Tutorial: Creating a Text Editor using C++

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

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

K3–C++_TE–0702

Contents

Creating a text editor using the C++ IDE

Starting a new application 1
Setting property values 3
Adding components to the form 3
Adding support for a menu and a toolbar 6
  Adding actions to the action list 7
  Adding standard actions to the action list 9
  Adding images to the image list 11
Adding a menu 13
Clearing the text area 15
Adding a toolbar 16
Writing event handlers 17
  Writing the New command event handler 17
  Writing the Open command event handler 19
  Writing the Save command event handler 21
  Writing the Save As command event handler 22
  Writing the Exit command event handler 23
Creating an About box 24
Completing your application 26

Index

Creating a text editor using the C++ IDE

This tutorial guides you through the creation of a text editor complete with menus, a toolbar, and a status bar. You will use the C++ IDE to create the text editor.

This tutorial assumes you are familiar with Linux and have read the introduction to Kylix programming and the IDE in the *Quick Start*.

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 the C++ IDE.

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

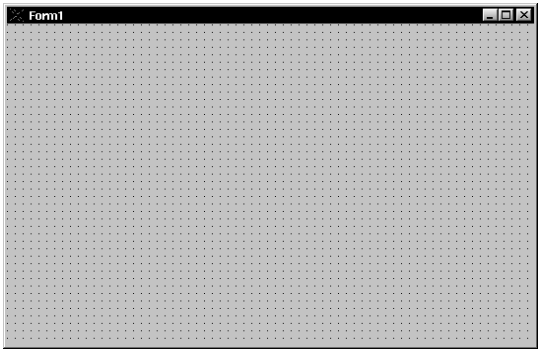
3 Choose File | Save All to save your files to disk. When the Save dialog box appears:

- Navigate to your TextEditor folder.
- Save Unit1 using the default name Unit1.cpp.
- Save the project using the name TextEditor.bpr. (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. Do not delete these files.

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.



The default form has Maximize and Minimize buttons, a Close button, and a Control menu.

If you run the form now by pressing *F9*, you'll see that these buttons all work.

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

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 and select Close from the menu;
- 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 (derived from *objects*) on the Component palette that make it easy to program menus, toolbars, dialog boxes, and many other visual and nonvisual program elements.

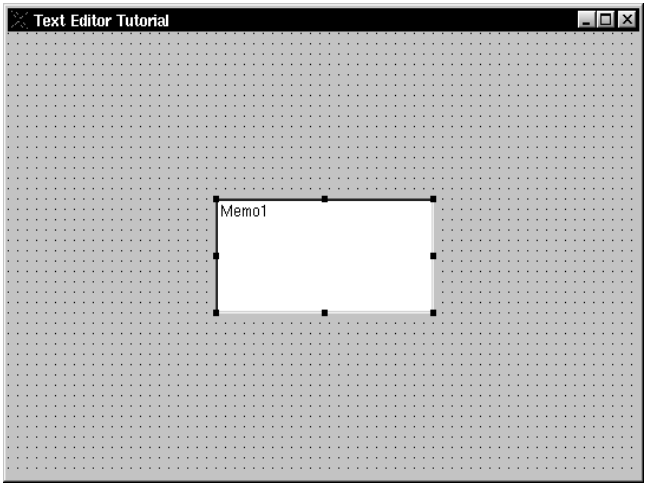
The text editor application requires an editing area, a status bar for displaying information such as the name of the file being edited, menus, and perhaps a toolbar with icons for easy access to commands. The beauty of designing the interface using Kylix is that you can experiment with different components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

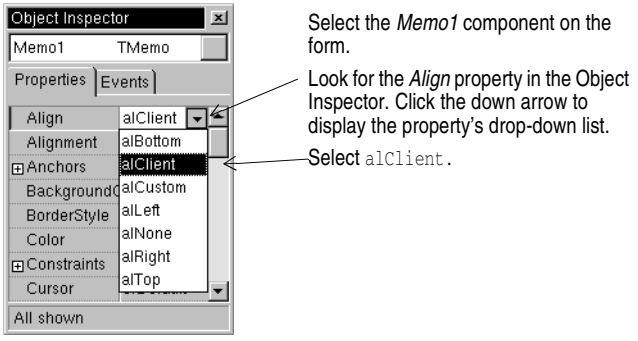
- Select the component on the palette and click on the form where you want to place the component; or
- Double-click it to place it in the middle of the form.



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.

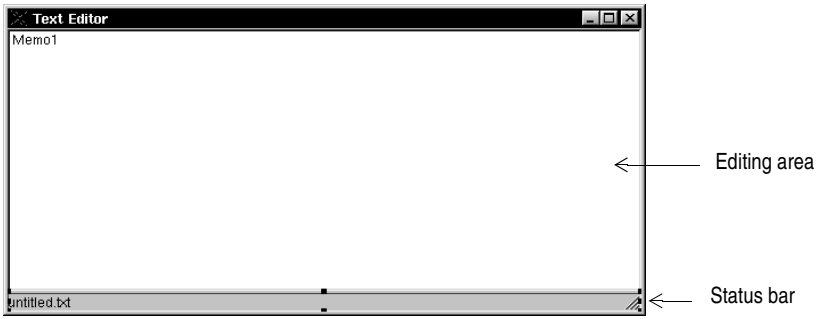


The *Memo* component now fills the entire form so you have a large text editing area. By choosing the *alClient* value for the *Align* property, the size of the *Memo* control will vary to fill whatever size window is displayed even if the form is resized.

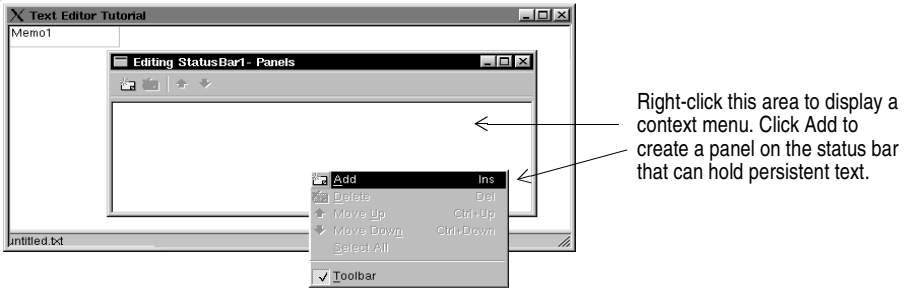
3 Double-click the *StatusBar* component on the Common Controls tab of the Component palette. This adds a status bar to the bottom of the form.

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`.

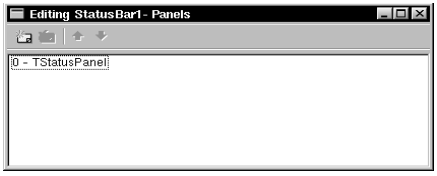


- Click the ellipse of the *Panels* property to open the Editing StatusBar1->Panels dialog box. You can stretch the dialog box to make it larger.
- Right-click the dialog box and click Add to add a panel to the status bar.



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 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.

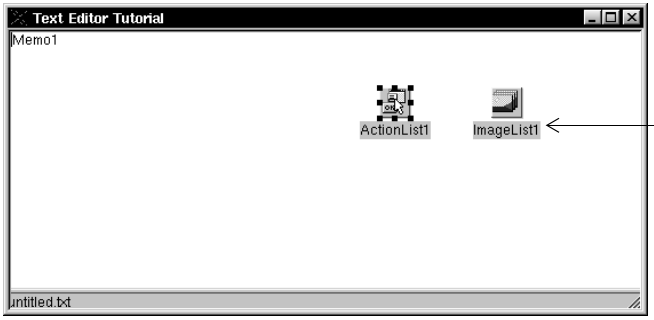
You can also centralize images to use for your toolbar and menus in an image list.

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| Designer 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

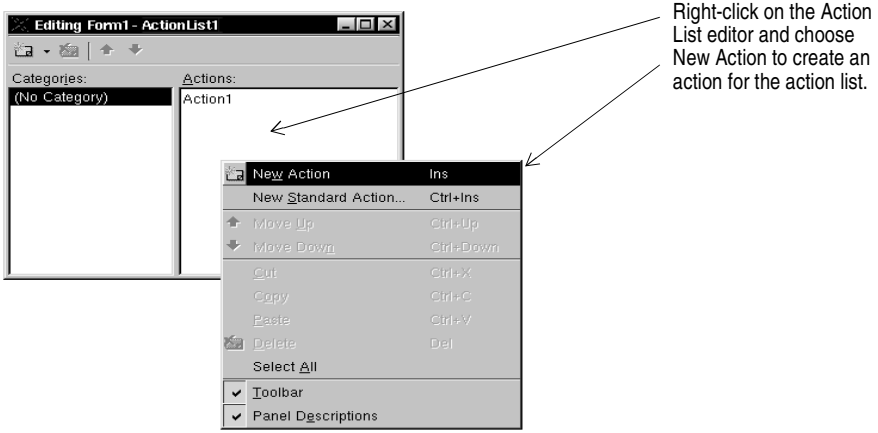
Next we'll add the actions to the action list.

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.

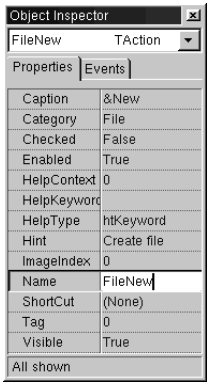


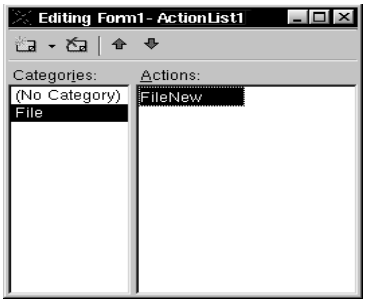
3. In the Object Inspector, set the following properties for the action:
   - After *Caption*, type `&New`. Note that typing an ampersand before one of the letters makes that letter a shortcut to accessing the command.
   - After *Category*, type `File` (this organizes the File commands in one place).
   - 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

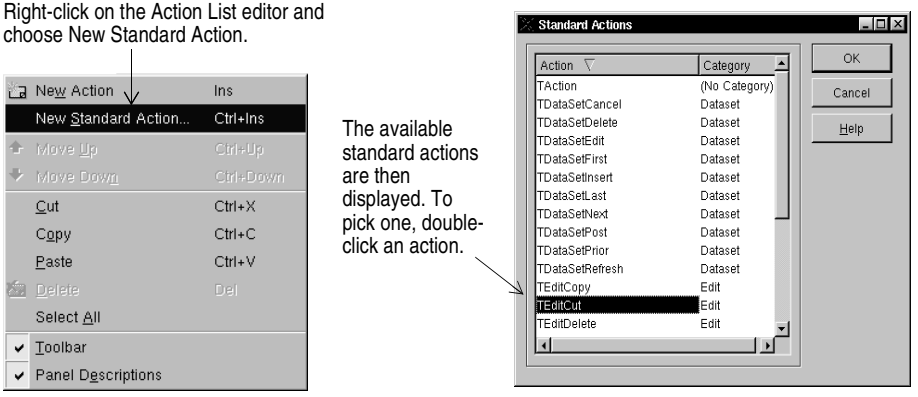
Kylix provides several standard actions that are often used when developing applications. Next we'll add the standard actions (cut, copy, and paste) to the action list.

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.



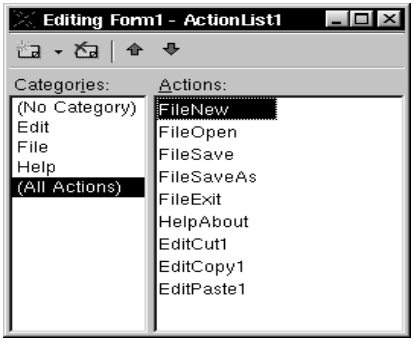
- Double-click TEditCut. The action is created in the Editing Form1->ActionList1 dialog box along with a new category called Edit. Select EditCut1.
- In the Object Inspector, set the *ImageIndex* property to `4`.

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`.

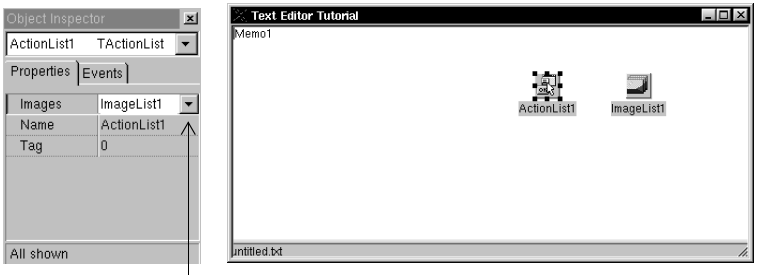
3 Right-click on the Action List editor and choose New Standard Action.
   - Double-click TEditPaste.
   - In the Object Inspector, set the *ImageIndex* property to `6`.

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:

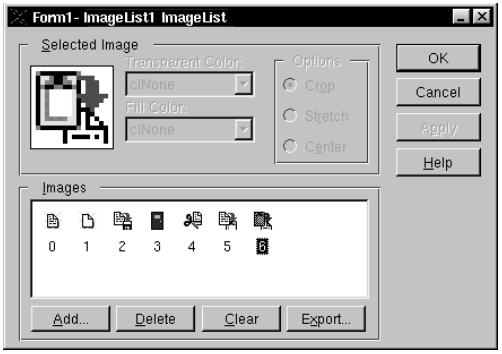
Table 2 Command Images

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/kylix directory, look in /usr/local/kylix/images/buttons.
3. Double-click fileopen.bmp.
4. When a message asks if you want to separate the bitmap into two separate ones, click Yes each time. Each of the icons includes an active and a grayed out version of the image. You'll see both images. Delete the grayed out (second) image.
   - Click Add. Double-click filenew.bmp. Delete the grayed out image.
   - Click Add. Double-click fileopen.bmp. Delete the grayed out image.
   - Click Add. Double-click filesave.bmp. Delete the grayed out image.
   - Click Add. Double-click doorshut.bmp. Delete the grayed out image.
   - Click Add. Double-click cut.bmp. Delete the grayed out image.
   - Click Add. Double-click copy.bmp. Delete the grayed out image.

- Click Add. Double-click paste.bmp. Delete the grayed out image.

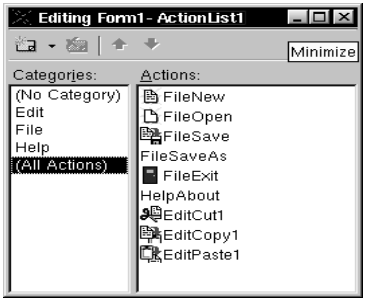


5 Click OK to close the Image List editor.

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 two of the commands because they 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.

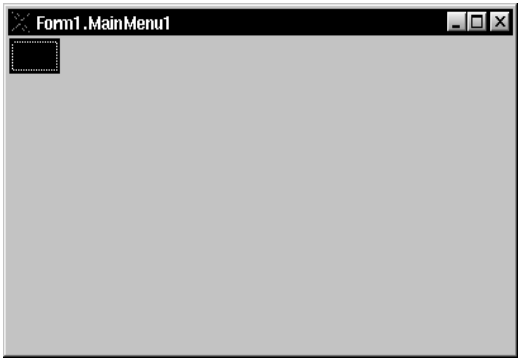
Adding a menu

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.

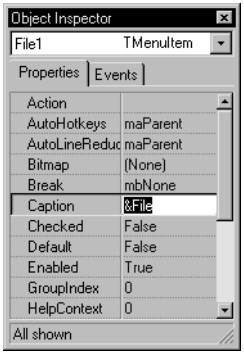
1 From the Standard tab of the Component palette, drop a *MainMenu* component onto the form. It doesn't matter where you place it.

2 Set the main menu's *Images* property to `ImageList1` so you can add the bitmaps to the menu items.

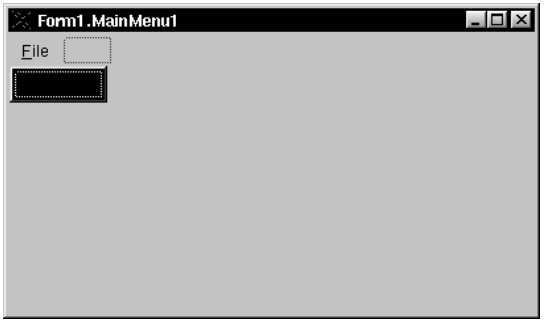
3 Double-click the main menu component to display the Menu Designer.



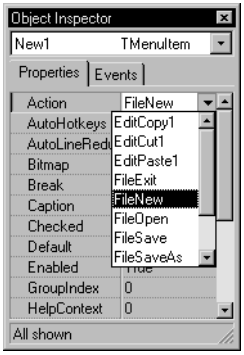
4 In the Object Inspector, after *Caption*, type `&File`, and press ***Enter*** to set the first top-level menu item.



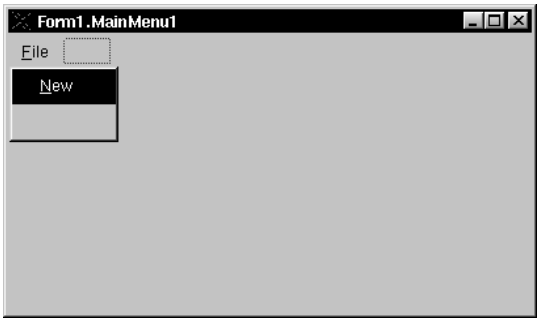
When you type *&File* and focus on the Menu Designer, the top-level File command appears ready for you to add the first menu item.



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`.



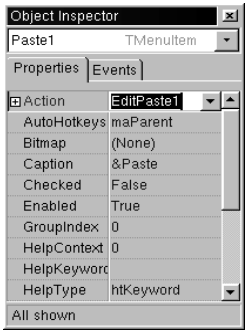
When you select FileNew from the Action property list, the New command appears with the correct Caption and ImageIndex.

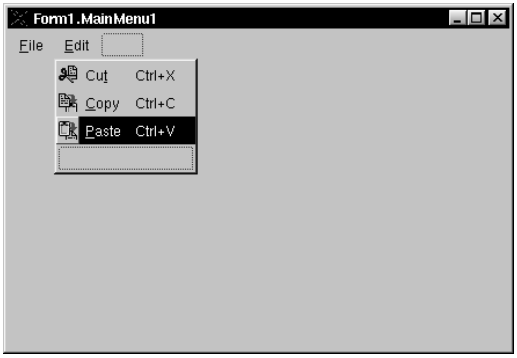


- Select the item under New and set its *Action* property to `FileOpen`.
- Select the item under Open and set its *Action* property to `FileSave`.
- Select the item under Save and set its *Action* property to `FileSaveAs`.
- Select the item under Save As, type a hyphen after its *Caption* property, and 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:

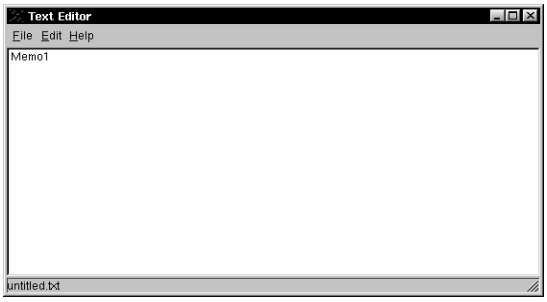
- Select the item to the right of the Edit command, set its *Caption* property to `&Help`, and press *Enter*.
- Select the item under Help and set its *Action* property to `HelpAbout`.

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 the menu items with which we associated the 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, choose Window | Form1 to redisplay it.

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 Memo1 text 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 those actions to a toolbar.

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

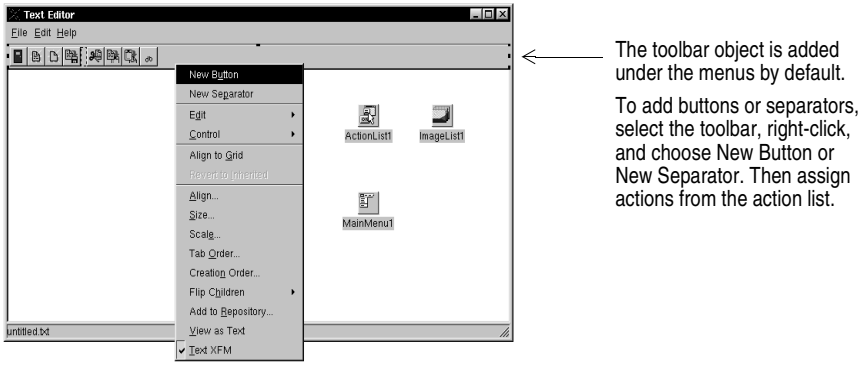
A blank toolbar is added under the main menu. With the toolbar still selected, change the following properties in the Object Inspector:

- Set the *Indent* property to 4. (This indents the icons 4 pixels from the left of the toolbar.)
- Set the *Images* property to `ImageList1`.
- Set the *ShowHint* property 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.



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.

   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 the Kylix RAD environment. In this section, you'll write functions called *event handlers* that respond to user input while the application is running. You'll connect the event handlers to the items on the menus and toolbar, so that when an item is selected your application executes the code in the handler.

For the nonstandard actions, you must create an event handler. For the standard actions, such as the File | Exit and Edit | Paste commands, the events are included in the code. However, for some of the standard actions, such as the File | Save As command, you will want to write your own event handler to customize the command.

Because all of 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.

Writing the New command event handler

To write an event handler for the New command:

1 Choose View | Units and select Unit1 to display the code associated with Form1.

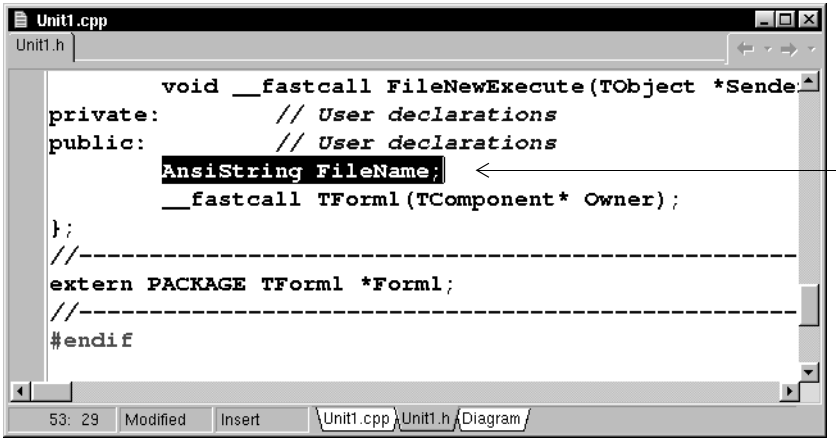
2 You need to declare a file name that will be used in the event handler. Add a custom property for the file name to make it globally accessible. Open the Unit1.h file by right-clicking in the Unit1.cpp file in the Code Editor and choosing Open Source/Header File. In the header file, locate the public declarations section for the class TForm1 and on the line after:

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

   type:

```
AnsiString FileName;
```

Your screen should look like this:



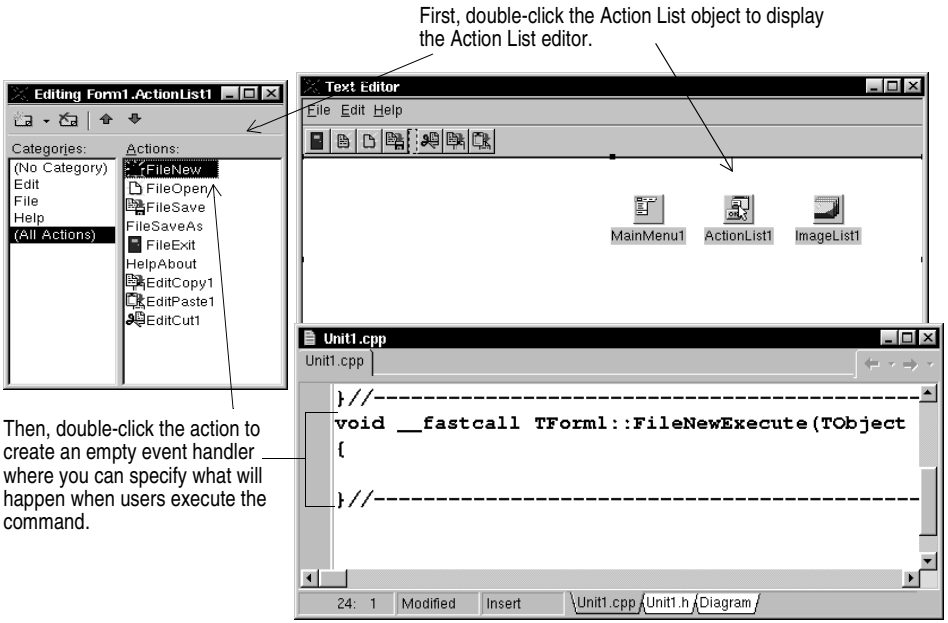
3 Press *F12* to go back to the main form.

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

4 Double-click the ActionList icon on the form to display the Action List editor.

5 In the Action List editor, select the File category and then double-click the FileNew action.

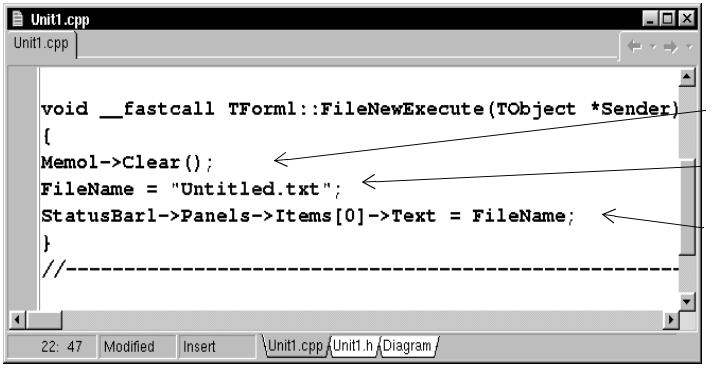
The Code editor opens with the cursor inside the event handler.



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

```
Memo1->Clear();
FileName = "Untitled.txt";
StatusBar1->Panels->Items[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.

Writing the Open command event handler

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:

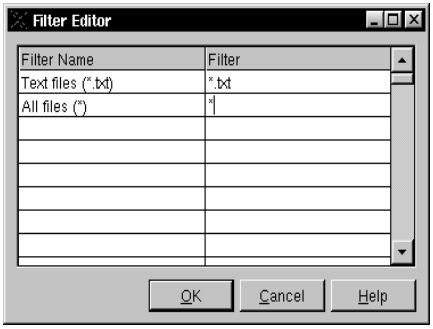
1 Locate the main form by choosing Window | Form1.

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 Filter Editor. In the first row of the Filter Name column, type `Text files (*.txt)`. In the Filter column,

type `*.txt`. In the second row of the Filter Name column, type `All files (*)` and in the Filter column, type `*`. Click OK.



Use the Filter Editor to define filters for the *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.

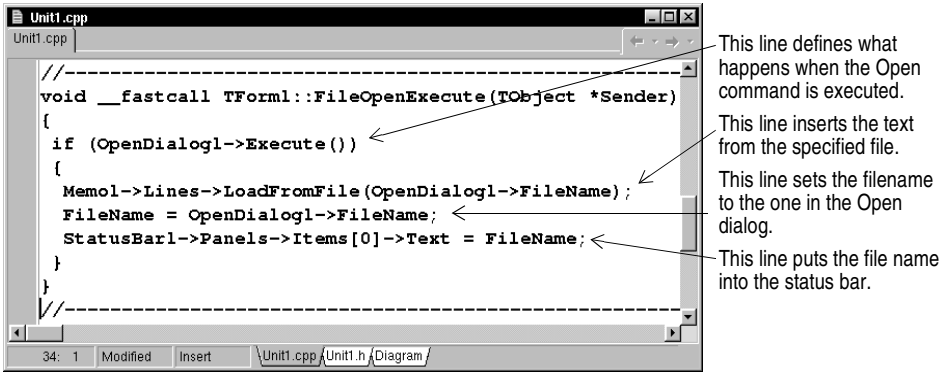
5 In the Action List editor, double-click the FileOpen action.

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

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

```
if (OpenDialog1->Execute())
   {
     Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
     FileName = OpenDialog1->FileName;
     StatusBar1->Panels->Items[0]->Text = FileName;
   }
```

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.

Writing the Save command event handler

To write an event handler for the Save command:

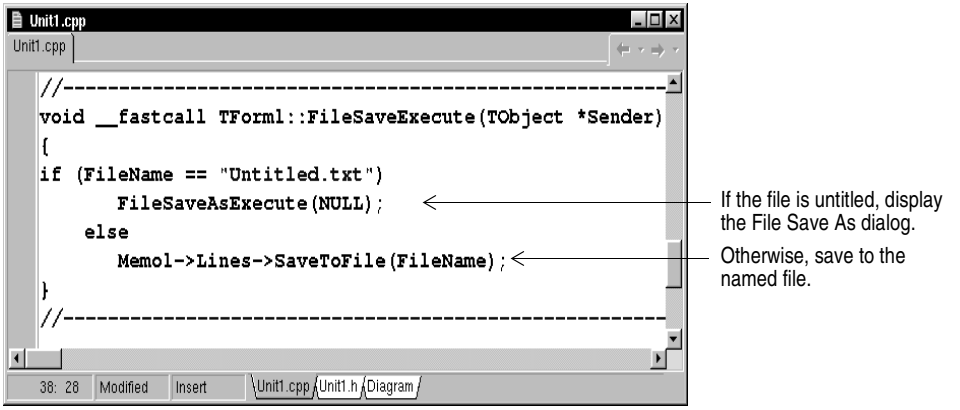
1. The Action List editor should still be displayed. If it's not, double-click the ActionList icon on the form.
2. On the Action List editor, double-click the FileSave action.

   The Code editor opens with the cursor inside the event handler.
3. Right where the cursor is positioned in the Code editor (between { and }), type the following lines:

```
if (FileName == "Untitled.txt")
      FileSaveAsExecute(NULL);
  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, it saves the file using its current name. The SaveAs dialog box is defined in the event handler for the Save As command on page -22. FileSaveAsExecute is the automatically generated name for the Save As command.

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



That's it for the File | Save command.

Writing the Save As command event handler

To write 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 Filter Editor. In the first row under the Filter Name column, type `Text files (*.txt)`. In the Filter column, type `*.txt`. In the second row under the Filter Name column, type `All files (*)` and in the Filter column, type `*`. Click OK.
- Set *Title* to `Save As`.

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

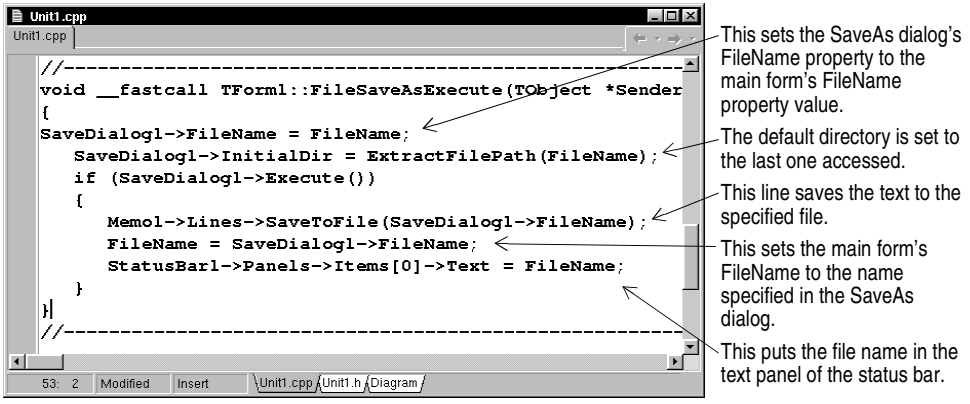
3 In the Action List editor, double-click the FileSaveAs action.

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

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

```
SaveDialog1->FileName = FileName;
   SaveDialog1->InitialDir = ExtractFilePath(FileName);
   if (SaveDialog1->Execute())
   {
      Memo1->Lines->SaveToFile(SaveDialog1->FileName);
      FileName = SaveDialog1->FileName;
      StatusBar1->Panels->Items[0]->Text = FileName;
   }
```

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



That's it for the File | SaveAs command.

Writing the Exit command event handler

To write 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.

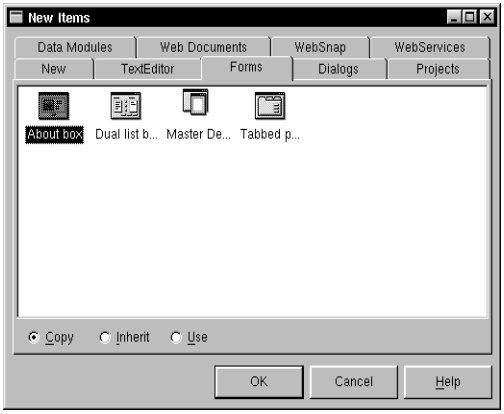
Creating an About box

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:

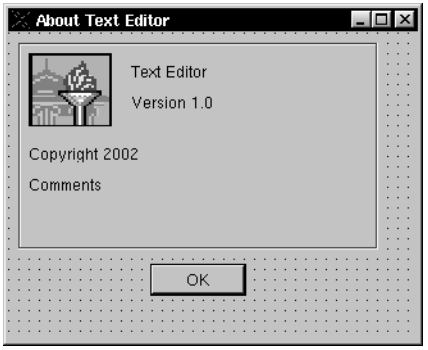
1. Choose File | New | Other to display the New Items dialog box and click the Forms tab.
2. On the Forms tab, double-click About Box.



A predesigned form for the About box appears.

3. Select the grid itself (click the grid portion) and in the Object Inspector, click the Properties tab and change the Caption property to `About Text Editor`.
4. Select the following *TLabel* items on the About box and in the Object Inspector, change their *Caption* properties:
   - Change Product Name to `Text Editor`.
   - Change Version to `Version 1.0`.

- Change Copyright to `Copyright 2002`.

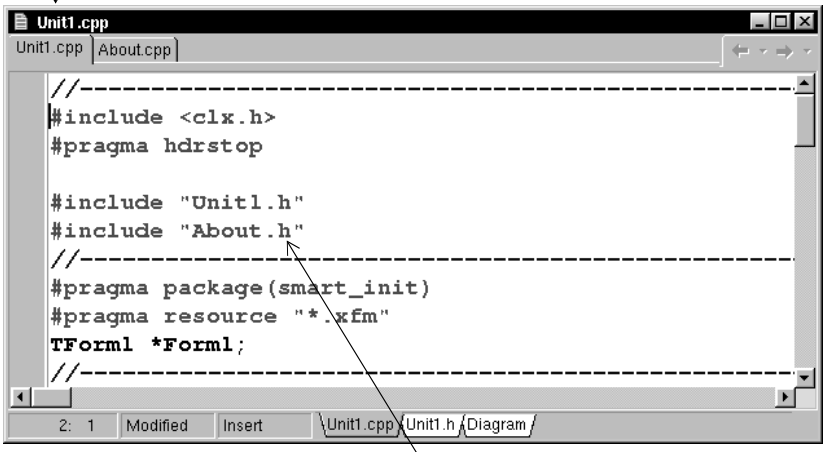


5 Save the About box form by choosing File | Save As and saving it as About.cpp.

6 In the Code editor, you should have these files displayed: Unit1.cpp, Unit1.h, and About.cpp. Click the Unit1.cpp tab to display Unit1.cpp.

7 Add an include statement for the About unit to Unit1. Choose File | Include Unit Hdr and then select About and click OK. Notice that #include About.h has been added to the top of the .cpp file.

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



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

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:

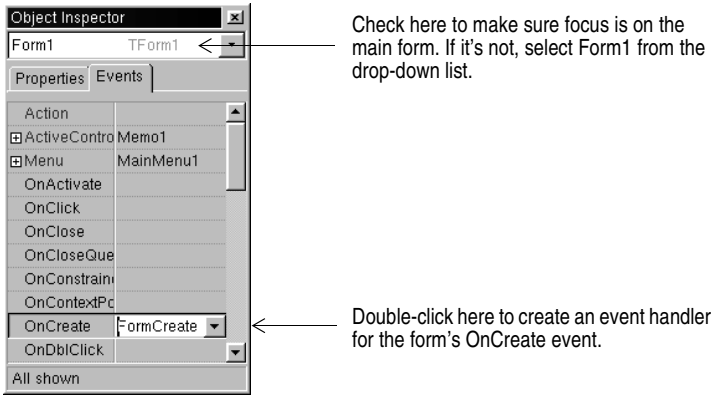
```
AboutBox->ShowModal();
```

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

Completing your application

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

1. Locate the main form (press *F12* to quickly find it).
2. Check that focus is on the form itself, not any of its components. The top list box on the Object Inspector should say Form1 TForm1. (If it doesn't, select Form1 from the drop-down list.)



3. In the Events tab, double-click OnCreate and choose FormCreate from the drop-down list to create an event handler that describes what happens when the form is created (that is, when you open the application).
4. Right where the cursor is positioned in the Code editor, type the following lines:

```
Application->HelpFile = ExtractFilePath(Application->ExeName) + "TextEditor.hlp";
FileName = "Untitled.txt";
StatusBar1->Panels->Items[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.

Congratulations! You're done!

Index

A

About box, adding 24
actions, adding to an application 7, 9
adding
  actions to an application 7
  an About box 24
  components to a form 3, 13
  images to an application 11
  menus to an application 13
  objects to a form 4
  standard actions to an application 9
  toolbars to an application 16
applications, compiling and debugging 15

B

bitmaps, adding to an application 11

C

code, writing 17 to 23
compiling programs 15
components
  adding to a form 3
  setting properties 3

D

design-time view 2

E

error messages 23, 26
event handlers, creating 17 to 23
events 17

F

files
  form 1
  project 1
  saving 2
  types 1
  unit 1
form files, defined 1
forms
  adding components to 3
  closing 2
  main 2

H

header files 1
Help tooltips 3

I

images, adding to an application 11

M

menus, adding to an application 13
messages, error 23, 26

N

New Items dialog box, using 24

O

Object Inspector, using 3
objects
  adding to a form 4
  defined 3

P

programs, compiling and debugging 15
project files 2
projects
  default files 1
  saving 2
properties, setting 3, 7

R

running applications 15

S

saving projects 2
setting properties 3, 7
source code, files 1
standard actions, adding to an application 9

T

toolbars, adding to an application 16
tooltips, viewing 3

U

unit files 1
unit header files 1
user interface, designing 3

X

.xfm files 1