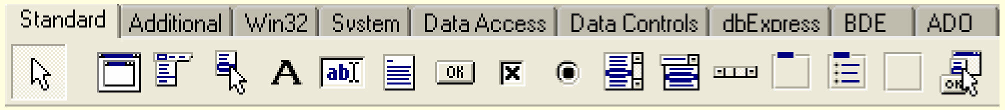
Standard tab GUI components

GUI components

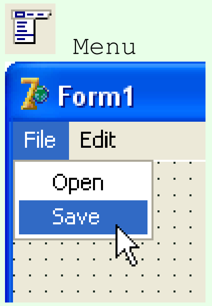
GUI stands for Graphical User Interface. It refers to the windows, buttons, dialogs, menus and everything visual in a modern application. A GUI component is one of these graphical building blocks. Delphi lets you build powerful applications using a rich variety of these components.

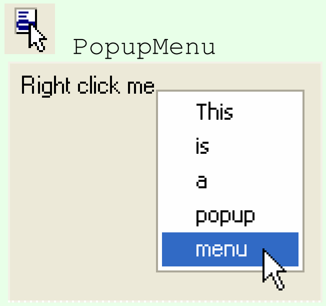
A rough metaphor for GUI would be the rotating handle on a MJJ market umbrella stands turning the unseen mechanism hidden within the umbrella.

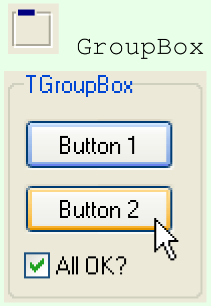
These components are grouped under a long set of tabs in the top part of the Delphi screen, starting with **Standard** at the left. We'll look at this Standard tab here. It looks something like this (Delphi allows you to tinker with nearly everything in its interface, so it may look different on your system):

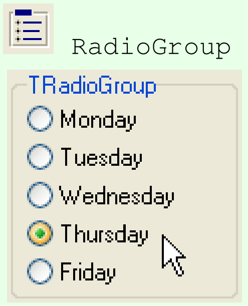


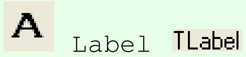
Each of the components is itemised below with a picture of a typical GUI object they can create:

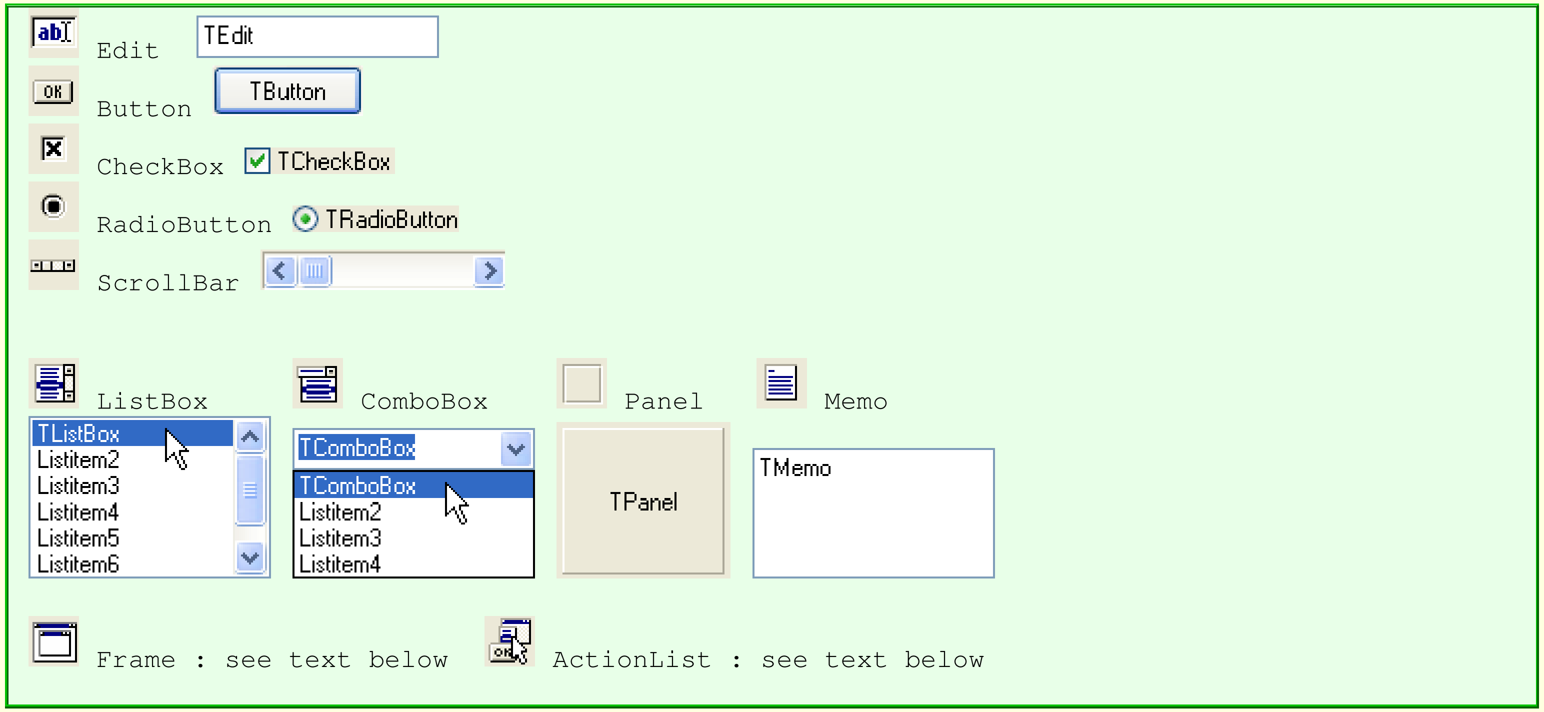

Menu


PopupMenu


GroupBox


RadioGroup


Label



Note that the displayed components were taken from an XP computer. In order to get the new XP look (the XP 'themed' GUI look), you must add the **XP Manifest** component to you form. It is found under the **Win32** component tab:

XP Manifest component.

We'll now cover each of the components in turn. Components have many properties and methods and events, but we'll keep the descriptions to the point to keep this article short enough. Each component is added to you form by clicking it and then clicking (or dragging and releasing) on your form.

Frame objects

These were introduced in Delphi 5. They represent a powerful mechanism, albeit one that is a little advanced for a Delphi Basics site. However, it is worth describing their role if you want to research further.

A frame is essentially a new object. It is defined using the **File|New** menu. Only then can you add the frame to your form using the **Frame** component. You can add the same frame to as many forms of your application as you want. This is because the frame is designed as a kind of template for a part of a form. It allows you to define the same look and feel for that part of each form. And more importantly, each instance of the frame inherits everything from the original frame.

For further reading, **Mastering Delphi** by **Cantu** covers this topic with example code.

Menus

After you add a TMenu component to your form, you can design the menu by double clicking it (or using the right button popup menu for it). You are then shown a panel with an empty menu. As you type, you are creating the top left menu item. Press enter and you are positioned at the first sub item of this menu item. Click the new empty box to the right of the first menu item to create a new menu item.

In this way, you can build the menu structure.

To make each menu item do something, just double click it. Delphi will then insert code into your program to handle the menu item, and position your cursor in the form unit ready for you to write your code.

Explore the popup menu for the menu editor to discover more options, such as sub-menus.

A menu can also be dynamically updated by your code.

Popup menus

A popup menu appears in many applications when you right click on something. For example, when you right click the Windows desktop. You create a popup menu by adding the popup menu component to your form and double clicking it. You then simply type in your menu item list.

You attach the popup menu to an existing form object (or the form itself) by selecting your new popup menu in the **PopupMenu** property of the object.

To activate the popup menu items, double click each in turn. Delphi will add the appropriate code to your form unit. You can then type in the code that each menu item should perform.

A popup menu can also be dynamically updated by your code.

Labels

Labels are the simplest component. They are used to literally label things on a form, but the text, colours and so on can be changed by your code. For example, you can change the label colour when the mouse hovers over it, and can run code when the user clicks it. This makes the label like a web page link. Normally, they are just kept as plain, unchanging text.

Edit boxes

An edit box allows the user to type in a single line of text. For example, the name of the user. You set up the initial value with the **Text** property either at design time or when your code runs.

Memo boxes

A memo box displays a single string as a multi line wrapped text display. You cannot apply any formatting. The displayed lines are set using the **Lines** property. This may be set at design time as well as at run time.

Buttons

A button is the simplest active item. When clicked by a user, it performs some action. You can change the button label by setting the **Caption** property. Double clicking the button when designing adds code to your form to run when the button is clicked at run time.

Check boxes

Check boxes are used to give a user a **yes/no** choice. For example, whether to wrap text or not. The label is set using the **Caption** property. You can preset the check box to ticked by setting the **Checked** property to **true**.

Radio buttons

Radio buttons are used to give a user **multiple** choices. For example, whether to left, centre or right align text. The label is set using the **Caption** property. You can preset a radio button to selecteded by setting the **Checked** property to **true**.

You would normally use radio buttons in groups of two or more. The **TRadioGroup**component allows you to do this in a neat and dynamic way.

List boxes

List boxes provide selectable items. For example, a collection of fish names. If you set the **MultiSelect** property to **true**, you allow the user to select more than one. The items in the list are added using the **Items.Add** method, passing the string of each item as a parameter.

You can act upon an item being selected by setting the **OnClick** event (by double clicking it) to a procedure in your form unit.

The following example displays the selected list item in a dialog box:

```
procedure TForm1.ListBox1Click(Sender: TObject);
var
  listBox : TListBox;
  index   : Integer;
begin
  // Cast the passed object to its correct type
  listBox := TListBox(Sender);

  // Get the index of the selected list item
  index   := listBox.ItemIndex;

  // Display the selected list item value
  ShowMessage(listBox.Items[index]);
end;
```

Combo boxes

A combo box is like a list box, and is set up in the same way (see above). It just takes up less space on your form by collapsing to a single line when deselected, showing the chosen list item. It is not recommend to use one for multi line selection.

Scroll bars

Many components have built in scroll bars. For those that don't, you can use this to do your own scrolling. You link the scrollbar to your component by setting the **OnScroll**event. This gives you the details of the last scroll activity made by the user.

Group boxes

A group box is like a panel. It differs in that it gives a name to the collection of components that you add to it. This title is set with the **Caption** property. Use a group box to help the user see what controls affect one particular aspect of the application.

Radio group panels

Radio buttons are used to give a user a **multiple** choices. For example, whether to left, centre or right align text. Unlike individual radio buttons, a group is only set up by your code. You define the buttons by calling the **Items.Add** method of the TRadioGroup object, passing the caption string of each radio button as a parameter. You can reference each button by using the **Buttons** indexed property. You might, for example, choose the third button to be checked. For example :

```
  // Set the third button to be pre-selected (index starts at 0)
  RadioGroup1.Buttons[2].Checked := true;
```

Empty panels

When building your form, you might want to add many components. These may fall into logical groups. If so, you can add each group to a panel, and use the panel to position the whole group on the form. The panel name can be blanked out by setting the**Caption** property.

You can even hide the panel by setting the **BevelOuter** and **BevelInner** properties to**bvNone**.

Action lists

Action lists are a large topic on their own. They allow you to define, for example, menus with sub-items that are also shown as buttons on your aplication. Only one **action** is defined, regardless of the number of references to it.