Chapter

1

# 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. It includes a simple help file accessible from the application.

**Note** This tutorial is for all versions of Delphi 5.

## Starting a new application

Before beginning a new application, create a folder to hold the source files:

1. Create a folder called TextEditor in the Projects directory off the main Delphi directory.
2. Create a new project.

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

   When you open a new project, Delphi 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.dfm*: 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.dfm*) files. If you create a second form, a second unit (*Unit2.pas*) and form (*Unit2.dfm*) file are automatically created.
3. Choose File | Save All to save your files to disk. When the Save dialog 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 with an exe extension.)

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

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

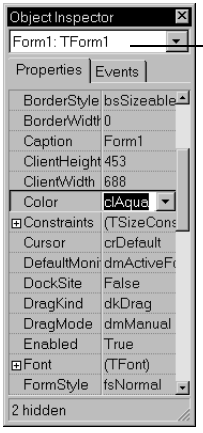
When you open a new project, Delphi displays the project's main form, named *Form1* by default. You'll create the user interface and other parts of your application by placing components on this form.


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.

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.




The drop-down list at the top of the Object Inspector shows the currently selected object. In this case, the object is *Form1* and its type is *TForm1*.

When an object is selected, the Object Inspector shows its properties.

# Setting property values

When you use the Object Inspector to set properties, Delphi 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:

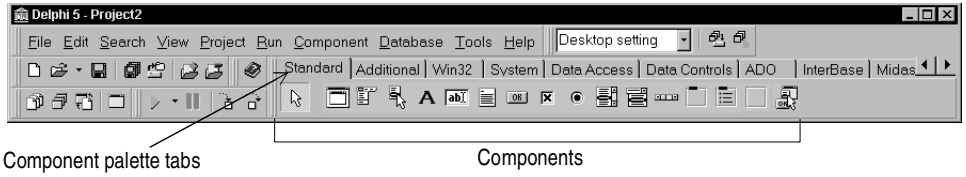
- Find the form's *Caption* property in the Object Inspector and type "Text Editor Tutorial" replacing the default caption "Form1." Notice that the caption in the heading of the form changes as you type.

# Adding objects to the form

Before you start adding objects 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. 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 Delphi is that you can experiment with different components and see the results right away. This way, you can quickly prototype an application interface.

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

The Component palette represents VCL components using icons grouped onto tabbed pages. Add a component to a form by selecting the component on the palette, then clicking on the form where you want to place it. You can also double-click a component to place it in the middle of the form. To get help on the components, select the component (either in the Component palette or on the form) and press F1.

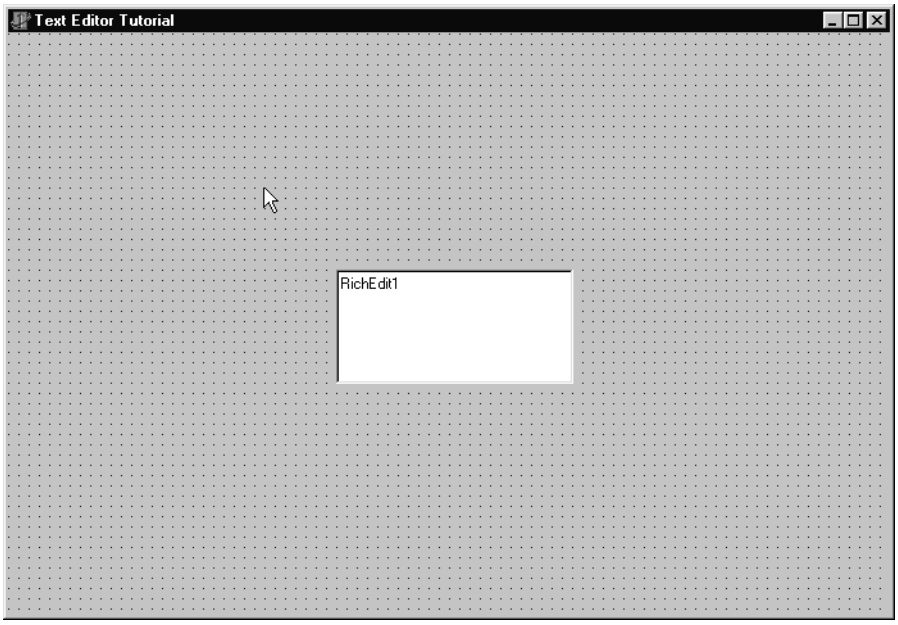



To start designing the text editor, add a *RichEdit* and a *StatusBar* component to the form:

1 To create a text area, drop a *RichEdit* component onto the form.

Click the Win32 page on the Component palette. To find the *RichEdit* component, point to an icon on the palette for a moment; Delphi displays a Help hint showing the name of the component.

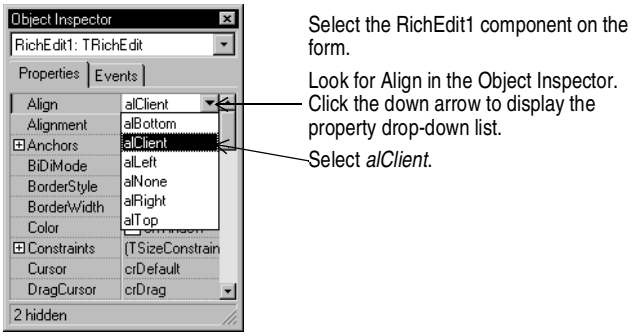

When you find the *RichEdit* component, double-click it to place it on the form.




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

**2** Set the *Align* property of *RichEdit1* to *alClient*.

To do this, click on *RichEdit1* to select it on the form, then choose the *Align* property in the Object Inspector. Select *alClient* from the drop-down list.




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

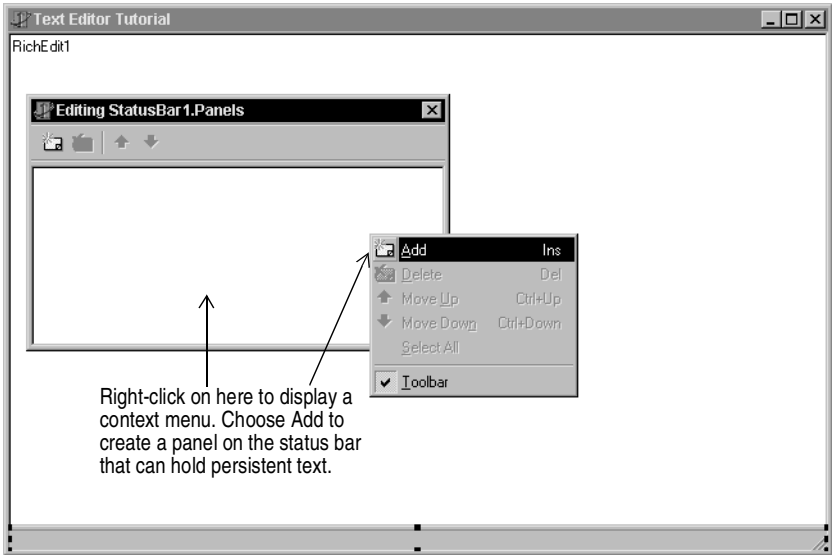
**3** Double-click the *StatusBar* component on the Win32 page of the Component palette. This adds a status bar to the bottom of the form.


Next we want to create a place to display the name of the file being edited. You can do this in two ways. The easiest way is to set the *SimplePanel* property of the *StatusBar1* object to True and assign any text that you want to display to the *SimpleText* property. This provides only one panel in the status bar. You can assign its value as follows:

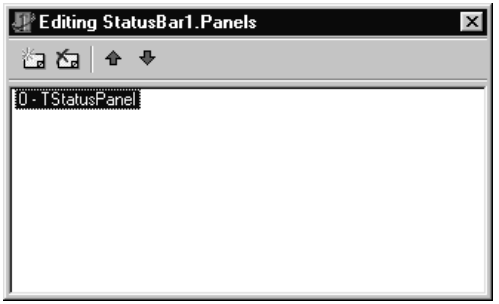
```
StatusBar1.SimpleText := 'untitled.txt';
```

However, many times you will want to include more than one panel in the status bar so you can include more than one piece of information. You can specify more than one panel by using the *Panels* property of the *TStatusBar* component as explained in the next few steps.

4 Double-click the status bar to display the Editing StatusBar1.Panels dialog box.




5 Right-click on the dialog box and choose Add to add the panel to the status bar.




The Panels property is a zero-based array that allows you to access each Panel that you create based on its unique index value (by default, it is 0 for this panel). Use the default property values for the panel. Click the **X** in the upper right corner to close the 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. Because some of the same commands will appear on the menu and the toolbar, you can centralize the code by creating an *action list*. Action lists help to centralize the code for the commands.

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

**Table 1.1** 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. |
| Contents | Help | Yes | Displays the Help contents screen from which you can access Help topics. |
| Index | Help | No | Displays the Help index screen. |
| 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 *ImageList*.

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

1 From the Standard page of the Component palette, drop an *ActionList* component onto the form. The *ActionList* component is nonvisual, so it doesn't matter where you put it on the form. It won't appear at runtime.

2 From the Win32 page, choose the *ImageList* component and drop it onto your form. It's also nonvisual so you can put it anywhere.

Your form should now resemble the following figure.

## Adding actions to the action list

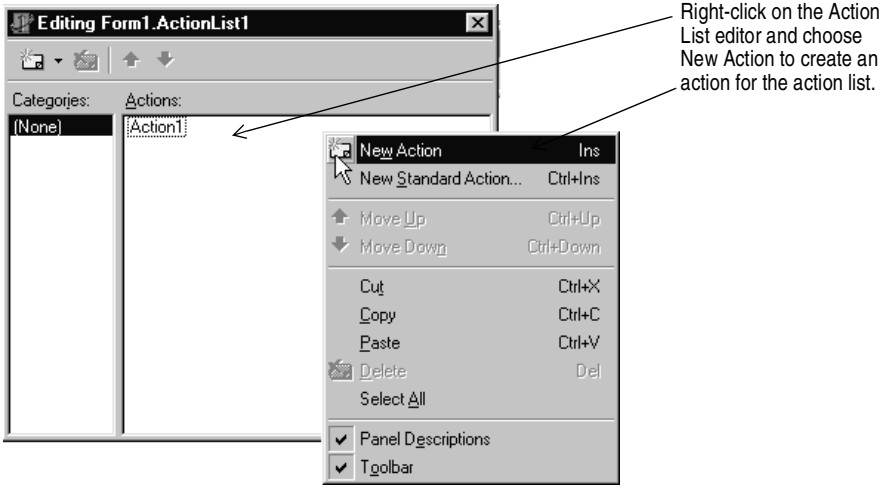
Next we'll add the actions to the action list.

**Tip** By convention, we'll name actions that are connected to menu items 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 is displayed. This is also called the Action List editor.

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



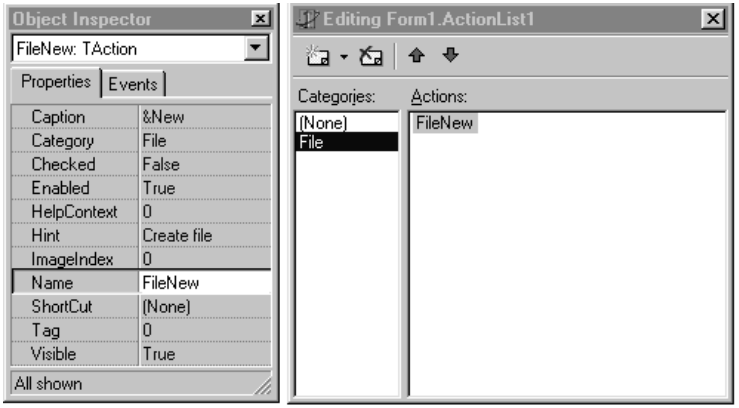

3. In the Object Inspector, set the following properties for the action:

   - 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 hint).
- 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).

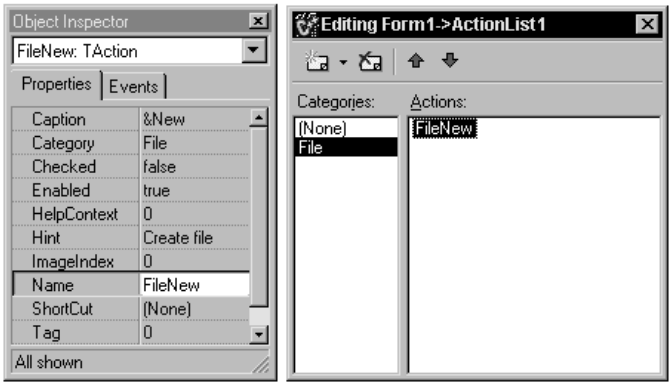
With the new action selected in the action list editor, change its properties in the Object Inspector.

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




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 hint, 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 the *Category* says File.
- After *Hint*, type `Open file`.
- After *ImageIndex*, type `1`.
- After *Name*, type `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 the *Category* says File.
   - After *Hint*, type `Save file`.
   - After *ImageIndex*, type `2`.
   - After *Name*, type 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 the *Category* says File.
   - After *Hint*, type `Save file as`.
   - No *ImageIndex* is needed. Leave the default value.
   - After *Name*, type 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 the *Category* says File.
   - After *Hint*, type `Exit application`.
   - After *ImageIndex*, type `3`.
   - After *Name*, type FileExit (for the File | Exit command).
12. Right-click on the Action List editor and choose New Action to create a customized Help | Contents command.
13. In the Object Inspector, set the following properties:
   - After *Caption*, type `&Contents`.
   - After *Category*, type `Help`.
   - After *Hint*, type `Display Help`.
   - After *ImageIndex*, type `7`.
   - After *Name*, type `HelpContents` (for the Help | Contents command).
14. Right-click on the Action List editor and choose New Action.
15. In the Object Inspector, set the following properties:
   - After *Caption*, type `&Index`.
   - Make sure the *Category* says Help.
   - After *Name*, type `HelpIndex` (for the Help | Index command).

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

17 In the Object Inspector, set the following properties:

- After *Caption*, type `&About`.
- Make sure the *Category* says Help.
- After *Name*, type `HelpAbout` (for the Help | About command).

Keep the Action List editor on the screen.

**Note** When you were adding actions to the action list, you might have noticed a standard Help | Contents command is provided. We added a custom Help | Contents command that will display the Help Contents tab at all times. The standard Help | Contents command brings up the last tabbed page that was displayed, either the Contents or the Index.

## Adding standard actions to the action list

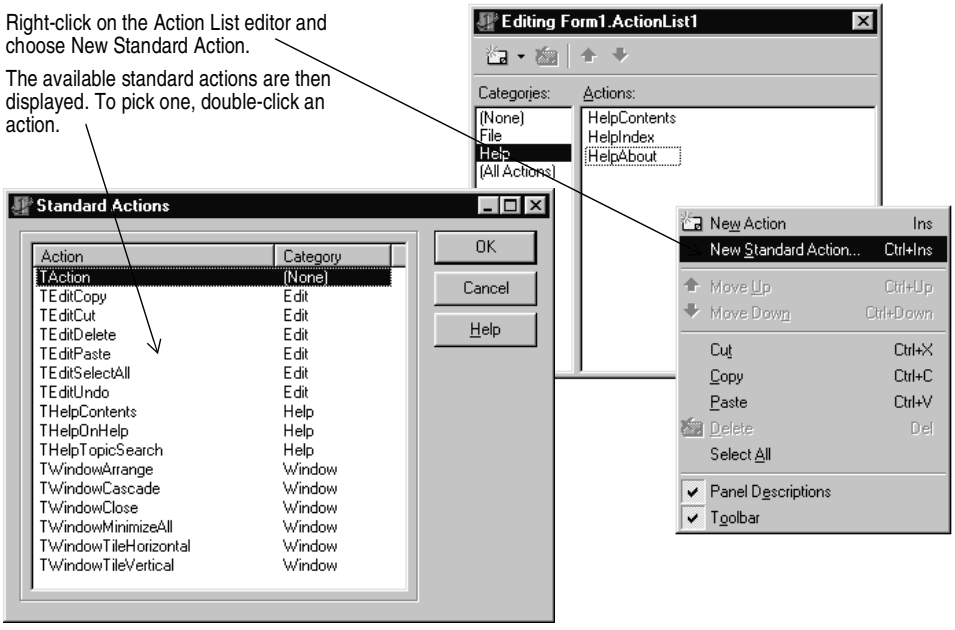
Delphi provides several standard actions that are often used when developing applications. Next we'll add the standard actions (cut, copy, and paste) to the action list.

**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 is displayed.



2 Double-click TEditCut. The action is created along with a new category called Edit. EditCut1 should be selected.

3 In the Object Inspector, set the following property for EditCut1:

- After *ImageIndex*, type `4`.

The other properties are set automatically.

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

5 Double-click TEditCopy.

6 In the Object Inspector, set the following properties:

- After *ImageIndex*, type `5`.

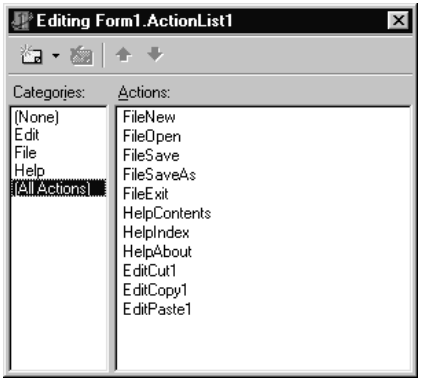
7 Right-click on the Action List editor and choose New Standard Action.

8 Double-click TEditPaste.

9 In the Object Inspector, set the following properties:

- After *ImageIndex*, type `6`.

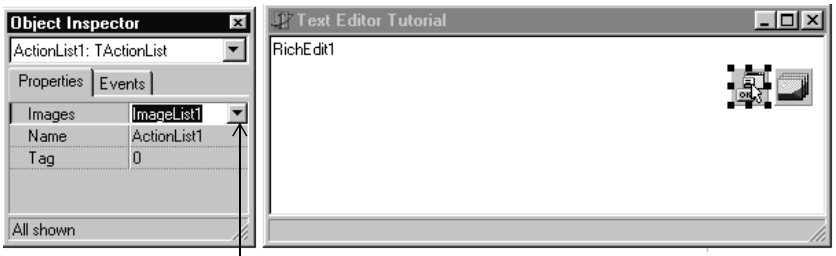
10 Now you've got all the actions that you'll need for the menus and toolbar. If you click on the category All Actions, you can see all the actions in the list:





11 Click on the **X** to close the Action List editor.

12 With the Action List still selected on the form, set its *Images* property to ImageList1.




Click on the Images property, then on the down arrow next to Images. ImageList1 is listed for you. Select it. This associates the images that we'll add to the image list with the actions in the action list.

## Adding images to the image list

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

| Command | Icon image name | ImageIndex property |
|---|---|---|
| File | Open | Fileopen.bmp | 0 |
| File | New | Filenew.bmp | 1 |
| File | Save | Filesave.bmp | 2 |
| File | Exit | Doorshut.bmp | 3 |
| Edit | Cut | Cut.bmp | 4 |
| Edit | Copy | Copy.bmp | 5 |
| Edit | Paste | Paste.bmp | 6 |
| Help | Contents | Help.bmp | 7 |

To add the images to the image list:

1 Double-click on the ImageList object on the form to display the Image List editor.

2 Click on the Add button and navigate to the Buttons directory provided with the product. The default location is C:\Program Files\Common Files\Borland Shared\Images\Buttons.

3 Select 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.

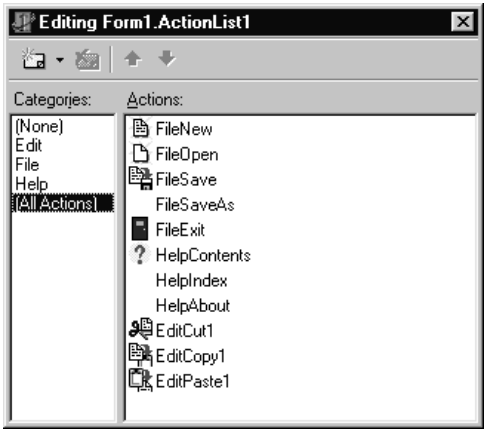
- Click Add and select filenew.bmp. Delete the grayed out image.

- Click Add and select filesave.bmp. Delete the grayed out image.
- Click Add and select doorshut.bmp. Delete the grayed out image.
- Click Add and select cut.bmp. Delete the grayed out image.
- Click Add and select copy.bmp. Delete the grayed out image.
- Click Add and select paste.bmp. Delete the grayed out image.
- Click Add and select help.bmp. Delete the grayed out image.

5 Click OK to close the Image List editor.

You've added 8 images to the image list and they're numbered 0-7 consistent with the ImageIndex numbers on each of the actions.

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 three 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

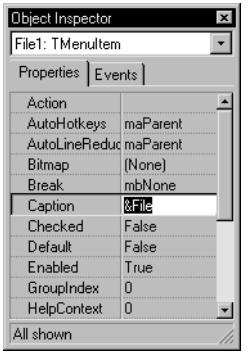
In this section, you'll add a main menu bar with three drop-down menus—File, Edit, and Help—and you'll add menu items to each one using the actions in the action list.

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

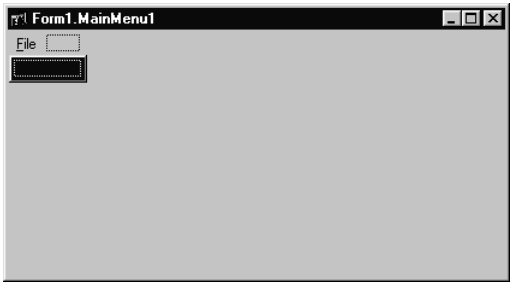
2 Set the main menu's *Images* property to *ImageList1*. This will allow you to add the images to the menu items.

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

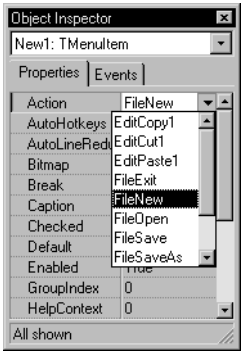

4 In the Object Inspector, type `&File` to set the *Caption* property of the first top-level menu item and press *Enter*.



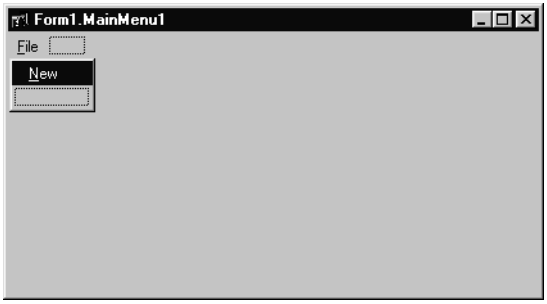

When you type *&File* and focus on the Menu Designer, the top-level File command appears ready for you to add the first menu item.




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.

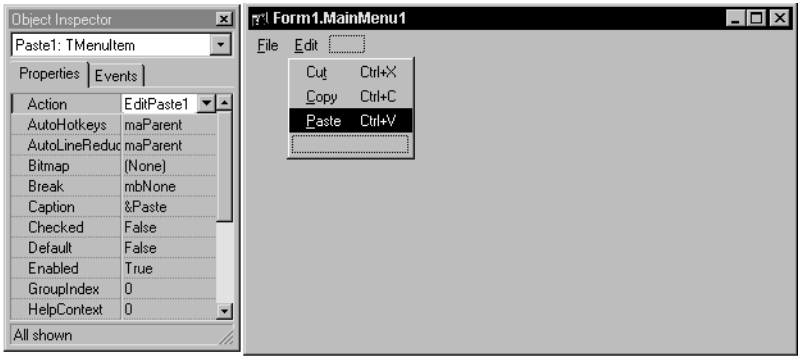



- Focus on the item under New and choose FileOpen from its Action property.
- Focus on the item under Open and choose FileSave from its Action property.
- Focus on the item under Save and choose FileSaveAs from its Action property.

- Focus on the item under Save As and type a hyphen as the Caption of the next item under the File menu and press *Enter* to create a separator bar on the menu.
- Focus on the item under the separator bar and choose FileExit from its Action property.

**6** Next create the Edit menu:

- Point to the item to the right of the File command and set its Caption property to &Edit and press Enter.
- Focus is now on the item under Edit; choose EditCut1 from its Action property.
- Select the item under Cut and choose EditCopy1 from its Action property.
- Select the item under Copy and choose EditPaste1 from its Action property.



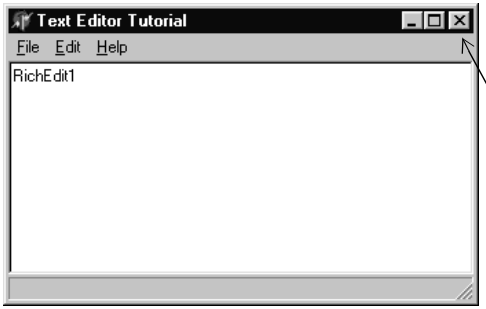

**7** Next create the Help menu:

- Point to the item to the right of the Edit command and type `&Help` as its caption.
- Focus on the Menu Designer to select the item under Help and choose HelpContents from its Action property.
- Select the item under Contents and choose HelpIndex from its Action property.
- Select the item under Index and type a hyphen its Caption and press *Enter* to create a separator bar on the Help menu.
- Select the item under the separator bar and choose HelpAbout from its Action property.

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

**9** Choose File | Save to save your project.

10 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.)




When you press *F9* to run your project, the application interface is displayed. The menus, text area, and status bar all appear on the form.

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

When you run your project, Delphi opens the program in a window like the one you designed on the form. The program is a full-fledged Windows application, complete with Minimize, Maximize, and Close buttons and a Control menu. The menus all work although most of the commands are grayed out. The images are displayed 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 Click the **X** in the upper right corner to close the application and return to the design-time view of the form.

## Clearing the text area (optional)

When you ran your program, the name of the *RichEdit* control appeared in the text area. You can remove that text using the Strings editor. This is optional because in a later step, the text will be removed when initializing the main form.

To clear the text area:

1 On the main form, click on the *RichEdit* component.

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

3 Select the text you want to remove in the String List editor, press the Delete key, 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 we've set up actions in an action list, we can add some of the same actions that were used on the menus onto a toolbar.

1 On the Win32 page of the Component palette, double-click the *ToolBar* to add it to the form.

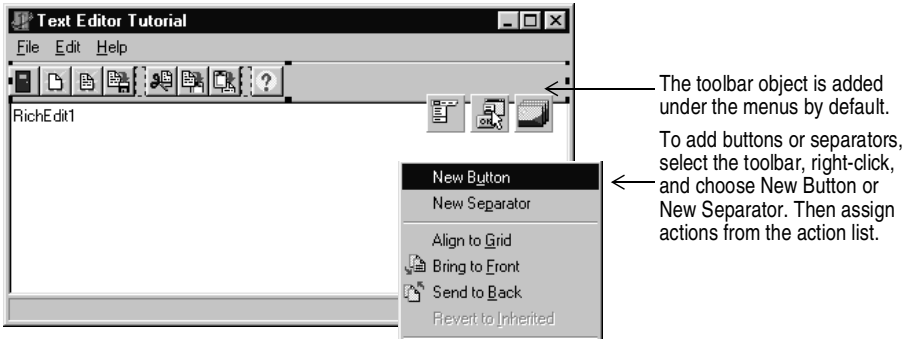
A blank toolbar is added under the main menu. With the toolbar still selected, change the following properties in the Object Inspector:

- 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 on 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.
- Right-click and choose New Separator.
- Right-click and choose New Button once again.

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

4 Assign actions to the second set of buttons.

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

5 Assign an action to the last button.

- Select the last button and set its *Action* to *HelpContents*.

6 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 work.

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

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

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

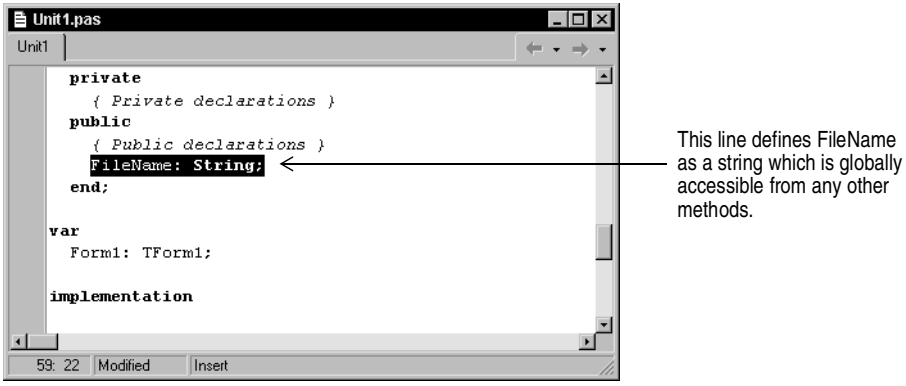
To create an event handler for the New command:

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

2 You need to declare a FileName that will be used in the event handler. Add a custom property for the file name to make it globally accessible. Earlier in the Unit1.pas file, locate the public declarations section for the class TForm1 and on the line after `{ Public declarations }`, type:

```
FileName: String;
```

Your screen should look like this:



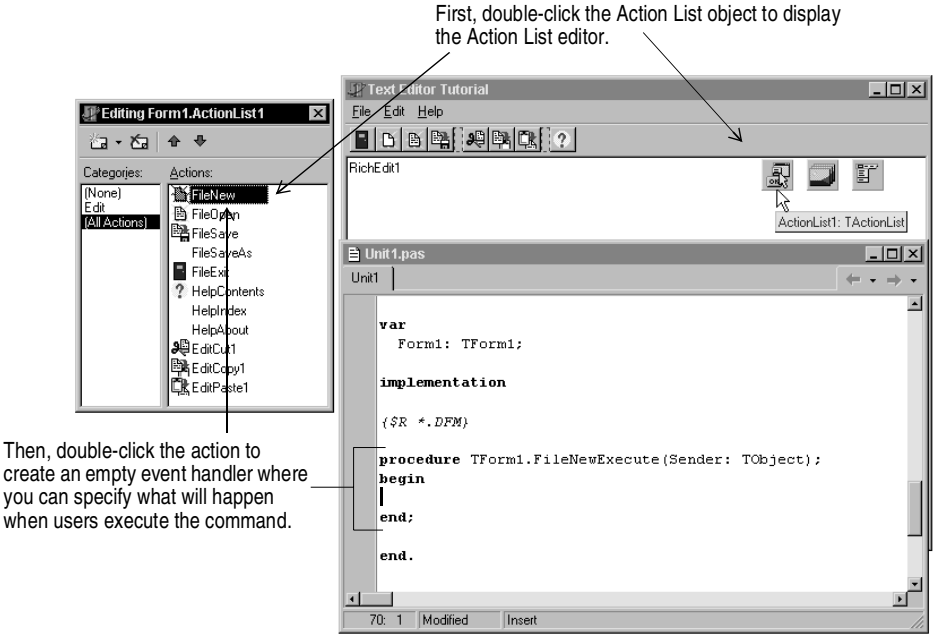

3 Press *F12* to go back to the main form.

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

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

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

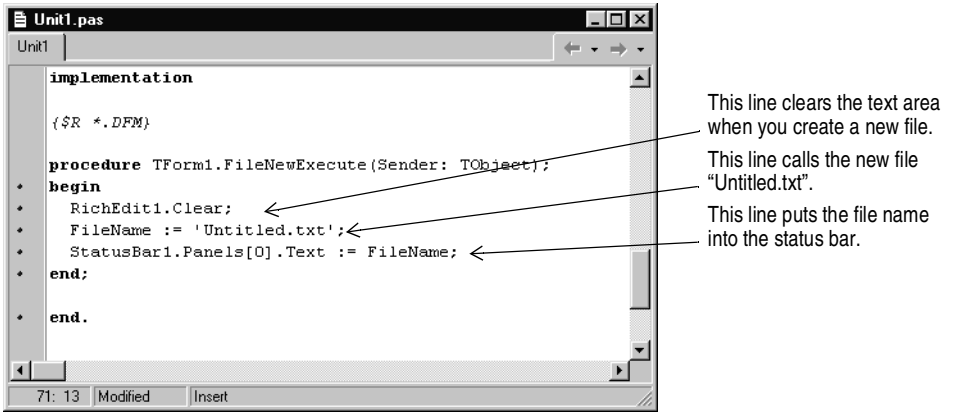
The Code editor opens with the cursor inside the event handler.




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

```
RichEdit1.Clear;
FileName := 'Untitled.txt';
StatusBar1.Panels[0].Text := FileName;
```

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




That's it for the File | New command.

## Creating an event handler for the Open command

When you open a file, a File Open dialog 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.

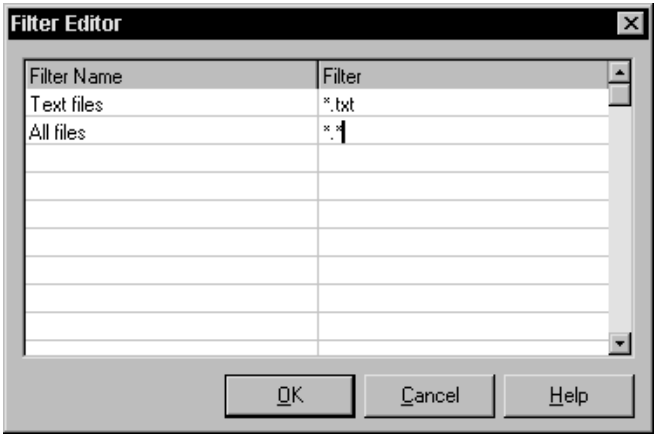
To create an Open dialog and an event handler for the Open command:

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

2 From the Dialogs page of the Component palette, drop an *OpenDialog* component onto the form. This is a nonvisual component, so it doesn't matter where you place it. Delphi names it *OpenDialog1* by default. (When *OpenDialog1*'s *Execute* method is called, it invokes a standard Windows dialog 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. Specify filters for file types: Type "Text files" as the Filter Name and *.txt as the filter and ""All files" as a second Filter Name and *.* as its filter). Then click OK.




- Set *Title* to Open File.

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

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

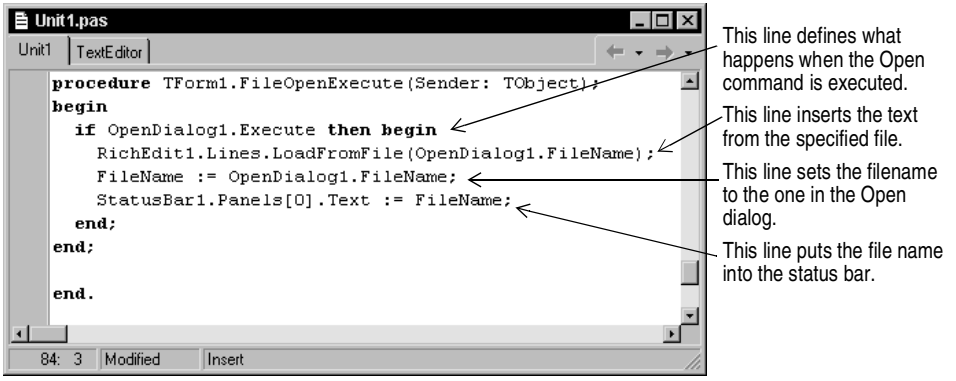
The Code editor opens with the cursor inside the event handler.

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

```
if OpenDialog1.Execute then begin
  RichEdit1.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.

## Creating an event handler for the Save command

To create an event handler for the Save command:

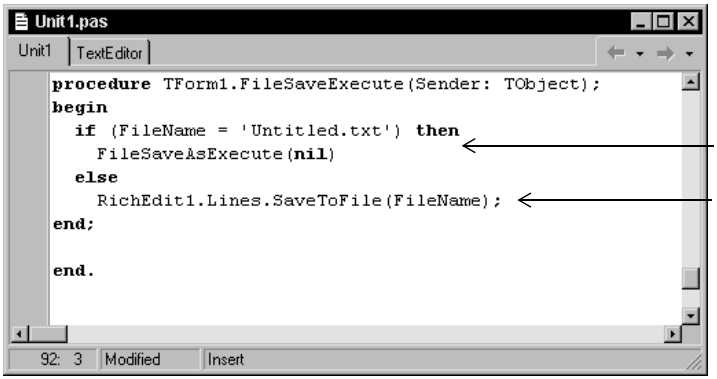
1. The Action List editor should still be displayed. If it's not, double-click the ActionList icon on the form.
2. On the Action List editor, double-click the FileSave action.

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

```
if (FileName = 'Untitled.txt') then
    FileSaveAsExecute(nil)
else
    RichEdit1.Lines.SaveToFile(FileName);
```

   This code tells the editor to display the SaveAs dialog if the file isn't named yet so the user can assign a name to it. Otherwise, save the file using its name. The SaveAs dialog is defined in the event handler for the Save As command on page 1-23. FileSaveAsExecute is the automatically generated name for the Save As command.

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




That's it for the File | Save command.

## Creating an event handler for the Save As command

To create an event handler for the Save As command:

1 From the Dialogs page of the Component palette, drop a *SaveDialog* component onto the form. This is a nonvisual component, so it doesn't matter where you place it. Delphi names it *SaveDialog1* by default. (When *SaveDialog*'s *Execute* method is called, it invokes a standard Windows dialog 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 editor, specify filters for file types as in the Open dialog (set Text files to *.txt and All files to *.*) then 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.

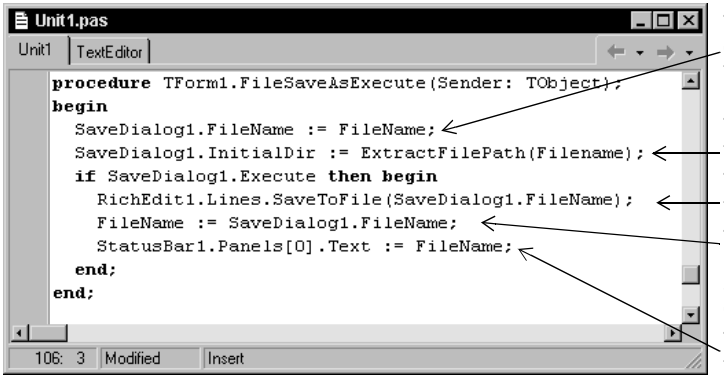
3 In the Action List editor, double-click the FileSaveAs action.

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

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

```
SaveDialog1.FileName := FileName;
SaveDialog1.InitialDir := ExtractFilePath(Filename);
if SaveDialog1.Execute then begin
    RichEdit1.Lines.SaveToFile(SaveDialog1.FileName);
    FileName := SaveDialog1.FileName;
    StatusBar1.Panels[0].Text := FileName;
end;
```

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




That's it for the File | SaveAs command.

## Creating an event handler for the Exit command

To create an event handler for the Exit command:

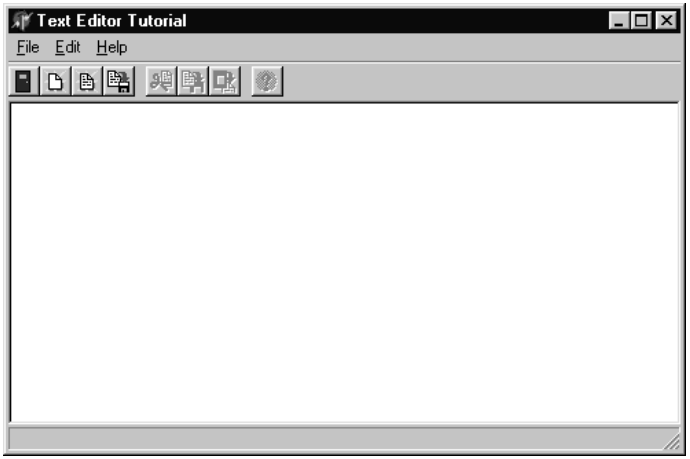
1. The Action List editor should still be displayed. If it's not, double-click the ActionList icon on the form.
2. On the Action List editor, double-click the FileExit action.

   The Code editor opens with the cursor inside the event handler.
3. Right where the cursor is positioned in the text 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* or by clicking on the green Run button ▶ on the toolbar.




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

You can close the application in three ways:

Click the **X**.

Choose File|Exit.

Click the Exit application button on the toolbar.

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

To return to design mode, close the Text Editor application by choosing File | Exit, by clicking the Exit application button on the toolbar of your application, or by clicking the **X** in the upper right corner.

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

## Creating a Help file

It's a good idea to create a Help file that explains how to use your application. Delphi provides Microsoft Help Workshop in the Help\Tools directory which includes information on designing and compiling a Windows Help file. In the sample editor application, users can choose Help | Contents or Help | Index to access a Help file with either the contents or index displayed.

Earlier, we created HelpContents and HelpIndex actions in the action list for displaying the Contents tab or Index tab of a compiled Help file. We need to assign constant values to the Help parameters and create event handlers that display what we want.

To use the Help commands, you'll have to create and compile a Windows Help file. Creating Help files is beyond the scope of this tutorial. A sample rtf file (TextEditor.rtf), Help file (TextEditor.hlp) and contents file (TextEditor.cnt) are downloadable from the http://www.borland.com/techpubs/delphi/ Web site. Or, to test the Help, you can use any HLP or CNT file (such as one of the Delphi Help files and its associated CNT file) in your project. You will have to rename them for the application to find them.

## Creating an event handler for the Help Contents command

To create an event handler for the Help Contents 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, select the Help category, then double-click the HelpContents action.

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

3 Right before where the cursor is positioned in the text editor, that is, right before begin, type the following lines:

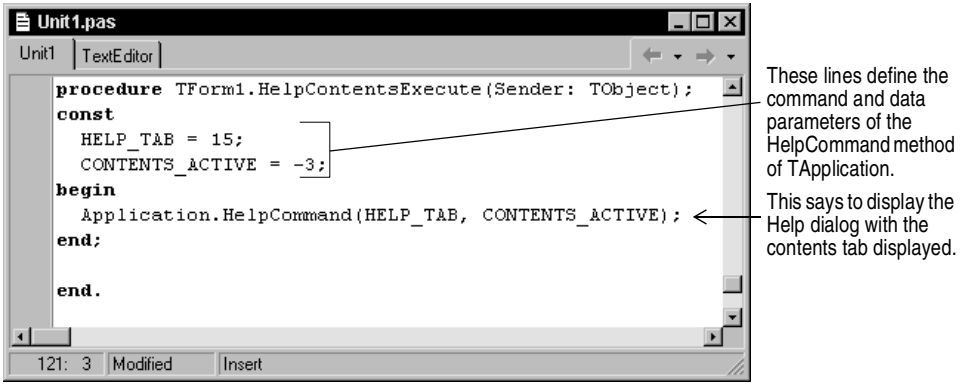
```
const
  HELP_TAB = 15;
  CONTENTS_ACTIVE = -3;
```

Right after `begin`, type:

```
Application.HelpCommand(HELP_TAB, CONTENTS_ACTIVE);
```

This code assigns constant values to the HelpCommand parameters. Setting HELP_TAB to 15 displays the Help dialog and setting CONTENTS_ACTIVE to -3 displays the Contents tab.

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




**Tip** To get Help on the HelpCommand method, put the cursor next to HelpCommand in the editor and press *F1*.

That's it for the Help | Contents command.

## Creating an event handler for the Help Index command

To create an event handler for the Help Index 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, select the Help category and then double-click the HelpIndex action.

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

3 Right before where the cursor is positioned in the text editor, that is right before begin, type the following lines:

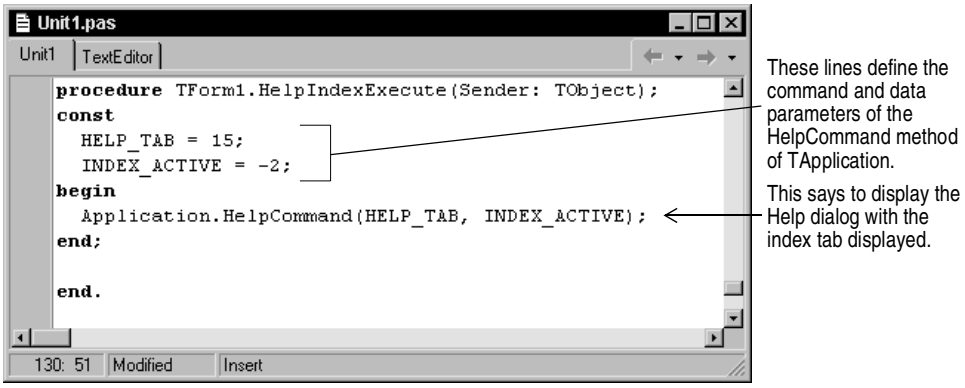
```
const
  HELP_TAB = 15;
  INDEX_ACTIVE = -2;
```

Right after `begin`, type

```
Application.HelpCommand(HELP_TAB, INDEX_ACTIVE);
```

This code assigns constant values to the HelpCommand parameters. Setting HELP_TAB to 15 again displays the Help dialog and setting INDEX_ACTIVE to -2 displays the Index tab.

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




That's it for the Help | Index command.

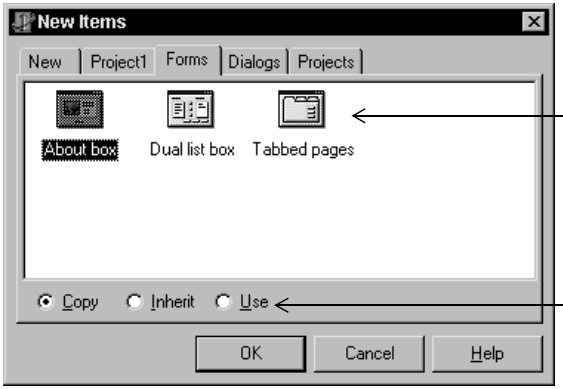
# Creating an About box

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

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

To create an About box:

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



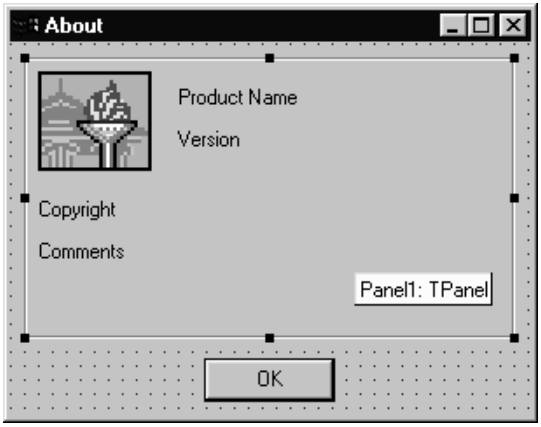

The New Items dialog box is also called the Object Repository.

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.

2 On the Forms tab, choose About Box.

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




A standard About box is created when you choose File|New and click About Box on the Forms tab. You can modify it as you like to describe your application.

3 Select the following *TLabel* items in the About box and change them in the Object Inspector:

- Change Product Name to Text Editor.
- Make it Version 1.0.
- Enter the year next to Copyright.

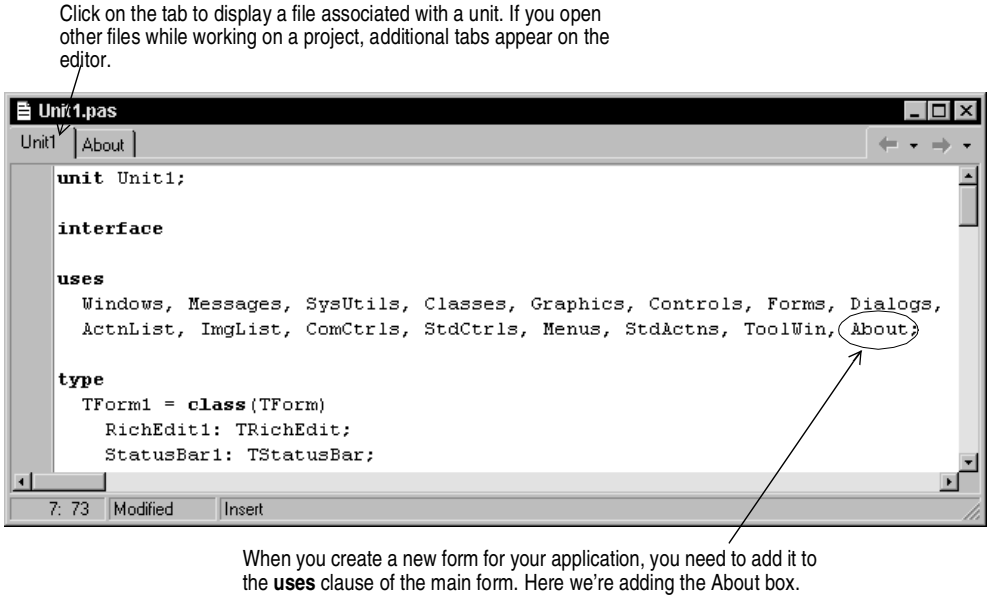
4 Select the form itself and change its *Caption* in the Object Inspector to About Text Editor.

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

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

6 In the Delphi editor, you should have two files displayed: Unit1 and About. Click on the Unit1 tab to display Unit1.pas.

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




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

9 Right where the cursor is positioned in the text 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. That means the user can't do anything until the form is closed.

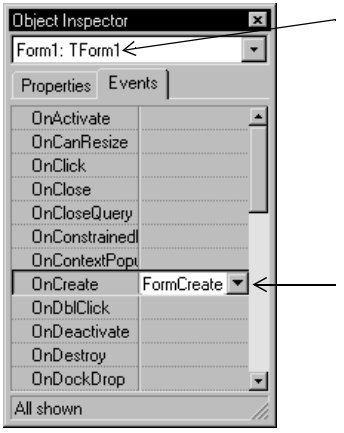
# Completing your application

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 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 text editor, type the following lines:

```
Application.HelpFile := ExtractFilePath(Application.ExeName) + 'TextEditor.hlp';
FileName := 'Untitled.txt';
StatusBar1.Panels[0].Text := FileName;
RichEdit1.Clear;
```

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

5 Put the .HLP file and the CNT file into the project application directory (called projects\TextEditor).

**Note** If you decided not to investigate how to create a Help file or use the sample one provided on the web, the application still works but you'll receive an error message when you choose either of the Help commands or click Help on the toolbar.

6 Press *F9* to run the application.

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

Congratulations! You're done.