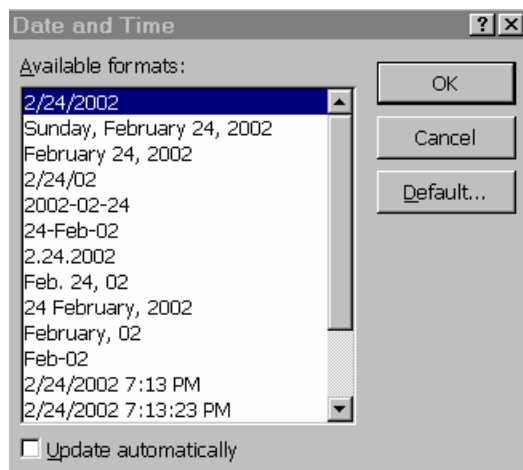


Controls and the Dialog Editor

Windows provides some standard interface devices known as *controls*, by means of which a user interacts with a program. Some commonly used controls are (but this is not an exhaustive list):

- edit boxes
- labels
- picture boxes
- push buttons
- radio buttons
- check boxes
- scroll bars and sliders
- list boxes
- combo boxes
- spin controls

To start with, we'll look at examples of each of these and study what their purposes are. Here's an example, snapped from Word:

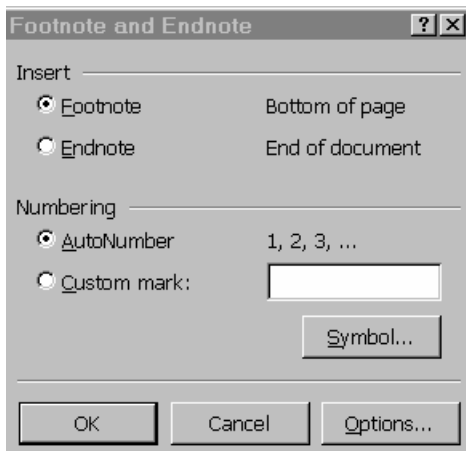


This small window comes up when some menu item is chosen, and disappears (is “dismissed”) when the user presses OK or Cancel. Such a window is called a “modal dialog”. This is an extremely common way to use controls.

This particular dialog uses a list box and three pushbuttons. At the bottom there is a check box. The check box can be checked, or not. The list box is used to select ONE of the available options. The OK button and cancel button, as in almost all dialogs, close the dialog box and either cause the choices to take effect, or be discarded. In this dialog there is also a Default button that causes them to become the default choices in the future.

Push buttons in general are used to *invoke actions*. That is, to *make something happen*, as opposed to just building up some data selections. Generally you use other controls to select or enter some data and then a push button to *do something with the data*.

Here's an example showing some radio buttons and an edit box:



Radio Buttons and Check Boxes compared

Radio buttons are used when the user has to choose one of several items, and is not allowed to select more than one.

Check boxes are used to select options, where the options can be selected independently, so you *can* choose more than one, or none.

The checkbox on the previous screen shot (*Update Automatically*) could be done with TWO radio buttons:

- *Update Automatically*
- *Don't Update Automatically*

of which only one could be selected. But a checkbox is better.

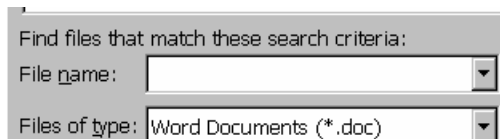
The *Footnote/Endnote* choice above cannot be reduced to a checkbox, since the alternatives need to be communicated.

Edit boxes and List boxes compared

Edit boxes are used to allow entry of possibly unanticipated text. If the possible choices come from a fixed list, use a list box. You do not have to know the list at compile time--the program can add items to the list box at run time.

Combo boxes

If you can anticipate some possible or likely choices, but unanticipated entries can still be typed in legally, use a combo box. Here's an example:



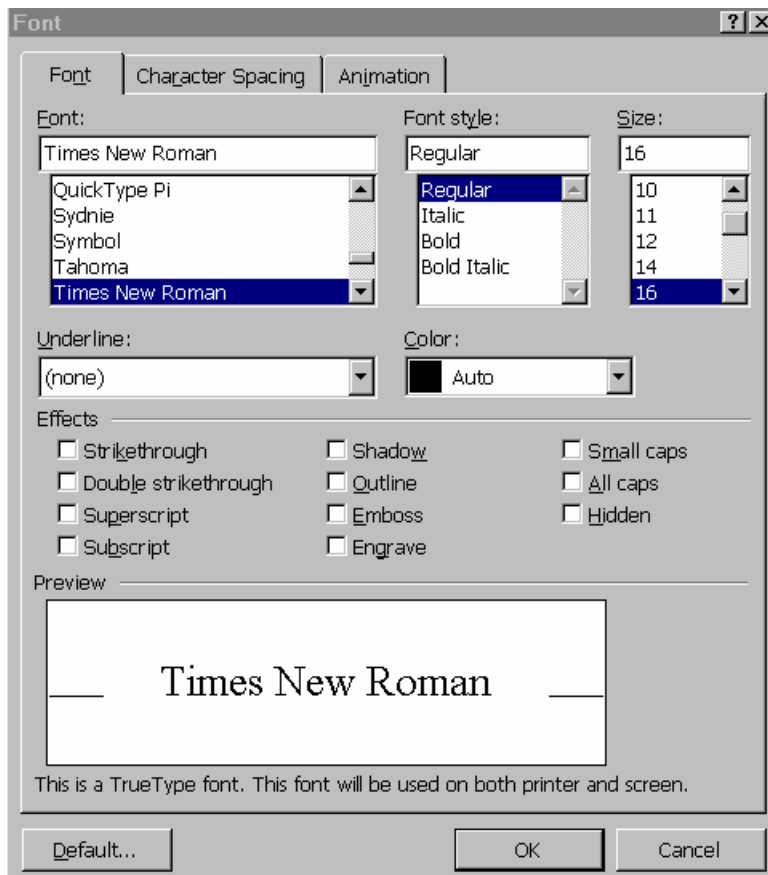
Find files that match these search criteria:

File name:

Files of type: Word Documents (*.doc)

You can type in a new filename, or select a recent file from the drop-down list.

Combo boxes should not be used where the list of choices is restricted and known. Here's an example of misuse in an old version of Microsoft Word:

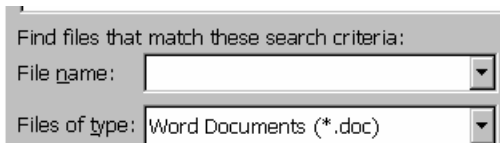


You can type in 13 for the point size, but (in an old version of Word) you can't actually get 13-point type. It came out 14-point. The current version of Word does allow 13-point type, and any size you type in. But if, as used to be the case, the possible point sizes are known in advance, it merely misleads the user to let them type in 13.

A similar problem still exists with the *Font Style* box. If you type in "Greek" (or anything but the four legal choices) it will give you an error message. A list box should have been used instead of a combo box, since there are exactly four legal choices.

Labels and Picture Boxes

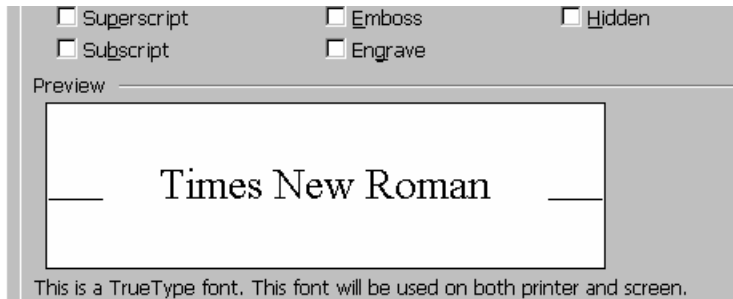
Labels and picture boxes are quite similar controls, both related to the underlying Win32 control “static text box”. Both provide a child window in which the programmer can do just about anything. The names “label” and “picture box” correspond to the two most common uses of such a window. Here’s an example from Word.



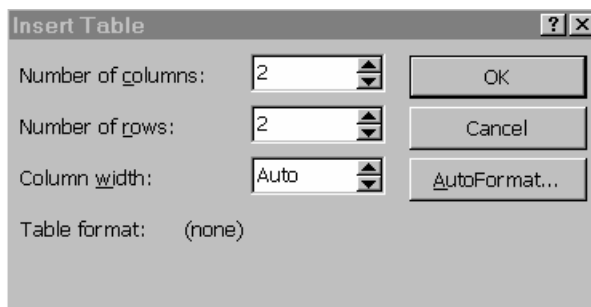
If this had been programmed in .NET, then the text "File name" would be in a label control, and the text "Files of type" would be in another label. You do not see the boundaries of these text boxes because their background colors are the same as the dialog's background color and they have no visible border. Nevertheless the *Label* control is not just the text; it is a small window that displays the text.

The text in a *Label* is specified in the *Text* property of the label. Of course it can be changed while the program is running. For that reason the old name “static textbox” was somewhat misleading; but what it was meant to convey is that the *user* cannot change that text. If you want the *user* to be able to edit the text, you need to use a *TextBox* control.

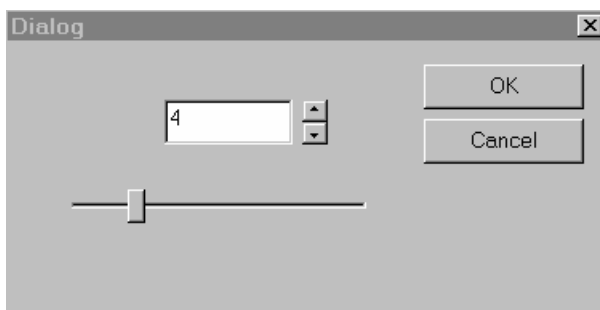
Using a label control to draw graphics



The white box after "Preview" is really a static text box ("label" in .NET) this time with a white background. It is used to draw a sample of the font in the selected style and size. Any Windows graphics can be used to draw in a static text box.



This dialog illustrates *spin controls*.



This dialog shows an edit box with a spin control, and a slider that communicates with the edit box. When you change the contents

of the edit box, the slider moves, and vice-versa, when you move the slider, it changes the number in the edit box.

A slider can be used by itself, of course, as in the volume control of an audio player, but in this course you will learn how to make a slider communicate with other controls while the dialog box is running.

Controls are windows

It is very important to understand that each control is really a window.

A list box is a window.

A button is a window.

An edit box is a window.

A radio button is a window.

A check box is window.

A static text box is a window.

Etc. for all the other controls.

Controls in the Foundation Class Libraries

There are pre-defined window procedures for each of these types of windows in the underlying Win32 API. These have been wrapped in classes in the .NET Foundation Class Libraries, so when you want to create a control (as a .NET programmer) you just create a new object of the appropriate class. You can do that in source code, or you can let Visual Studio write the source code as you use the “form editor”, adding controls by drop-and-drag and modifying them through their property sheets.