

Advanced Programming Methods Lecture 10

C# GUI

Contents

- System.Drawing Namespace
- System.Windows.Forms Namespace
 - Creating forms applications by hand
 - Creating forms applications using Visual Studio designer

Forms Programming

- Forms programming allows you to create stand-alone Windows GUI applications
- The API has been modernized for .NET
- It now supports a modern delegate-based programming model
- Forms programming uses
 - System.Drawing
 - Basic GDI+ functionality
 - System.Windows.Forms
 - Higher-level controls

System.Drawing

- This namespace provides many graphic data structures which are used throughout the GUI programming model
- It also provides support for low-level drawing operations
- These can be used to draw anything, not just what is offered by the pre-built controls

System.Drawing.Point

Structure which represents a 2-D point

Constructor

```
Point(int x, int y)
```

Properties

X – get/set of X coordinate

Y – get/set of Y coordinate

System.Drawing.Size

Structure which stores the width and height of something

Constructor

```
Size(int width, int height)
```

Properties

`Width` – get/set width

`Height` – get/set height

System.Drawing.Rectangle

Structure representing a rectangle as the point of the top-left-corner, width and height

Constructor

`Rectangle(Point tlc, Size sz)`

`Rectangle(int tlx, int tly, int wd, int ht)`

Properties

`X` – get/set top left X coordinate

`Y` – get/set top left Y coordinate

`Height` – get/set height

`Width` – get/set width

`Bottom` – get Y coordinate of rectangle bottom

`Top` – get Y coordinate of rectangle top

`Left` – get X coordinate of right of rectangle

`Right` – get X coordinate of left of rectangle

System.Drawing.Color

Structure representing an alpha-RGB color

Methods

`Color FromARGB(int r, int g, int b)`

`Color FromARGB(int alpha, int r, int g, int b)`

Properties

`A` – get alpha value

`R` – get red value

`G` – get green value

`B` – get blue value

`Black, White, Red, Green, Yellow, Cyan, Coral, Blue, etc.` –
get values of pre-defined colors

System.Drawing.Font

Class representing a font, size and style

Constructor

`Font(string family, int points, FontStyle style)`

Properties

`FontFamily` – get the FontFamily value

`Style` – get the FontStyle Value

`Size` – get the font size

`Bold` – get true if bold

`Italic` – get true if italic

System.Drawing.FontStyle

An enumeration with members

Bold, Italic, Regular, Strikeout, Underline

The values can be OR-ed together to indicate that more than one style should apply at once

System.Windows.Forms

- This namespace contains all of the controls used on the average Windows interface
- A control is a higher-level object composed of
 - A window in which the control is drawn
 - Visual parts of the control which are drawn in the window
 - A set of delegates which are triggered when various events occur

Form Class

- This is the top-level window class
- This class contains all other controls
- Normally, your top-level form inherits from the Form class
- Although the class has numerous methods, most of the time you interact with it via properties and delegates

Form Properties

Property	Description
Location	Point of to left corner
Size	Size of form in pixels
Text	Text displayed or caption
AutoScaleDimensions	DPI resolution of display it was built for. Will be scaled to look correct on other displays.
BackColor	Background color
ForeColor	Foreground or drawing color
ClientSize	Size of drawing area without borders or scrollbars
Controls	A collection of controls owned by the form
WindowState	Whether maximized, minimized or normal
DefaultSize	Size when initially created
MinimumSize	Minimum size window can be resized to
MaximumSize	Maximum size window can be resized to

Form Events

- Forms provide support for a large number of events
- You add one or more delegates to these events
- When the event happens, the delegates are invoked
- The delegates must have the signature of an event handler

```
void EventHandler(object sender, EventArgs e)
```

Form Events

Event	Description
Load	Just before form is loaded the first time
Closing	Just before the form is closed
Closed	When the form is actually closed
Shown	Occurs when a form is first displayed
ResizeBegin	Resize operation has begun
ResizeEnd	Resize operation has ended

Form Methods

Method	Description
Activate	Activates the window and gives it focus
Close	Closes the form
Show	Makes the form visible
BringToFront	Moves to top of stacking order
Hide	Makes the form invisible
Focus	Gives the form focus

Creating Windows Applications

- We will demonstrate how to build a simple GUI interface using a text editor
- Most of the time, the designer in Visual Studio will be used
- Doing it by hand
 - Shows how it works under the hood
 - Let's you make modifications by hand
 - Provides an understanding so you can create your own controls

Creating Windows Applications

In creating a GUI application we will use

- Application – a class with static methods to control operation of an application
- Label – a widget that can display static text or an image
- Button – a push button with a textual or image displayed. Able to respond to mouse clicks.

Creating Windows Applications

The first step is to create a class which

- Inherits from Form
- Declares the widgets within it

```
public class GreetingForm : Form {  
    Label    greetingLabel;  
    Button   cancelButton;  
  
    ...  
}
```

Creating Windows Applications

Next, create the label and set its properties

```
greetingLabel = new Label();  
greetingLabel.Location = new Point(16, 24);  
greetingLabel.Text = "Hello, World";  
greetingLabel.Size = new Size(216, 24);  
greetingLabel.ForeColor = Color.Black;
```

Creating Windows Applications

Create the cancel button and set its properties

```
cancelButton = new Button();  
cancelButton.Location = new Point(150, 200);  
cancelButton.Size = new Size(112, 32);  
cancelButton.Text = "&Cancel";  
cancelButton.Click += new  
    EventHandler(cancelButton_Click);
```

Creating Windows Applications

Set the properties of the main form

```
this.AutoScaleDimensions = new  
    SizeF(95.0f, 95.0f);  
this.ClientSize = new Size(300, 300);  
this.Text = "Hello, World";
```

Creating Windows Applications

Add the controls to the form

```
this.Controls.Add(cancelButton);  
this.Controls.Add(greetingLabel);
```

And provide the event handler

```
protected void cancelButton_Click(  
    object sender, EventArgs e) {  
    Application.Exit();  
}
```

*** see the attached Example HelloForm**

The Application Class:

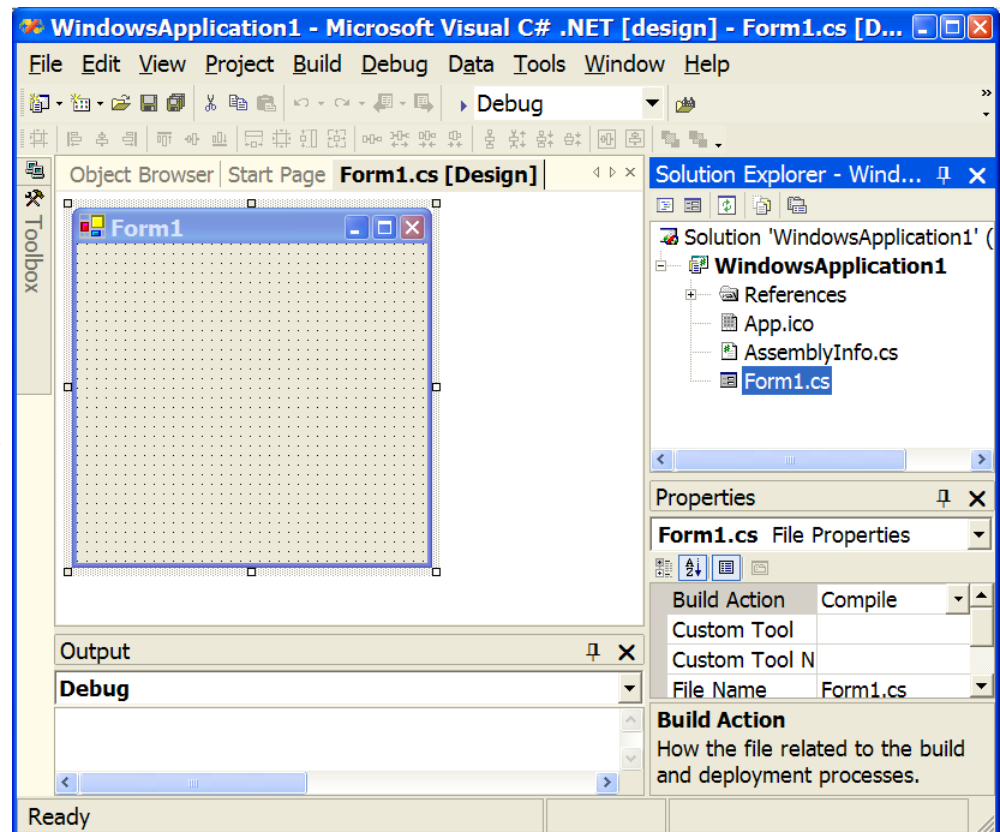
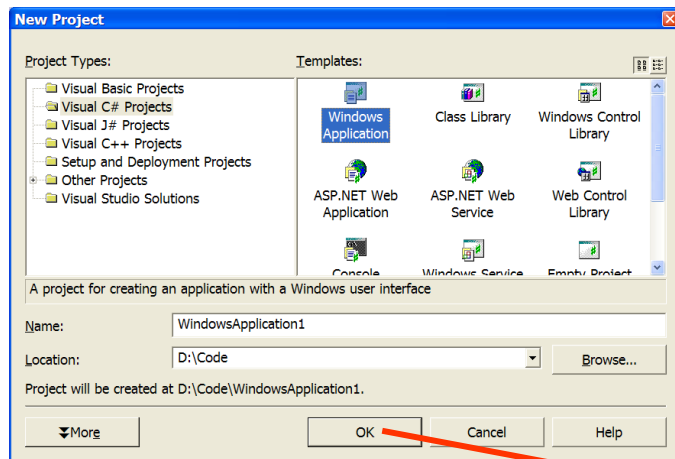
- provides methods to start and stop applications and threads, and to process Windows messages.
- calling **Run** method starts an application **message loop** on the current thread and makes **a form visible**
- the method **Run** adds an event handler to the mainForm parameter for the **Closed event**. The event handler calls **ExitThread** to clean up the application.

Visual Studio Designer

- This is a drag and drop interface for drawing a GUI
- The code is automatically generated
- You can hook event handlers onto the events and write the code for them
- It speeds writing code
- You cannot make major modifications to the code it generates

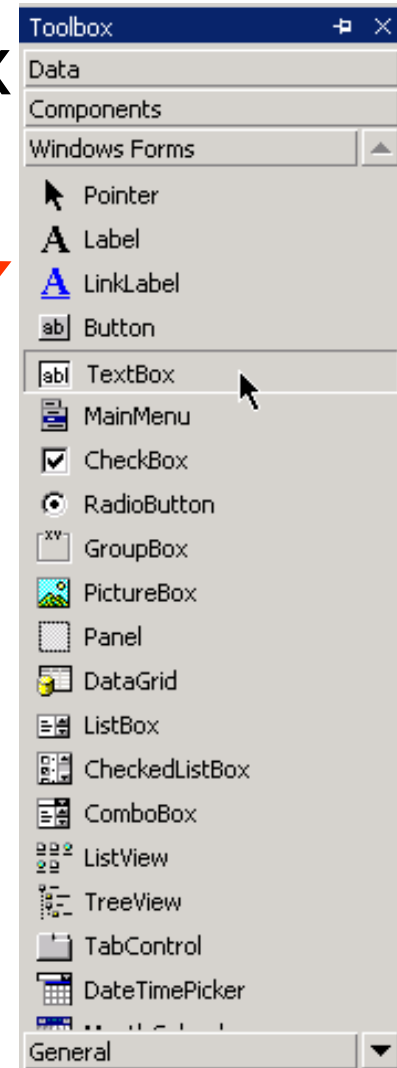
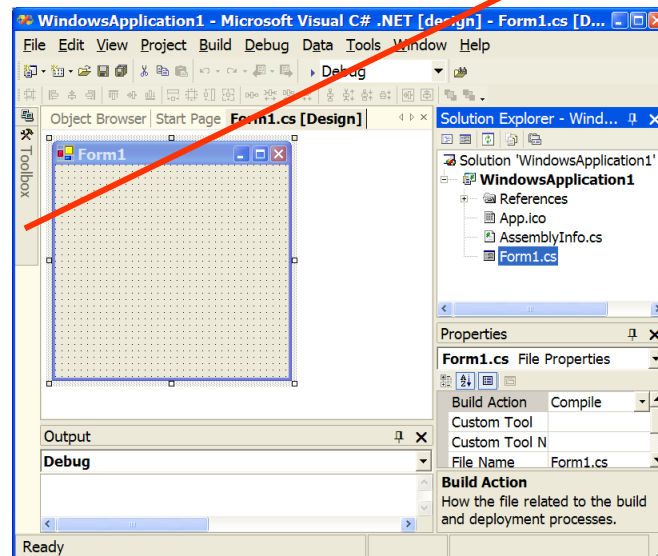
Step 1

Create a new project of type “Windows Application”
a form will be created for you automatically...



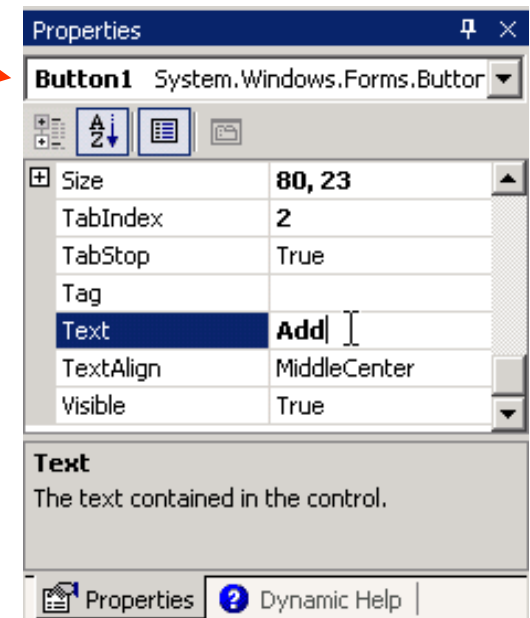
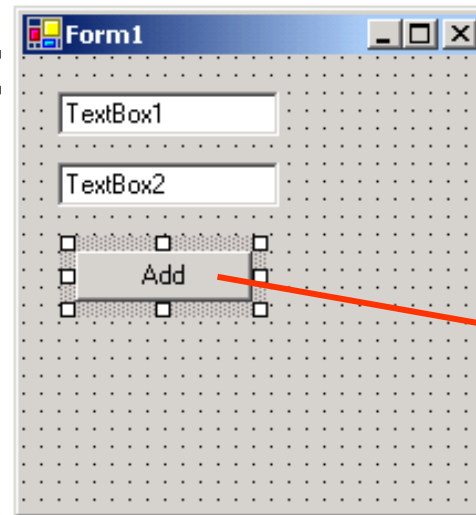
Step 2 — GUI design

Select desired controls from toolbox
hover mouse over toolbox to reveal
drag-and-drop onto form
position and resize control



GUI design cont'd...

A simple calculator:

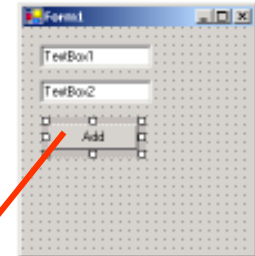


Position and configure controls
click to select
set properties via Properties window

Step 3 — code design

“Code behind” the form...

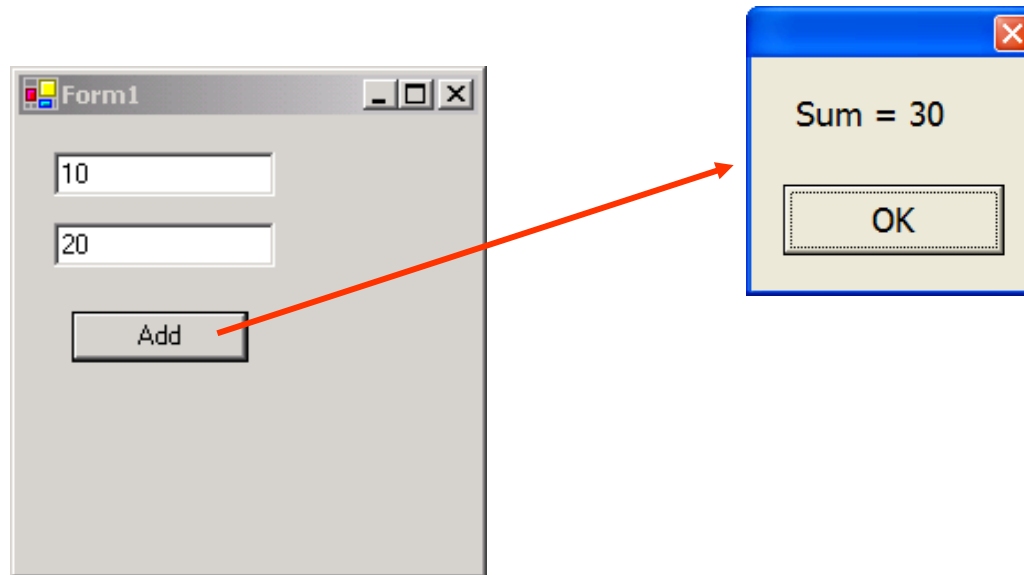
Double-click the control you want to program
reveals coding window



```
/**/  
static void Main() ...  
  
private void button1_Click(object sender, System.EventArgs e)  
{  
    int i, j, k;  
    i = System.Convert.ToInt32( this.textBox1.Text );  
    j = System.Convert.ToInt32( this.textBox2.Text );  
    k = i + j;  
    System.Windows.Forms.MessageBox.Show("Sum = " + k);  
}  
}
```

Step 4 — run mode

Run!



Simple Windows Application

IDE separates the source code into three separate files

- Form1.cs: normally this is the only one you edit
- Form1.Designer.cs: holds the auto generated code
- Program.cs: contains the Main() method, where execution always begins

Form1.cs and Form1.Designer.cs both include partial class definitions for the Form1 class

Visual Studio Generated Code

- The auto-generated code is saved in the Designer.cs file of the Form
- `partial` modifier allow the class created to be split among multiple files
- By default, all variable declarations for controls created through C# have a `private` access modifier
- The code also includes `Dispose` and `InitializeComponent`

Class Control properties and methods	Description
<i>Common Properties</i>	
BackColor	The control's background color.
BackgroundImage	The control's background image.
Enabled	Specifies whether the control is enabled (i.e., if the user can interact with it). Typically, portions of a disabled control appear “grayed out” as a visual indication to the user that the control is disabled.
Focused	Indicates whether the control has the focus.
Font	The Font used to display the control's text.
ForeColor	The control's foreground color. This usually determines the color of the text in the Text property.
TabIndex	The tab order of the control. When the <i>Tab</i> key is pressed, the focus transfers between controls based on the tab order. You can set this order.
TabStop	If true , then a user can give focus to this control via the <i>Tab</i> key.

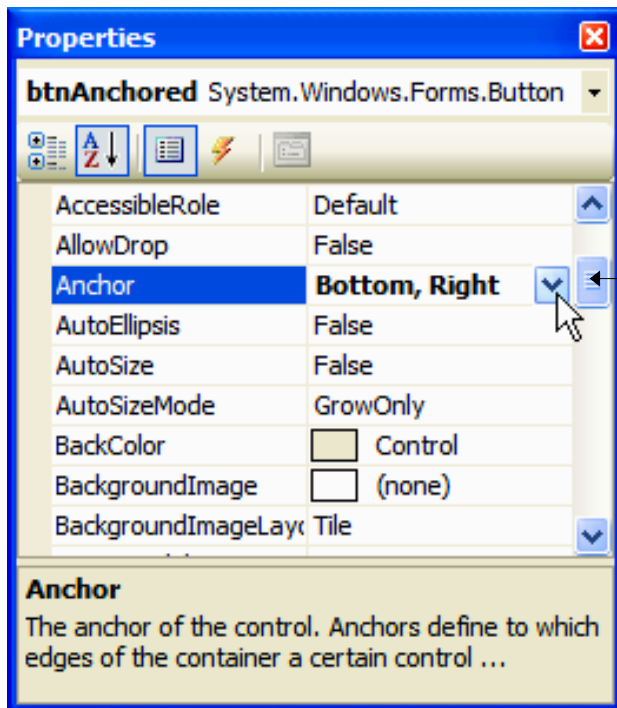
Class **Control** properties and methods. (Part 1 of 2)



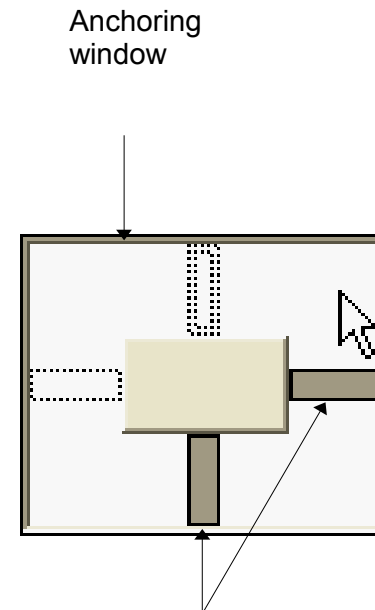
Class Control properties and methods	Description
TabStop	If true , then a user can give focus to this control via the <i>Tab</i> key.
Text	The text associated with the control. The location and appearance of the text vary depending on the type of control.
visible	Indicates whether the control is visible.
<i>Common Methods</i>	
Focus	Acquires the focus.
Hide	Hides the control (sets the Visible property to false).
Show	Shows the control (sets the Visible property to true).

Class **Control** properties and methods. (Part 2 of 2)





Click down-arrow in **Anchor** property to display anchoring window



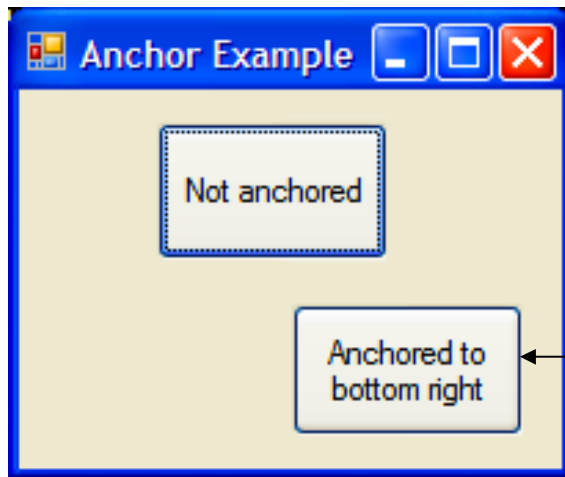
Darkened bars indicate the container's side(s) to which the control is anchored; use mouse clicks to select or deselect a bar

Manipulating the **Anchor** property of a control.

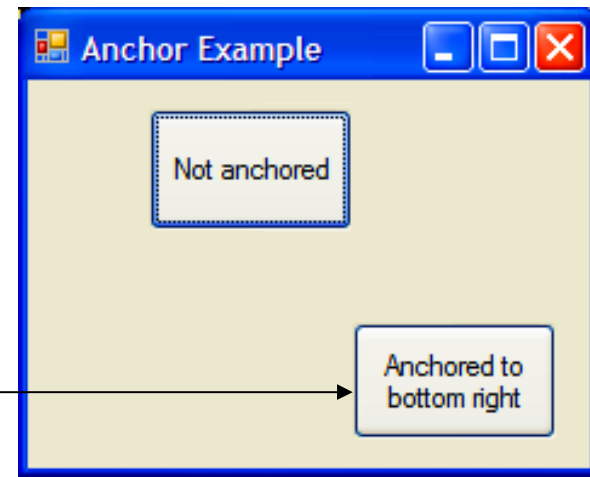


Before resizing

After resizing

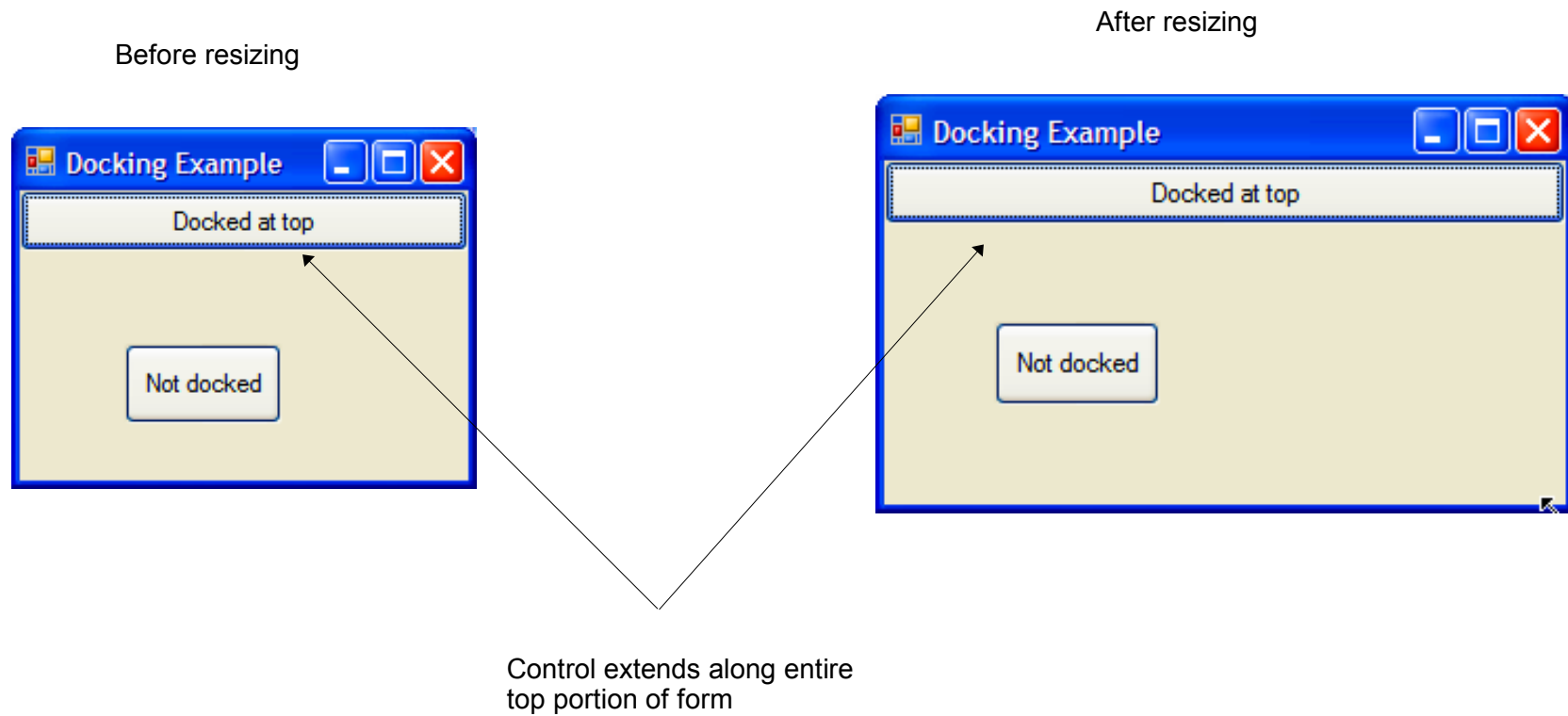


Constant distance to
right and bottom
sides



Anchoring demonstration.





Docking a **Button** to the top of a **Form**.

Control layout properties	Description
Anchor	Causes a control to remain at a fixed distance from the side(s) of the container even when the container is resized.
Dock	Allows a control to span one side of its container or to fill the entire container.
Padding	Sets the space between a container's edges and docked controls. The default is 0, causing the control to appear flush with the container's sides.
Location	Specifies the location (as a set of coordinates) of the upper-left corner of the control, in relation to its container.
Size	Specifies the size of the control in pixels as a Size object, which has properties Width and Height .
MinimumSize, MaximumSize	Indicates the minimum and maximum size of a Control , respectively.

Fig. 13.15 | Control layout properties.



CheckBoxes



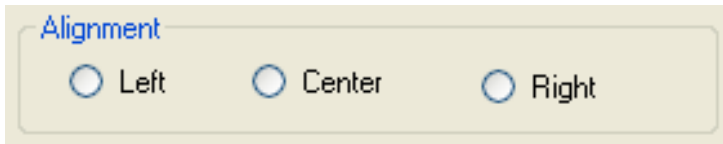
Labeled boxes which can be checked or unchecked

`Checked` – get/set Boolean to determine if box is checked

`CheckedChanged` – delegate called when the box is checked or unchecked

* see the attached example `ListBoxDemo`

GroupBox



Displays a border around a group of controls

Can have optional label controlled by Text property

Controls can be added by

Placing them within the group box in the designer

Adding to the Controls list programmatically

* see the attached example TextBoxDemo

Panels

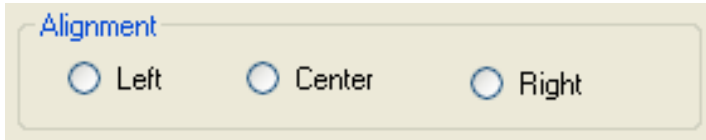
A panel is like a group box but does not have a text label

It contains a group of controls just like group box

`BorderStyle` – get/set border style as

- `BorderStyle.Fixed3D`
- `BorderStyle.FixedSingle`
- `BorderStyle.None`

Radio Buttons



Radio buttons are similar to checkboxes, but

Appear slightly different

Allow buttons to be grouped so that only one can be checked at a time

A group is formed when the radio buttons are in the same container – usually a group box or panel

Radio Buttons

`Checked` – get/set Boolean indicating if the button is checked

`CheckedChanged` – delegate invoked when the button is checked or unchecked

* see the attached example `TextBoxDemo`

TextBox

This is a single line or multi-line text editor

`Multiline` – get/set Boolean to make multiline

`AcceptsReturn` – in a multiline box, if true then pressing Return will create a new line. If false then the button referenced by the `AcceptButton` property of the form, will be clicked.

`PasswordChar` – if this is set to a char, then the box becomes a password box

TextBox

`ReadOnly` – if true, the control is grayed out and will not accept user input

`ScrollBars` – determines which scrollbars will be used: `ScrollBars.None`, `Vertical`, `Horizontal`, `Both`

`TextAlign` – get/set `HorizontalAlignment.Left`, `Center`, or `Right`

`TextChanged` – event raised when the text is changed

File Dialog

The file dialog allows you to navigate through directories and load or save files

This is an abstract class and you use

`OpenFileDialog`

`SaveFileDialog`

You should create the dialog once and reuse it so that it will remember the last directory the user had navigated to

File Dialog

`InitialDirectory` – string representing the directory to start in

`Filter` – a string indicating the different types of files to be displayed

A set of pairs of display name and pattern separated by vertical bars

- `Windows Bitmap|*.bmp|JPEG|*.jpg|GIF|*.gif`

`FilterIndex` – the filter to use as an origin 1 index

File Dialog

FileName – the name of the file selected

ShowDialog – a method to show the dialog and block until cancel or OK is clicked

ToolTips

These are the small pop-up boxes which explain the purpose of a control

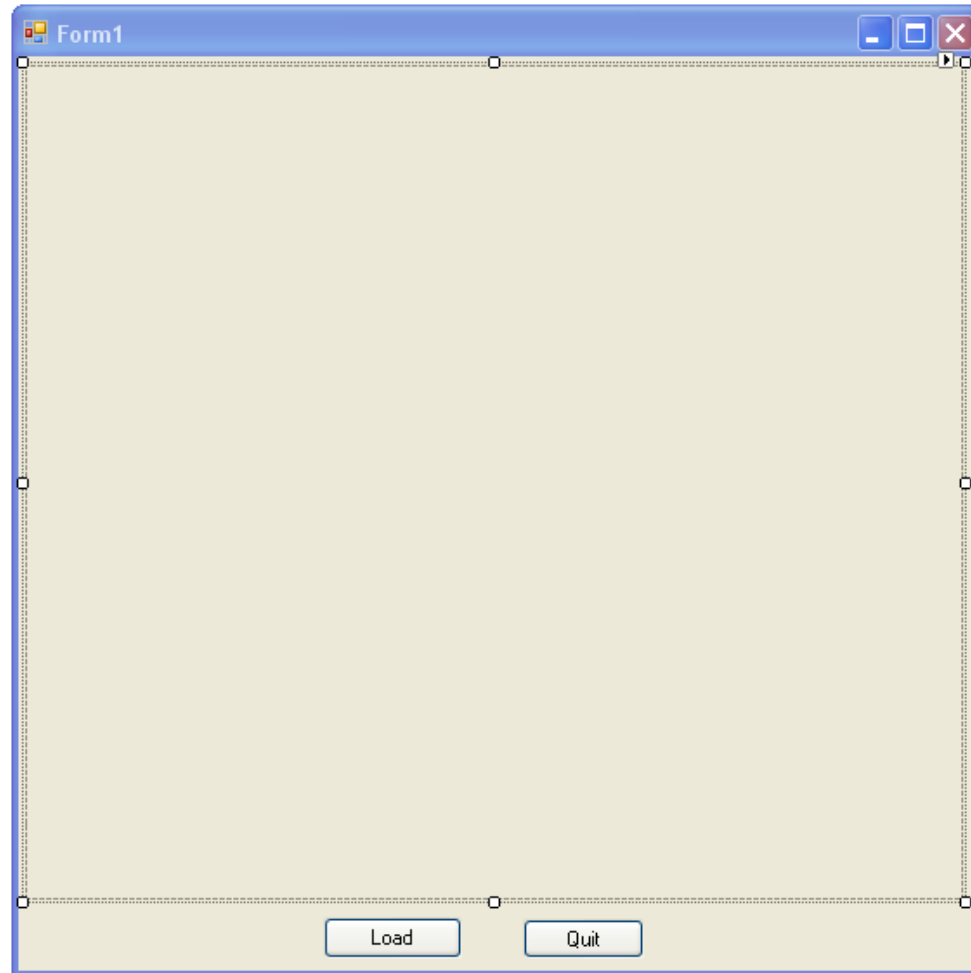
To use


Create a new tooltip in the designer

Drop the tooltip onto the form

The tooltip will appear on a tray below the form

ToolTips



 loadTip

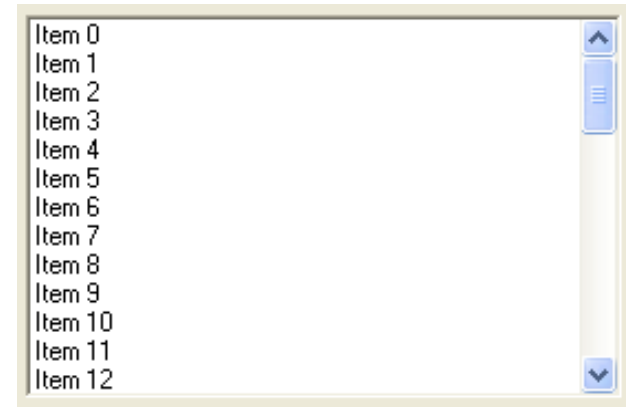
ToolTips

After the tooltip appears in the tray, a new tooltip property appears for every component

This can be assigned different text for each component

That text will be displayed when the mouse hovers over that component

ListBox



The ListBox presents a list of items which can be selected

A scrollbar is displayed if needed

`MultiColumn` – displays list as multiple columns

`SelectedIndex` – index of selected item

`SelectedIndices` – collection of selected indices

`SelectedItem` – the selected item

ListBox

`SelectedItems` – collection of selected items

`SelectionMode` – how items can be selected

- `None` – no selection
- `One` – single selection
- `MultiSimple` – each click selects additional item
- `MultiExtended` – uses shift and control keys

`Sorted` – if true the items will be sorted alphabetically

ListBox

`Items` – a collection of items in the list box

`ClearSelected` – method to clear selection

`GetSelected` – returns true if the parameter passed is **selected**

`SelectedIndexChanged` – **event** when selection **changes**

* see the attached example `ListBoxDemo`

Populating a ListBox

Any object can be placed into a ListBox
The display is generated by ToString()

```
for(int i = 0; i < 50; i++) {  
    listBox1.Items.Add(  
        "Item " + i.ToString() );  
}
```


ComboBox

A combo box is like a list but lets you displays a selected value.

The list pulls down when a selection is being made.

Options allow the selected text to be editable or to require it to be selected from the drop-down list

ComboBox

`DropDownStyle` –

`Simple` – text is editable & list always visible

`DropDown` – default indicating text is editable & user must click to see list

`DropDownList` – value is not editable & user must click to see list

`Items` – the collection of items in the list

ComboBox

`MaxDropDownItems` – max number of items in pulldown before scrollbar used

`SelectedIndex` – index of selection

`SelectedItem` – selected item

`Sorted` – whether entries are sorted

`SelectedIndexChanged` – event raised when selection changes

Menus

Pulldown menus provide a way to select commands and options

Normally these are in a MenuStrip across the top of the application

Begin by placing a MenuStrip across the application

It displays boxes into which menu items and cascading menus can be typed

Menus

The *type here* text allows new items to be added

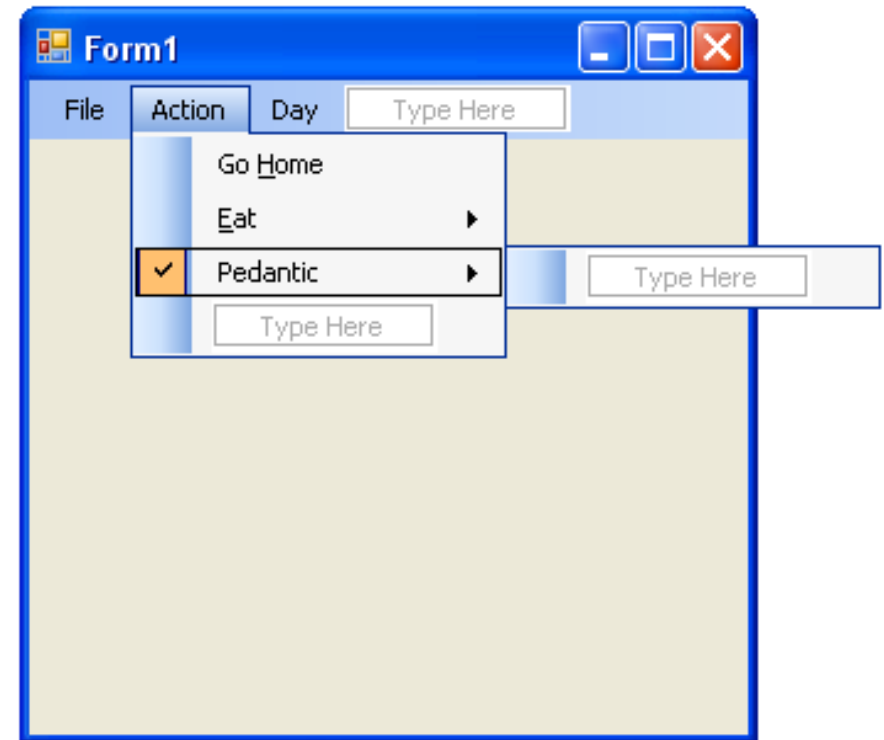
When you click on *type here* a pull down appears letting you select

Menu item

ComboBox

Separator

TextBox



MenuItem Properties

`Checked` – if true displays check mark

`CheckOnClick` – check state changes when clicked

`CheckState` – one of

`CheckState.Checked`

`CheckState.Unchecked`

`CheckState.Indeterminate`

`ShortcutKeys` – a member of the `Shortcut` enumeration indicating the shortcut key

MenuItem

`Click` – event which is raised when the menu item is clicked

Menu items are similar to buttons and are handled in the same way

* see the attached example MenuDemo

ComboBox Menu Items

You can use a `ComboBox` as a menu item

Use the designer to add a set of `Items` to the combo box

You can then select a value

The click event is raised only when you click on the selected value, not when you change the selection

If you have nothing selected, the selected item will be null

Automatic Layout

All of the forms we have seen so far have been laid out based on X-Y coordinates

If the outer form is resized, the contents stay at the same size and position

There are three ways to change this

The Dock property

The FlowLayoutPanel

The TableLayoutPanel

The Dock Property

Every control has a Dock property which can be set to

`None` – no docking

`Left` – docked to left side of container

`Right` – docked to right side of container

`Top` – docked to top of container

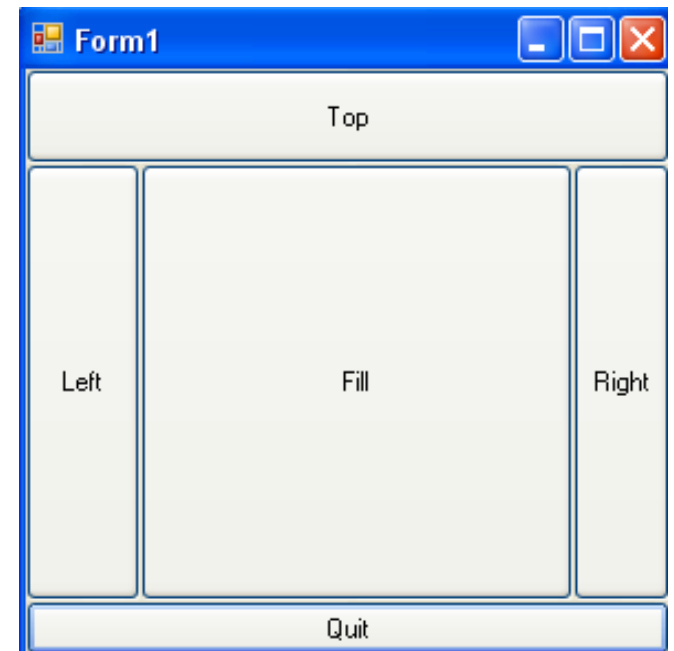
`Bottom` – docked to bottom of container

`Fill` – fills all available space

The Dock Property

When controls are docked to the sides of their containing form, resizing the form resizes the controls inside as well

* see the attached example DockDemo



FlowLayoutPanel

Arranges its contents into horizontal or vertical rows

As form is resized, controls are arranged into new rows or columns

Created in designer by dropping controls into it

In designer it has small arrow on the top that activates menu of editing actions



FlowLayoutPanel

`FlowDirection` – layout direction

`BottomUp`

`TopDown`

`LeftToRight`

`RightToLeft`

`WrapContents` – whether the contents should be wrapped to a new row or column or clipped

* see the attached example `LayoutDemo`

TableLayoutPanel

Arranges the controls within it into rows and columns

Automatically resizes contents when the surrounding form is resized

The pull-out menu in designer can be used to add or delete rows and columns

TableLayoutPanel

`ColumnCount` – get/set number of columns

`RowCount` – get/set number of rows

* see the attached example `LayoutDemo`

MessageBox

A pop-up dialog which displays a message

Use the static Show methods to see one

`DialogResult Show(string)` – box with simple string and OK button

`DialogResult Show(string text, string caption)`

`DialogResult Show(string text, string caption, MessageBoxButtons)`

- OK
- OKCancel
- RetryCancel
- YesNo
- YesNoCancel

DialogResult

Abort

Cancel

Ignore

No

None

OK

Retry

Yes