10
- toolbars
 - adding and deleting components from 6-2
 - adding to an application 4-15
 - organizing 6-1
 - using 2-2 to 2-3
- tooltips, viewing 4-3
- tutorials
 - database 5-1 to 5-15
 - text editor 4-1 to 4-25
- typographic conventions 1-7

U

- uninstalling Kylix 1-7
- unit files 4-1
- user interface, designing 3-1, 4-3

W

- Web server applications 3-9
- Web services 3-9
- Web site, Borland 1-6
- WebSnap 3-9
- windows, docking 6-2
- wizards (Object Repository) 2-4
- writing code, overview 3-4

X

- .xfrm files 2-8, 4-1

