Borland

A quick tour of Kylix™

Table of Contents

Starting Kylix 2
Getting started: The IDE 2
Controlling Kylix: The menu and toolbars 2
Managing projects: The Project Manager 3
Browsing project structure and elements: The Project Browser 3
Adding items to your projects: The Object Repository 4
Building the user interface: The Form Designer, Component palette, and Object Inspector 6
Viewing and editing code: The Code Editor And Code Explorer 7
Programming with Kylix 10
Creating a project 10
Building the user interface 11
Writing code 13
Compiling and debugging projects 14
Deploying programs 15
Internationalizing applications 15

Introduction

Borland® 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, 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.

This white paper explains how to start Kylix and gives you a quick tour of the main parts and tools of the desktop, or integrated desktop environment (IDE) and it gives you an overview of software development with Kylix. This includes creating a project, working with forms, writing code, and compiling, debugging, deploying, and internationalizing programs.

Kylix™

white paper

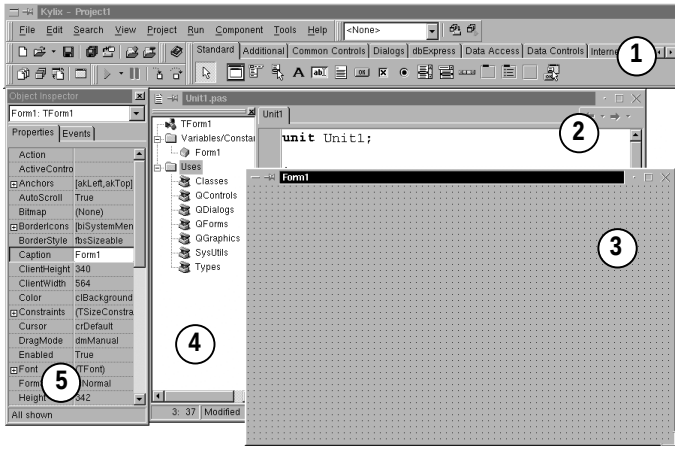
Starting Kylix

You can start Kylix in the following ways:

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

Getting started: The IDE

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



1. Palette of ready-made components to use in your applications. 2. Code Editor for viewing and editing code. 3. The Form Designer contains a blank form on which to start designing the user interface for your application. An application can include many forms.
4. The Code Explorer shows you the classes, variables, and routines in your unit and lets you navigate quickly. The Code Explorer does not come with all editions of Kylix.
5. The Object Inspector is used to change objects' properties and select event handlers.

Kylix's development model is based on *two-way* tools. This means that you can move back and forth between visual design tools and text-based code editing. For example, after using the Form Designer to arrange buttons and other elements in a graphical interface, you can immediately view the form file that contains the textual description of your form. You can also manually edit any code generated by Kylix without losing access to the visual programming environment.

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

Controlling Kylix: The menu and toolbars

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

Main window in its default arrangement.

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

Standard toolbar

1 New
2 Open project
3 Save
4 Save all
5 Open project
6 Add file to project
7 Remove file from project

View toolbar

1 View unit
2 View form
3 Toggle form/unit
4 New form

Desktops toolbar

1 Name of saved desktop layout
2 Save current desktop
3 Set debug desktop

Debug toolbar

1 Run
2 List of projects you can run
3 Pause
4 Trace into
5 Step over

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.

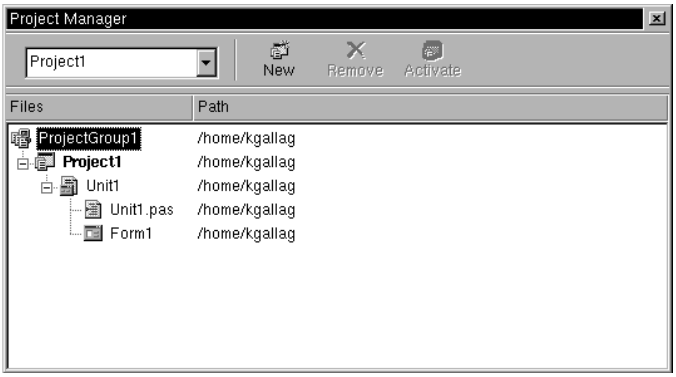
Many operations have keyboard shortcuts as well as toolbar buttons. When a keyboard shortcut is available, it is always shown next to the command on the drop-down menu.
You can right-click on many tools and icons to display a menu of commands appropriate to the object you are working with. These are called context menus.

The toolbars are also customizable. You can add commands you want to them or move them to different locations.

Managing projects: The Project Manager

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

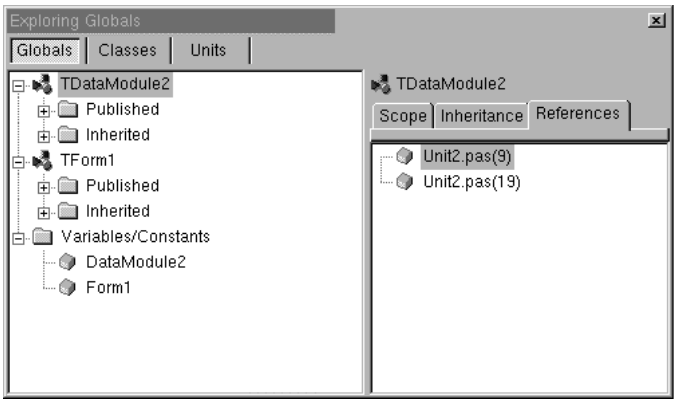
A project includes several files that make up the application or shared object you are going to develop. You can view and organize these files—such as form, unit, resource, object, and library files—in a project management tool called the Project Manager. To display the Project Manager, choose View | Project Manager.



You can use the Project Manager to combine and display information on related projects into a single *project group*. By organizing related projects into a group, such as multiple executables, you can compile them at the same time.

Browsing project structure and elements: The Project Browser

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



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

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

Classes displays classes in a hierarchical diagram.

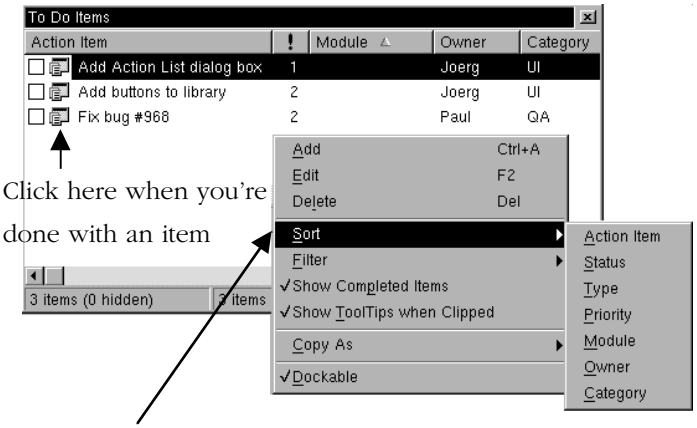
Units displays units, identifiers declared in each unit, and the other units that use and are used by each unit.

You can change the way the contents are grouped within the diagram by right-clicking in the Browser, choosing Properties, and, under Explorer categories, checking and unchecking the check boxes. If a category is checked, elements in that category are grouped under a single node. If a category is unchecked, each element in that category is displayed independently on the diagram's trunk.

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

Creating to-do lists

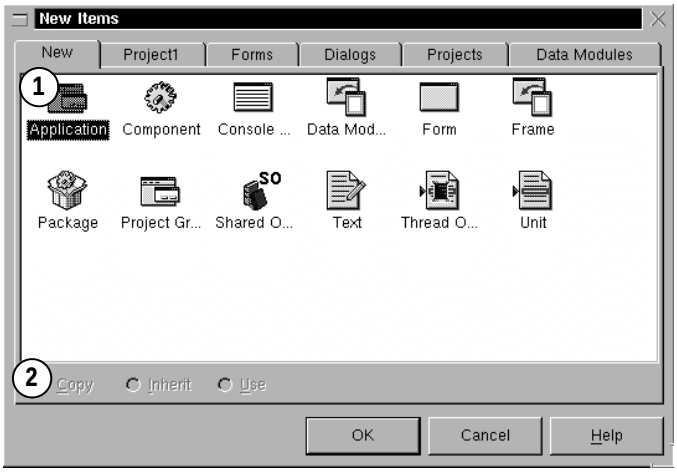
To-do lists record items that need to be completed for a project. You can add project-wide items to a list by adding them directly to the list, or you can add specific items directly in the source code. Choose the View | To-Do list to add or view information associated with a project.



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

Adding items to your projects: The Object Repository

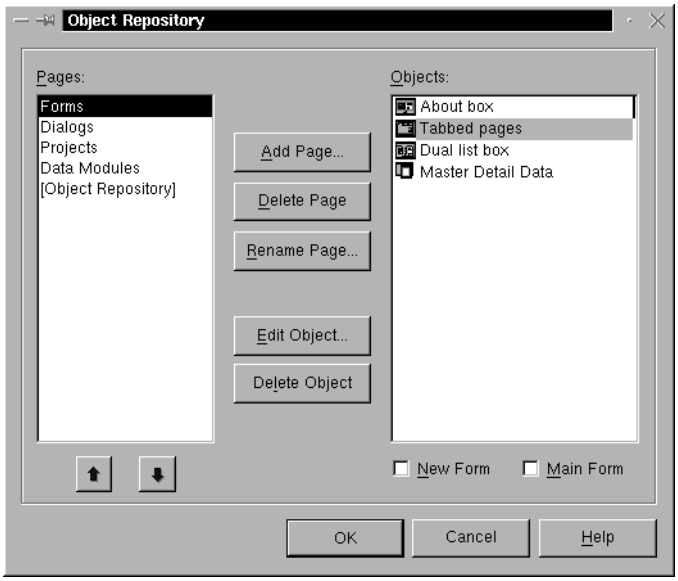
The Object Repository contains forms, dialog boxes, data modules, wizards, shared object files, sample applications, and other items that can simplify development. Choose File | New to display the New Items dialog box when you begin a project. The New Items dialog box is the same as the Object Repository. Check the Repository to see if it contains an object that resembles one you want to create.



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

2 When you're creating an item based on one from the Object Repository, you can copy, inherit, or use the item: *Copy* (the default) creates a copy of the item in your project. *Inherit* means changes to the object in the Repository are inherited by the one in your project. *Use* means changes to the object in your project are inherited by the object in the Repository.

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



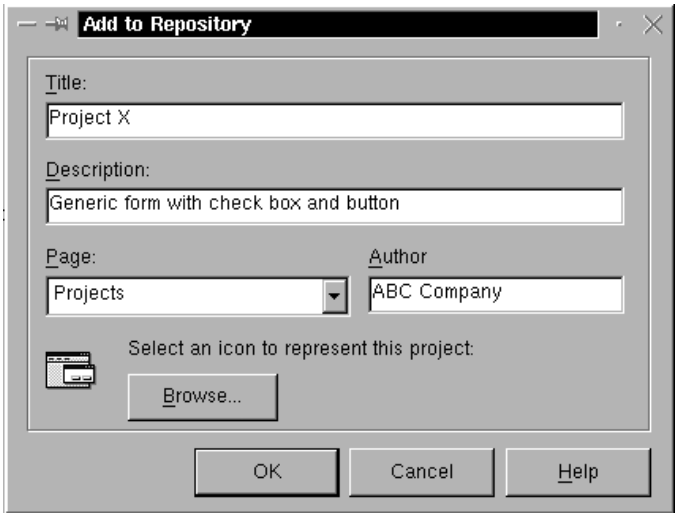
You can add, remove, or rename tabbed pages from the Object Repository.

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

Adding templates to the Object Repository

You can add your own objects to the Object Repository as *templates* to reuse and share with other developers over a network. Reusing objects lets you build families of applications with common user interfaces and functionality that reduces development time and improves quality.

For example, to add a project to the Repository as a template, first save the project and choose Project|Add To Repository. Complete the Add to Repository dialog box.



Enter a title, description, and author. In the Page list box, choose Projects so that your project will appear on the Repository's Projects tabbed page.

The next time you open the New Items dialog box, your project template will appear on the Projects page (or the page to which you had saved it).

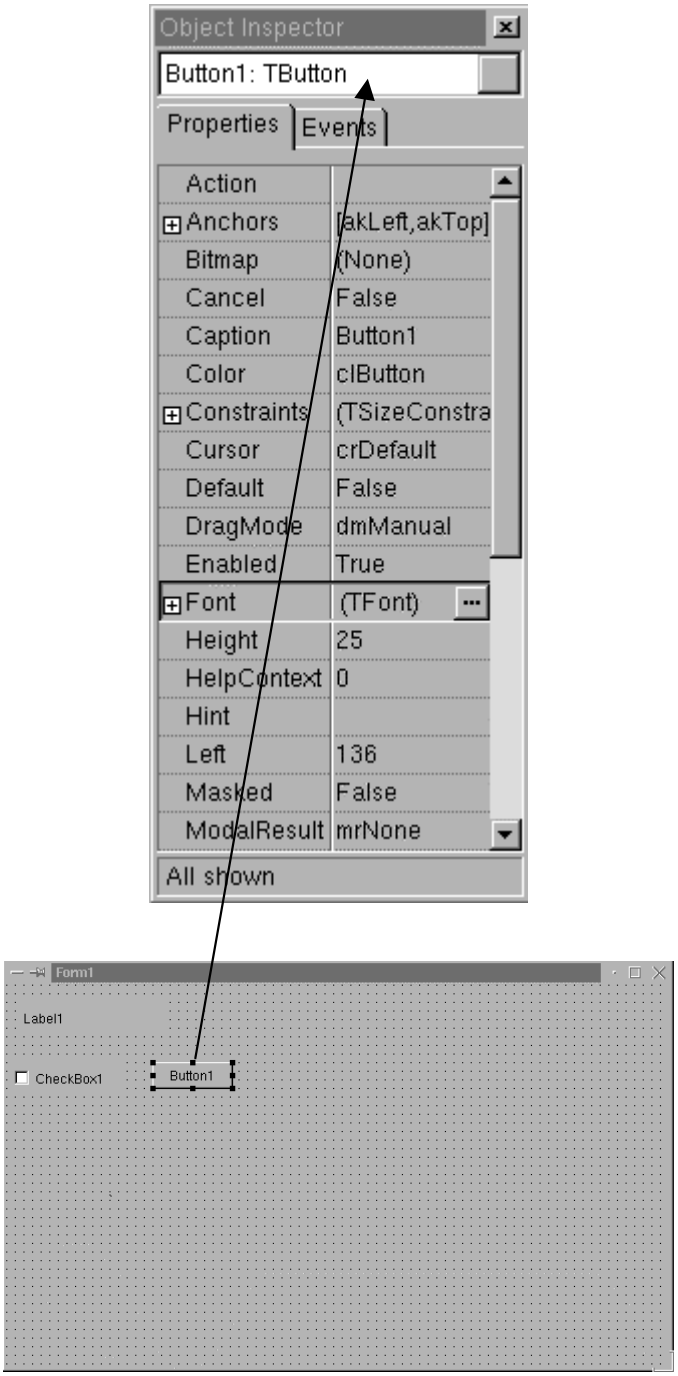
Building the user interface: The Form Designer, Component palette, and Object Inspector™

The Component palette includes tabbed pages with groups of icons representing visual or nonvisual CLX™ components you use to design your application interface. The pages divide the components into various functional groups. For example, the Dialogs page includes common dialog boxes to use for file operations such as opening and saving files.

Component palette pages, grouped by function

Components

Each component has specific attributes—properties, events, and methods—that enable you to control your application. Use the Form Designer to arrange components the way they should look on your user interface. For the components you place on the form, use the Object Inspector to set design-time properties, create event handlers, and filter visible properties and events, making the connection between your application's visual appearance and the code that makes your application run.

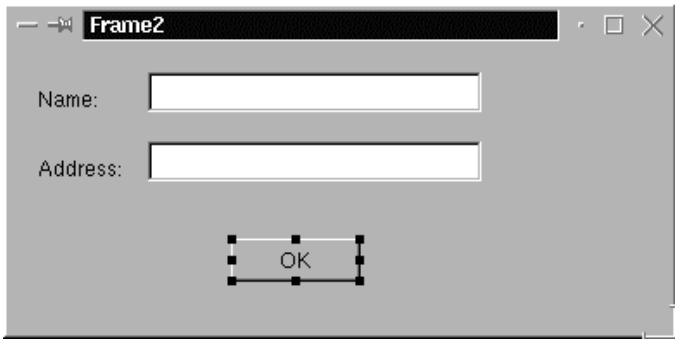


After you place components on a form, the Object Inspector dynamically changes the set of properties it displays, based on the component selected.

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.

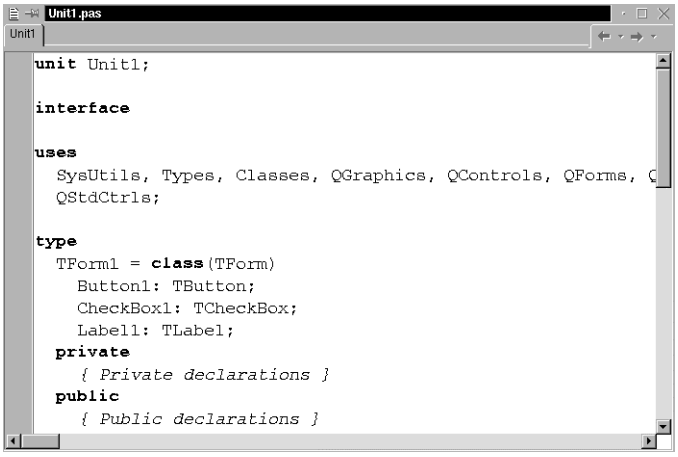


You can add whatever visual or nonvisual components you need to the frame. A new unit is automatically added to the Code Editor.

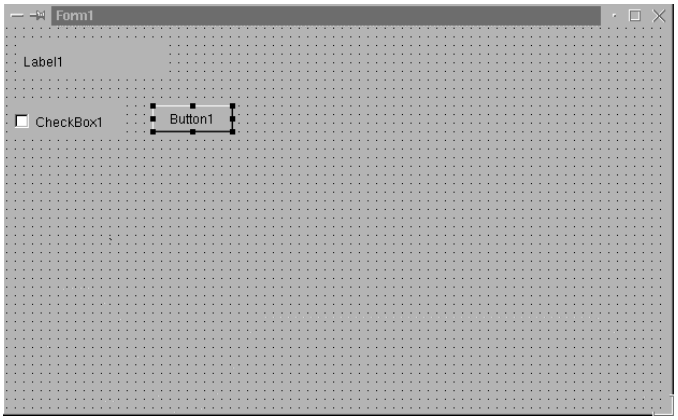
Viewing and editing code: The Code Editor and Code Explorer

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

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



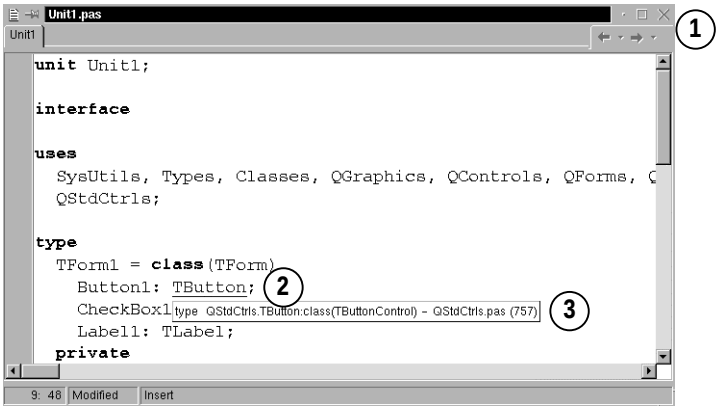
Generated code.



Components added to the form are reflected in the code.

Browsing with the Code Editor

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



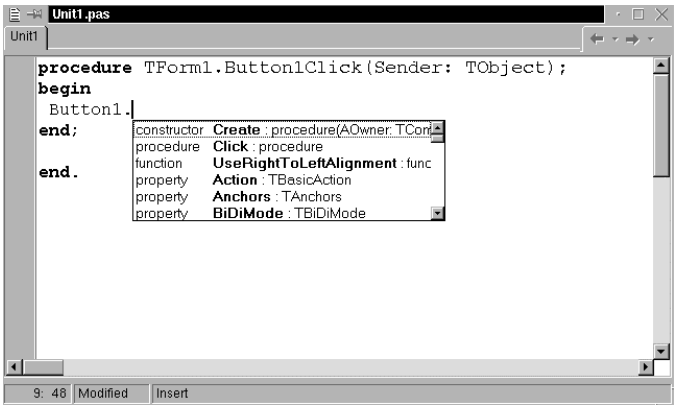
1 Use the Editor like a Web browser.

2 Press *Ctrl* and click to jump to the definition of the identifier.

3 The Tooltip Symbol Insight displays declaration information for any identifier when you pass the mouse over it.

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

Table 2.1	CodeInsight™ tools
Tool	How it works
Code Completion	Type a class name followed by a dot (.) to display a list of properties, methods, and events appropriate to the class. Type the beginning of an assignment statement and press Ctrl+space to display a list of valid values for the variable. Type a procedure, function, or method name to bring up a list of arguments.
Code Parameters	Type a method name and an open parenthesis to display the syntax for the method's arguments.
Code Templates	Press Ctrl+J to see a list of common programming statements that you can insert into your code. You can create your own templates in addition to the ones supplied with Kylix.
Tooltip Expression Evaluation	While your program has paused during debugging, point to any variable to display its current value.
Tooltip Symbol Insight	While editing code, point to any identifier to display its declaration.



When you type the dot in Button1. Kylix displays a list of properties, methods, and events for the class. As you type, the list automatically filters to the selection that pertains to that class.

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

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

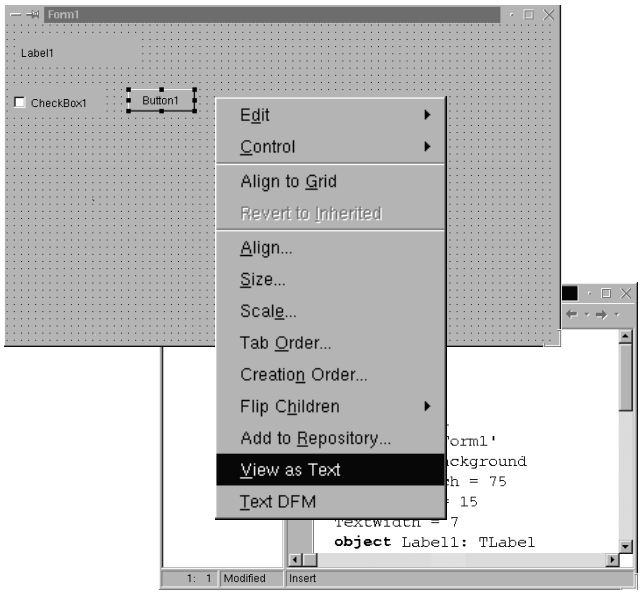
Class Completion

Class Completion generates skeleton code for classes. Place the cursor anywhere within a class declaration; then press Ctrl+Shift+C, or right-click and select Complete Class at Cursor. Kylix automatically adds private read and

write specifiers to the declarations for any properties that require them, then creates skeleton code for all class 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.

Viewing and editing form code

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

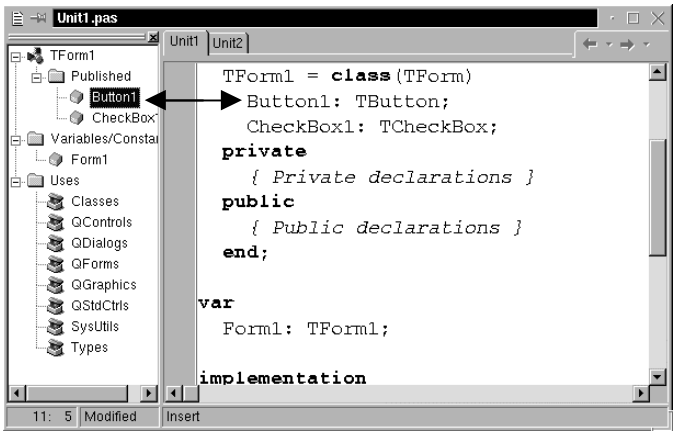


Use View As Text to view a text description of the form's attributes in the Code Editor.

You can save form files in either text (the default) or binary format. Use the Environment Options dialog box to designate which format to use for newly created forms.

Viewing code with the Code Explorer

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



Select an item in the Code Explorer and the cursor moves to that item's implementation in the Code Editor.
Move the cursor in the Code Editor and the highlight moves to the appropriate item in the Code Explorer.
To search for a class, property, method, variable, or routine, just type the first letter of its name.

Programming with Kylix

The following sections provide an overview of software development with Kylix, including creating a project, working with forms, writing code, and compiling, debugging, deploying, and internationalizing programs.

Creating a project

A project is a collection of files that are either created at design time or generated when you compile the project source code. When you first start Kylix, a new project opens. It automatically generates a project file (Project1.dpr), unit file (Unit1.pas), and resource file (Unit1.xfm), among others.

If a project is already open but you want to open a new one, choose either File | New Application or File | New and double-click the Application icon. File | New opens the Object Repository, which provides additional forms, frames, and modules as well as predesigned templates such as dialog boxes to add to your project. To learn more about the Object Repository, see "Adding items to your projects: The Object Repository" on page 4.

When you start a project, you have to know what you want to develop, such as an application or shared object.

Types of projects

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

Database applications

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

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

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

Web server applications

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

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

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

Shared object libraries

Shared object libraries are compiled modules containing routines that can be called by applications and by other shared objects. A shared object library contains code or resources typically used by more than one application.

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.

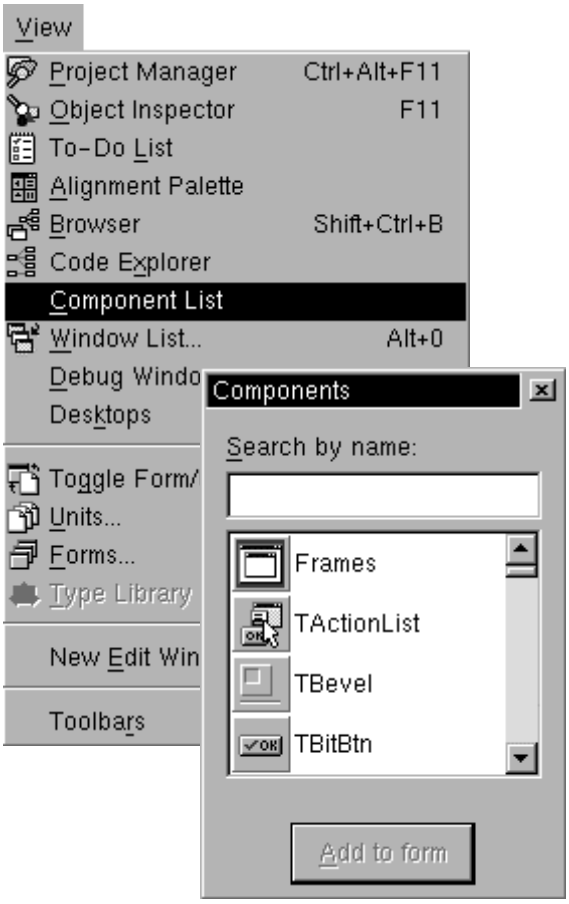
Building the user interface

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

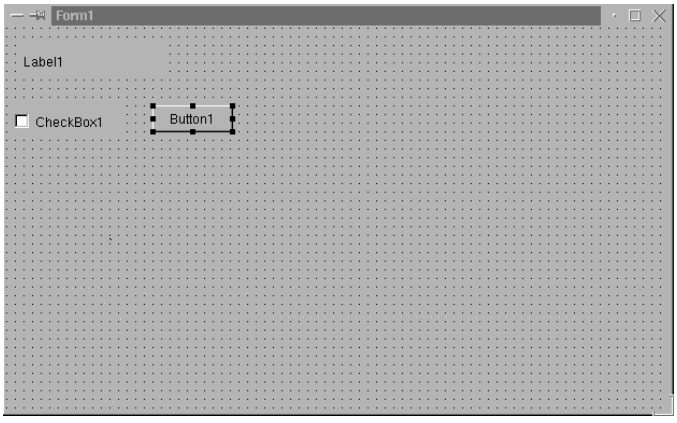
Placing components on a form

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

Click a component on the Component palette.



Or choose a component from an alphabetical list.

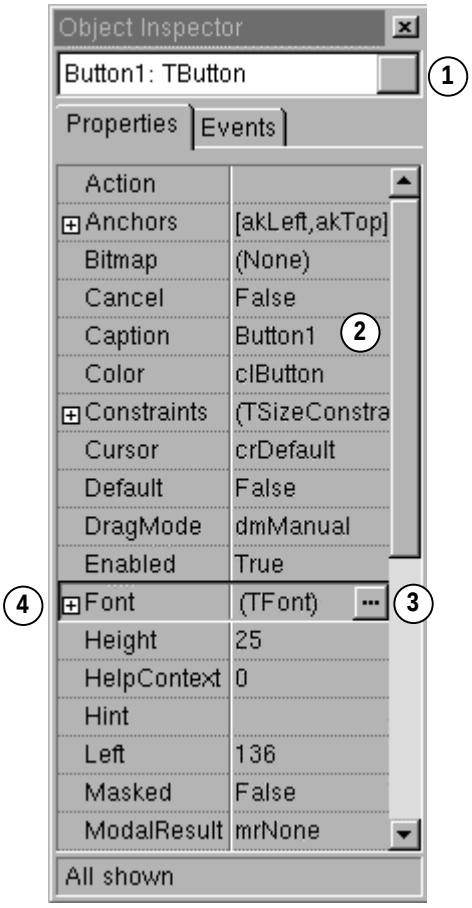


Then click where you want to place it on the form.

Setting component properties

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

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

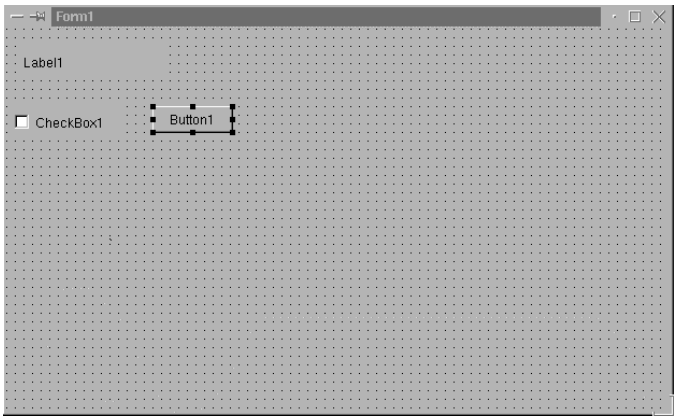


1 Or use this drop-down list to select an object. Here, Button1 is selected, and its properties are displayed.

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

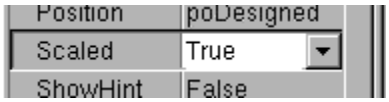
3 Click an ellipsis to open a dialog box where you can change the properties of a helper object.

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

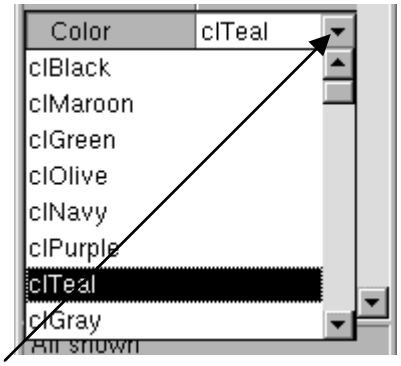


You can select a component, or object, on the form by clicking on it.

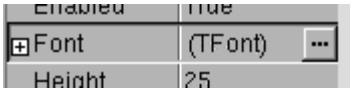
Many properties have simple values—such as names of colors, True or False, and integers. For Boolean properties, you can double-click the word to toggle between True and False. Some properties have associated property editors to set more complex values. When you click on such a property value, you'll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



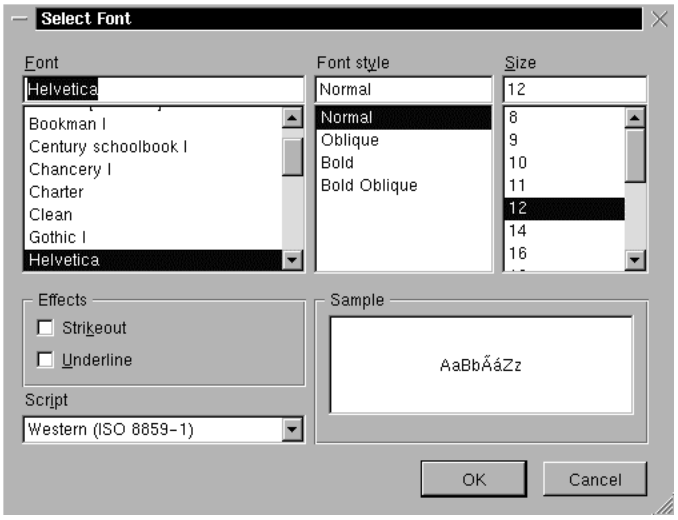
Double-click here to change the value from *True* to *False*.



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



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



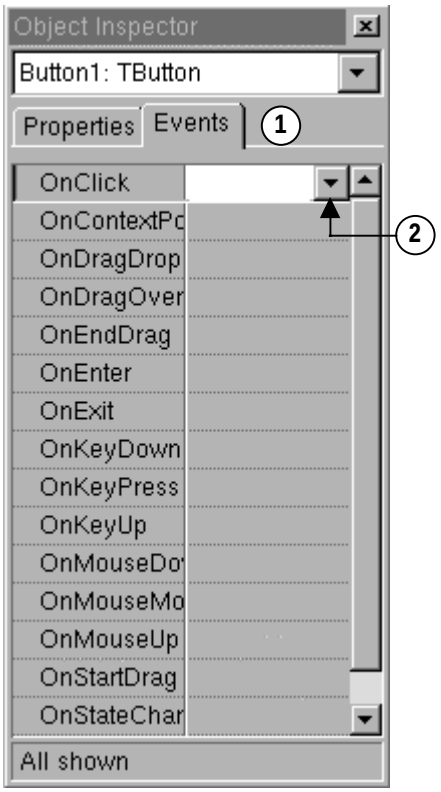
When more than one component is selected in the form, the Object Inspector displays all properties that are shared among the selected components.

Writing code

An integral part of any application is the code behind each component. While Kylix's RAD environment provides most of the building blocks for you, such as prepackaged visual and nonvisual components, you will usually need to write event handlers and perhaps some of your own classes. To help you with this task, you can choose from Kylix's CLX class library of nearly 750 objects. To view and edit your source code, see "Viewing and editing code: The Code Editor and Code Explorer" on page 7.

Writing event handlers

Your code may need to respond to events that might occur to a component at runtime. An event is a link between an occurrence in the system, such as clicking a button, and a piece of code that responds to that occurrence. The responding code is an event handler. This code modifies property values and calls methods. To view predefined event handlers for a component on your form, select the component and, on the Object Inspector, click the Events tab.



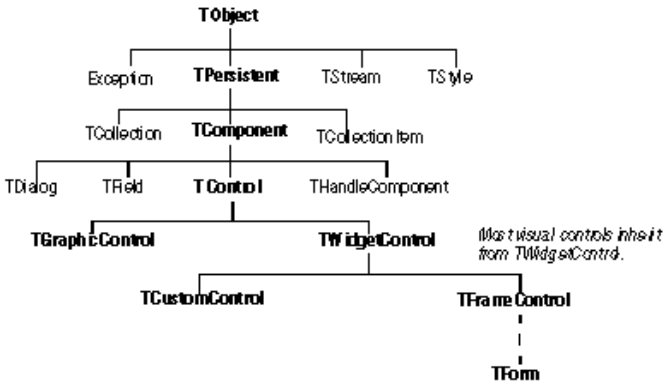
1 Here, Button 1 is selected and its type is displayed: TButton. Click the Events tab in the Object Inspector to see the events that Button component can handle.

2 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.

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

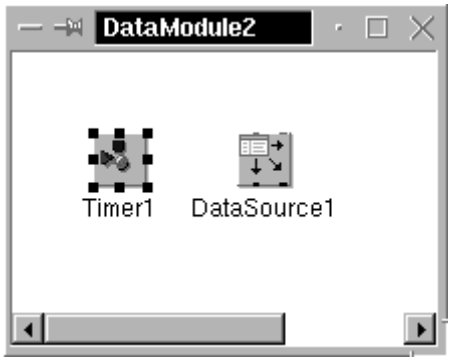
controls, buttons, and other user interface elements—as well as nonvisual controls like datasets and timers.
The diagram below shows some of the principal classes that make up CLX.



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.

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 you would to a form.



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

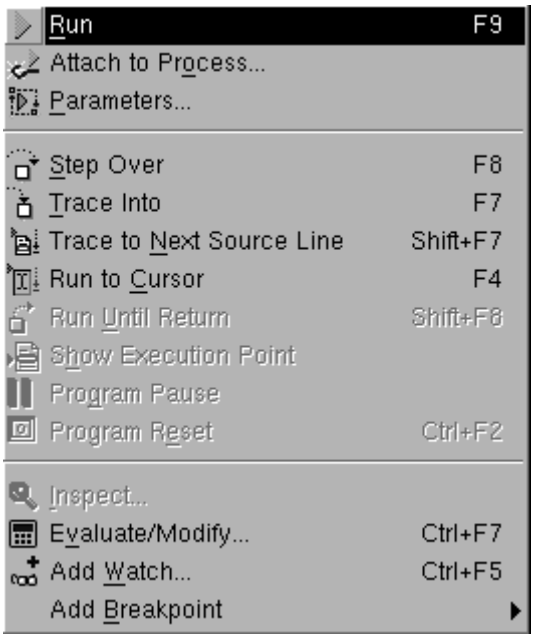
When you reopen an existing data module, Kylix displays its components.

Compiling and debugging projects

After you have written your code, you will need to compile and debug your project. With Kylix, you can either compile your project first and then separately debug it, or you can compile and debug in one step using the integrated debugger. To compile your program with debug information, choose Project | Options, click the Compiler page, and make sure Debug information is checked.
Kylix uses an integrated debugger so that you can control program execution, watch variables, and modify data values. You can step through your code line by line,

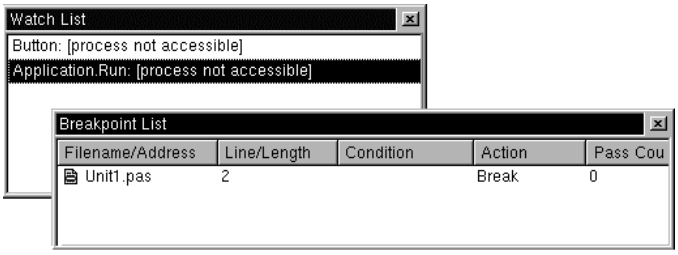
examining the state of the program at each breakpoint.
To use the integrated debugger, choose Tools | Debugger Options, click the General page, and make sure Integrated debugging is checked.
You can begin a debugging session in the IDE by choosing Run | Run or pressing F9.

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.



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.

Deploying programs

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

Internationalizing applications

Kylix offers several features for internationalizing and localizing applications for different locales. The IDE and CLX provide 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.

Borland

100 Enterprise Way
Scotts Valley, CA 95066-3249
www.borland.com | 831-431-1000