**Windows Forms Controls v11**

[Most Controls Have the Following Properties 5](#_Toc53918920)

[CausesValidation (EditPerson) 5](#_Toc53918921)

[TabIndex 5](#_Toc53918922)

[Dock 5](#_Toc53918923)

[Background Color 5](#_Toc53918924)

[Foreground Color 5](#_Toc53918925)

[Font 5](#_Toc53918926)

[Background Image 5](#_Toc53918927)

[Opacity 5](#_Toc53918928)

[Tag (EditPerson) 5](#_Toc53918929)

[Modifiers (GuessingGame) 5](#_Toc53918930)

[ContextMenuStrip (MyEditor) 6](#_Toc53918931)

[Event Handlers 6](#_Toc53918932)

[Special Topics 7](#_Toc53918933)

[Scaling Issues 7](#_Toc53918934)

[Mark of the Web Files 7](#_Toc53918935)

[Converting Between an Application (.exe) and a Library (.dll) 8](#_Toc53918936)

[The Windows Forms Controls 8](#_Toc53918937)

[Form 8](#_Toc53918938)

[Important Members 8](#_Toc53918939)

[TextBox (EditPerson) 9](#_Toc53918940)

[Important Fields 9](#_Toc53918941)

[Important Events 9](#_Toc53918942)

[Validating 9](#_Toc53918943)

[TextChanged 10](#_Toc53918944)

[KeyPress 10](#_Toc53918945)

[RichTextBox (MyEditor) 10](#_Toc53918946)

[Important Fields and Methods 10](#_Toc53918947)

[Important Events 11](#_Toc53918948)

[Validating (see TextBox) 11](#_Toc53918949)

[TextChanged (see TextBox) 11](#_Toc53918950)

[KeyPress (see TextBox) 11](#_Toc53918951)

[SelectionChanged 11](#_Toc53918952)

[ComboBox (EditPerson) 11](#_Toc53918953)

[Important Fields 11](#_Toc53918954)

[Important Events 11](#_Toc53918955)

[SelectedIndexChanged 12](#_Toc53918956)

[Label (EditPerson) 12](#_Toc53918957)

[Important Fields 12](#_Toc53918958)

[Button (EditPerson) 12](#_Toc53918959)

[Important Fields 12](#_Toc53918960)

[Important Events 12](#_Toc53918961)

[Click 12](#_Toc53918962)

[ErrorProvider: Error Messaging Control for the Form (EditPerson) 13](#_Toc53918963)

[Important Members 13](#_Toc53918964)

[ListView (CourseApp) 13](#_Toc53918965)

[Important Field and Methods 13](#_Toc53918966)

[Important Events 14](#_Toc53918967)

[KeyDown (can also be added to most other controls such as TextBox and RichTextBox) 14](#_Toc53918968)

[ItemActivate 15](#_Toc53918969)

[SelectedIndexChanged 15](#_Toc53918970)

[CheckedListBox 15](#_Toc53918971)

[ListBox 16](#_Toc53918972)

[RadioButton (EditPerson) 16](#_Toc53918973)

[Important Fields 16](#_Toc53918974)

[Important Events 16](#_Toc53918975)

[CheckedChanged 16](#_Toc53918976)

[GroupBox (EditPerson) 16](#_Toc53918977)

[Important Fields 16](#_Toc53918978)

[Panel 17](#_Toc53918979)

[Important Fields 17](#_Toc53918980)

[TabControl (EditPerson) 17](#_Toc53918981)

[Important Fields 17](#_Toc53918982)

[Important Events 17](#_Toc53918983)

[SelectedIndexChanged 17](#_Toc53918984)

[OpenFileDialog (MyEditor) 18](#_Toc53918985)

[Important Fields and Methods 18](#_Toc53918986)

[SaveFileDialog (MyEditor) 18](#_Toc53918987)

[Important Fields and Methods 18](#_Toc53918988)

[ColorDialog (MyEditor) 18](#_Toc53918989)

[Important Fields and Methods 18](#_Toc53918990)

[Timer (MyEditor) 18](#_Toc53918991)

[Important Fields and Methods 18](#_Toc53918992)

[Important Events 18](#_Toc53918993)

[Tick 18](#_Toc53918994)

[ContextMenuStrip (MyEditor) 19](#_Toc53918995)

[MenuStrip (MyEditor) 19](#_Toc53918996)

[ToolStrip (MyEditor) 19](#_Toc53918997)

[ToolTip (EditPerson) 19](#_Toc53918998)

[Important Fields and Methods 19](#_Toc53918999)

[WebBrowser (EditPerson) 19](#_Toc53919000)

[Registry Setting to Enable Internet Explorer v11 19](#_Toc53919001)

[Important Fields and Methods 20](#_Toc53919002)

[Important Events 20](#_Toc53919003)

[DocumentCompleted 20](#_Toc53919004)

[HtmlElement / HtmlElementCollection / DOM Manipulation (PE-20, EditPerson, Discord, Sherlock) 21](#_Toc53919005)

[PictureBox (EditPerson) 23](#_Toc53919006)

[Important Fields and Methods 23](#_Toc53919007)

[Important Events 23](#_Toc53919008)

[MouseHover 23](#_Toc53919009)

[MouseEnter 23](#_Toc53919010)

[MouseLeave 24](#_Toc53919011)

[ProgressBar (MyEditor) 24](#_Toc53919012)

[Important Fields 24](#_Toc53919013)

[Important Events 24](#_Toc53919014)

[SplitContainer (MyEditor) 25](#_Toc53919015)

[Important Fields 25](#_Toc53919016)

[Important Events 25](#_Toc53919017)

[StatusStrip (MyEditor) 25](#_Toc53919018)

[Important Fields 25](#_Toc53919019)

[Important Events 25](#_Toc53919020)

[CheckBox 25](#_Toc53919021)

[Important Fields 25](#_Toc53919022)

[Important Events 25](#_Toc53919023)

[CheckedChanged (see RadioButton) 25](#_Toc53919024)

[DateTimePicker (EditPerson) 26](#_Toc53919025)

[Important Fields 26](#_Toc53919026)

[Important Events 26](#_Toc53919027)

[ValueChanged 26](#_Toc53919028)

[MonthCalendar 26](#_Toc53919029)

# **Most Controls H**ave the Following Properties

## **CausesVali**dation (EditPerson)

The user tabs from field to field on a form, for example from the "nameText" TextBox to the "emailText" TextBox. If CausesValidation is set to True for the second object (emailText), then the "Validating" event attached to the first object is raised and any handlers attached to that event for the first object are fired. This allows for validating the contents of a field upon exiting the field, and perhaps not allowing the user to tab or click out of the field if there is invalid data.

A common practice is to set CausesValidation to False for the Cancel Button to allow the user to exit the form without enforcing valid data in all of the fields, otherwise the user may be trapped on the form!

## TabIndex

The order in which the objects on the form should be tabbed through.

## Dock

Specifies how the control should be placed in the form. The center rectangle is "Fill" which means to fully fill the form and resize with the form.

## Background Color

The "BackColor" property. Note by choosing the "custom" tab, you can enter custom RGB values.

## Foreground Color

The text and border colors. The "ForeColor" property. Note by choosing the "custom" tab, you can enter custom RGB values.

## Font

The "Font" property. Select style, family, size and decoration

## Background Image

BackgroundImage: choose "Local Resource/Import…" to import downloaded images

BackgroundImageLayout

Tile: if the image is smaller than object, tile to cover full object (cropped if larger than object)

Center: center the image over the object (cropped if larger than object)

Stretch:resize the image to exactly fit in the object

Zoom:resize the image to fit in the object while staying proportional

ImageAlign: a selection box for how the image aligns with the control

## Opacity

How transparent the control is

## Tag (EditPerson)

A System.Object which can be used to persist any application-specific data against the Windows Form object. For example it could be used to store the valid state of an object or the index key of a SortedList<> item in a ListView.

## Modifiers (GuessingGame)

The controls on the form default to being private to the form's class. The modifiers property allows you to set the control to be public so that the control's value and settings can be accessed from outside of the class. For example, if you have a main form which creates a child form and you want the child form to access a TextBox or enable or disable a Button on the main form, you could set the controls to have public accessibility and use the following code:

Main Form code:

// pass a reference to MainForm in the ChildForm constructor

ChildForm childForm = new ChildForm(this);

class ChildForm

{

MainForm mainForm;

public ChildForm( MainForm mainForm )

{

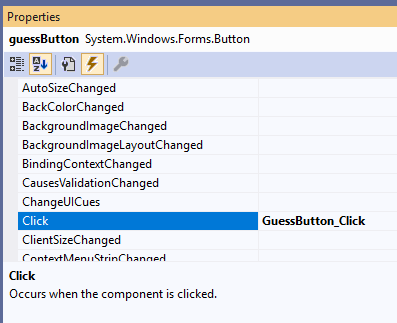
// set class-scoped field to allow all methods in ChildForm class to access MainForm

this.mainForm = mainForm;

// change the contents of Main Form's TextBox

mainForm.textBox.Text = "let's change the text";

// disable the Start Button on Main Form

mainForm.startButton.Enabled = false;

}

}

## ContextMenuStrip (MyEditor)

The shortcut menu to display when the user right-clicks the control.

## Event Handlers

The Event Handlers Added via the Designer Can be Viewed via the Properties Window by Selecting the Lightning Bolt Toolbar Button.

This can be confusing as they are hidden from view, and erroneous clicking on controls on the form can mistakenly add event handlers.

Visual Studio's convention is to add the event handler as ControlName\_EventName, therefore it is recommended that you use 2 underscores between the ControlName and the EventName when you manually add the event handler method and add the delegate method to the event listener. Using 2 underscores will make it obvious which event handlers were added via the designer and which were manually added.

For example:

// add the Validating event listener

this.nameText.Validating += new CancelEventHandler(this.TxtBoxEmpty\_\_Validating);

// add the Validating event handler

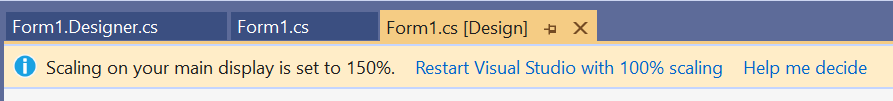
private void TxtBoxEmpty\_\_Validating(object sender, CancelEventArgs e) { }

Note that if an event handler is listed in the Properties window, then removing the event handler from the source code without first clearing the associated Properties field will break the form and not allow you to continue editing the form.

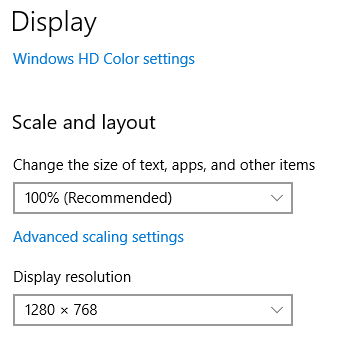
# Special Topics

## Scaling Issues

If your display settings are not set to 100% scaling, then the form designer tries to scale the form which results in (x,y) coordinates and control sizes not matching between developers. The designer will warn you that the scaling is not set to 100%:



If this happens, you should go to Windows/Settings/Display Settings, and set the scaling to 100% as follows:



If you have a very high resolution display, then these settings will not be ideal for your normal computer use. You will want to make these changes only while using Visual Studio .NET.

If these changes do not fix the scaling issues, then you have to add a Windows Registry setting to mark Visual Studio as DPI-unaware. Open Registry Editor and add an entry to the HKEY\_CURRENT\_USER\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ AppCompatFlags\ Layers key:

A new String value:

Name = C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\IDE\devenv.exe

Note if you're using the Professional or Enterprise edition of Visual Studio, replace Community with Professional or Enterprise in the entry. Also replace the drive letter as necessary.

Type: REG\_SZ (String value)  
Value: DPIUNAWARE

With this registry setting, you can permanently use your previous display settings (ie. you will no longer need to switch back and forth).

## Mark of the Web Files

When using downloaded files in a project, you may see the following error:

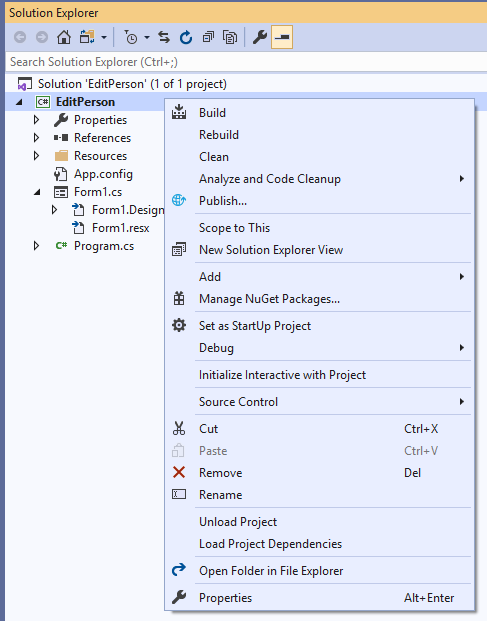
https://media.discordapp.net/attachments/666246274692939786/683890475974787101/9pW1r.png

This is a new Windows security enhancement where files downloaded from the internet result in a "blocked" attribute being set if you add them to a Visual Studio project.

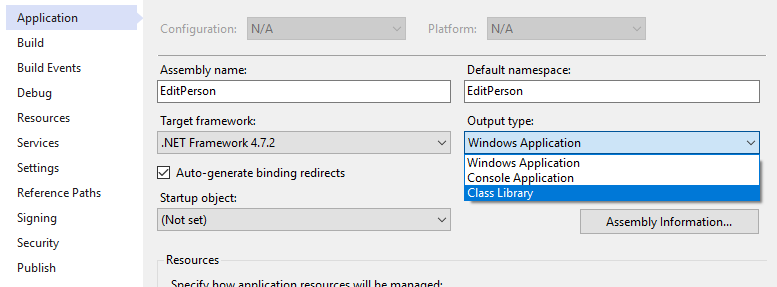
The solution seems to be:

1. Open the Windows file explorer. Navigate to the project/solution directory
2. Search for \*.resx. --> You will get the list of resx files
3. Right click the resx file, open the Properties and check the option 'Unblock'
4. Repeat #3 for each resx file.
5. Reload the project.​

## Converting Between an Application (.exe) and a Library (.dll)

 Right-click on the project name in the Solution Explorer and select Properties.

On the "Application" tab, you can switch the "Output type" between Windows Application (which creates an exe) and Class Library (which creates a dll). Build the project after switching the Output type.



# The Windows Forms Controls

# Form

### Important Members

(**Name**): the name of the Form Class

**FormBorderStyle**: Fixed3D

**MaximizeBox**: False (don't show the Maximize control)

**MaximizeSize**: set to Size

**MinimizeBox**: False (don't show the Minimize control)

**MinimizeSize**: Set to Size

**ShowInTaskbar**: False (only show in Taskbar if it's the "parent" form of the application)

**Size**: width, height

**Text**: Form's title at runtime

**AcceptButton**: the button to activate when Enter is pressed on the Form

**CancelButton**: the button to activate when Escape is pressed on the Form

**AutoScroll**: boolean for whether to allow the form to scroll if there are controls which flow offscreen

**Owner**: the parent Form containing this Form (if any)

**CenterToParent**(): center this Form within the **Owner** Form

**InitializeComponent**(): must be the first command in the Form's constructor (creates all designed objects on the Form)

**Show**(): display and activate the Form as a non-modal form (any other enabled application forms are still enabled)

**ShowDialog**(): display and activate the Form as a modal form (a modal form means that there is only 1 active form for the application. Any other open forms are disabled.)

There are 3 ways to display and activate a Form:

1. Application.Run(new EditPersonForm());

In the application's Main() method, we usually use Application.Run(new EditPersonForm()); to create the form and link it to the Windows Messaging Framework, so that your Windows Application interacts with the Operating System and any other running applications (allowing Copy/Paste etc).

1. If the Show() or ShowDialog() command is the last line of the Form constructor, then simply creating the form object will display and activate the form

EditPersonForm editPersonForm = new EditPersonForm();

1. We can create the form and call its Show() or ShowDialog() method

EditPersonForm editPersonForm = new EditPersonForm();

editPersonForm.Show();

**Close**(): closes the Form and releases member controls

**Dispose**(): releases the Form from memory

# TextBox (EditPerson)

### Important Fields

**(Name)**: the fieldname within the Form Class

**Location**: x, y

**Size**: width, height

**TabIndex**: the order of the field when tabbing through the form

**Text**: the data entered into the control at runtime

**Visible**: boolean to control visibility

**Enabled**: boolean to control whether the TextBox is enabled for entry

**Tag**: System.Object which can be used to store valid state of the TextBox **Text** contents

### Important Events

Validating

Occurs when the object is leaving scope

Accepts the event handler **CancelEventHandler**() because the primary purpose of Validating is to determine whether this event should cause the focus to leave the current control and enter the next control (if CausesValidation == True for the next control). If the validation fails, then the navigation event is cancelled and the current control stays in focus.

Example for adding the delegate method:

this.objectName.Validating += new CancelEventHandler(this.ObjectName\_\_Validating);

The **CancelEventHandler**() delegate method must accept the following parameters:

private void **ObjectName\_\_Validating(object sender, CancelEventArgs e)**

By convention, the delegate methods are named:

ObjectName\_EventName()

Important Fields in sender (TextBox)

TextBox tb = (TextBox)sender;

**tb.Text**: validate for certain contents before allowing user to exit the field

Important Fields in CancelEventArgs

**e.Cancel**: a boolean to indicate whether the event should be cancelled or not. If set to true, then the current TextBox will stay in focus.

#### TextChanged

Occurs when the contents of Text changes

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.TextChanged += new EventHandler(this.ObjectName\_\_TextChanged);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_TextChanged(object sender, EventArgs e)**

Important Fields in sender

TextBox tb = (TextBox)sender;

**tb.Text:** the current text in the TextBox

Important Fields in EventArgs

None.

#### KeyPress

Occurs when the user presses a key sequence which generates a character (shift+A for example) within the control

Accepts the **KeyEventHandler**() delegate, whose method must have the following signature:

private void **ObjectName\_\_KeyPress(object sender, KeyPressEventArgs e)**

Example for adding the delegate method:

this.objectName.KeyPress += new KeyPressEventHandler(this.ObjectName\_\_KeyPress);

Important Fields in sender

TextBox tb = (TextBox)sender;

**tb.Text:** the current text in the TextBox

Important Fields in KeyPressEventArgs

**e.KeyChar**: gets or sets the character just pressed allowing you to change, suppress or pass-through each character

**e.Handled**: a boolean to indicate whether the delegate's method "handled" the keypress. If it is set to true, then .NET will not process the keypress (ie. the keyboard buffer will be cleared).

**KeyDown (see ListView)**

# RichTextBox (MyEditor)

### Important Fields and Methods

**(Name)**: the fieldname within the Form Class

**Location**: x, y

**Size**: width, height

**TabIndex**: the order of the field when tabbing through the form

**Text**: the data entered into the control at runtime

**Visible**: boolean to control visibility

**Enabled**: boolean to control whether the TextBox is enabled for entry

**Tag**: System.Object which can be used to store valid state of the TextBox **Text** contents

**Clear()**: clear all current text

**Cut():** cut the selected text into the Windows clipboard

**Copy():** copy the selected text into the Windows clipboard

**Paste():** paste the Windows clipboard at the cursor position

**LoadFile(string filename, RichTextBoxStreamType type):** loads the contents of a file into Text (MyEditor code example)

**SaveFile(string filename, RichTextBoxStreamType type):** saves the Text to a file (MyEditor code example)

**SelectionFont**: get or set the Font of the selected text (MyEditor example code)

**SelectionColor**: get or set the Color of the selected text (MyEditor example code)

**Select(int start, int length):** selects the text from start offset for length characters

### Important Events

Validating (see TextBox)

#### TextChanged (see TextBox)

#### KeyPress (see TextBox)

**KeyDown (see ListView)**

#### SelectionChanged

Occurs when the currently selected Text changes (ie. cursor movement or highlighting text)

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.SelectionChanged += new EventHandler(this.ObjectName\_\_SelectionChanged);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_SelectionChanged(object sender, EventArgs e)**

Important Fields in sender

RichTextBox rtb = (TextBox)sender;

**rtb.Text:** the current text in the TextBox

Important Fields in EventArgs

None.

# ComboBox (EditPerson)

### Important Fields

(**Name**): the field name within the Form Class

**DropDownStyle**: DropDownList

**Items**: string[] to display in the list to choose from. Can be programmatically set using Items.Add()

**Location**: x, y

**MaxDropDownItems**: max number of items to show in list

**Size**: width, height

**TabIndex**: the order of the field when tabbing through the form

**Visible**: boolean to control visibility

**Enabled**: boolean to control whether the ComboBox is enabled for entry

**SelectedIndex**: the 0-based Items array el currently selected

**Tag**: System.Object which can be used to store application-related data

### Important Events

#### SelectedIndexChanged

Occurs when the user changes the ComboBox value

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.SelectedIndexChanged +=

new EventHandler(this.ObjectName\_\_SelectedIndexChanged);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_SelectedIndexChanged(object sender, EventArgs e)**

Important Fields in sender

ComboBox cb = (ComboBox)sender;

**cb.SelectedIndex**: the 0-based index of the selected item

**cb.SelectedItem**: the string of the display value of the selected item

Important Fields in EventArgs

None.

# Label (EditPerson)

### Important Fields

(**Name**): the field name within the Form Class

**AutoSize**: False (if True, then the designer will size the field according to the length of Text - not always desirable when aligning fields or supporting multiple languages)

**Location**: x, y

**Size**: width, height

**Text**: text to display in the label

**Visible**: boolean to control visibility

**Tag**: System.Object which can be used to store application-related data

# Button (EditPerson)

### Important Fields

(**Name**): the control name within the Form Class

**CausesValidation**: True (set to False for the Cancel button)

**Location**: x, y

**Size**: (width, height) (56, 24 is a good size for OK/Cancel)

**TabIndex**: the order of the control when tabbing through the form

**Text**: text to show on the button

**Visible**: boolean to control visibility

**Enabled**: boolean to control whether the Button is enabled for clicking on

**Tag**: System.Object which can be used to store application-related data

### Important Events

#### Click

Occurs when the user clicks the Button. The **Click** event handler can be added by double-clicking on the button in the designer.

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.Click +=

new EventHandler(this.ObjectName\_\_Click);

The EventHandler delegate method must have the following signature:

**private void ObjectName\_\_Click(object sender, EventArgs e)**

Important Fields in sender

Button b = (Button)sender;

b.Enabled: you may want to disable the button after it was clicked

Important Fields in EventArgs

None.

# ErrorProvider: Error Messaging Control for the Form (EditPerson)

### Important Members

SetError( object obj, string errorMsg ); // displays an error icon and tooltip beside object

obj is the Form object to display the error beside

set errorMsg = null or "" to clear the error flag

# ListView (CourseApp)

### Important Field and Methods

(**Name**): the object name within the Form Class

**Enabled**: boolean to control whether the ListView is enabled for clicking on

**FullRowSelect**: Normally only the first column is highlighted when a row is selected. True = select the whole row.

**GridLines**: boolean for whether to display grid lines

**Location**: x, y

**MultiSelect**: boolean for whether multiple items can be selected

**Size**: width, height

**TabIndex**: the order of the control when tabbing through the form

**Tag**: System.Object which can be used to store application-related data

**View**: selects 1 of 5 different views the items can be shown in

**VirtualMode**: boolean for whether the ListView should load 1 screen of data at a time

**VirtualListSize**: how many entries per screen in VirtualMode

**Visible**: boolean to control visibility

**CheckBoxes**: boolean to control whether a checkbox is added at the beginning of each row

**CheckedIndices**: the array of numeric indexes which are currently checked

**CheckedItems**: the array of **ListViewItem** which are currently checked

**Columns**: ColumnHeader[] (Click […])

Each **ColumnHeader**

(**Name**): the object name within the ListView object

**DisplayIndex**: the 0-based numeric order of the column

**Tag**: System.Object which can be used to store application-related data

**Text**: the text to display as the title of the column

**TextAlign**: how to align the text in the column (first column is **always** left-aligned and cannot be changed)

**Width**: width of the column

**BeginUpdate()**: locks the ListView to begin updating it

**EndUpdate**(): unlocks the ListView at end of updating

**Focused**(): sets the control as current active control

**PaintListView**(): a method you need to write to populate the ListView from your data (this method can have any name you wish, and you may have multiple methods to do the job, depending on data complexity)

**Items**: an array of ListViewItem

**Clear**(): clears the items

Each **ListViewItem**

**BackColor**: the background color of this item (useful for making alternating colors in the list for easier reading)

**Tag**: System.Object which can be used to store application-related data

**Text**: the text to show in the first column of the ListView

**Checked**: boolean for whether the checkbox is checked for this row

**Focused**: boolean for whether to this row should be in focus

**Selected:** boolean for whether the row is selected

**SubItems**: an array of ListViewItem.ListViewSubItem to hold the remainder of the column data

Each **ListViewItem.ListViewSubItem**

**Text**: the text to show in this column

**Tag**: System.Object which can be used to store application-related data

**TopItem**: the **ListViewItem** to show at the top of the list

### Important Events

#### KeyDown (can also be added to most other controls such as TextBox and RichTextBox)

Occurs for any key pressed on the keyboard including Shift, Ctrl, etc. So Shift+A will generate 2 events. KeyDown is overly complicated to check for letter keys, especially if you need to check the case, but it is useful for catching if Enter is pressed and you can also quite easily check for digits or backspace with the following code:

// assume that the keystroke is invalid

e. SuppressKeyPress = true;

// if a digit is pressed, allow .NET to process it

// checks both the number pad and the number keys

if ((e.KeyCode >= Keys.NumPad0 && e.KeyCode <= Keys.NumPad9) ||

(e.KeyCode >= Keys.D0 && e.KeyCode <= Keys.D9))

{

e. SuppressKeyPress = false;

}

// allow .NET to handle backspace

if (e.KeyCode == Keys.Back)

{

e. SuppressKeyPress = false;

}

If you need to check for both alphabetic characters and Enter, then you should use KeyPress and validate for e.KeyChar == '\r' for Enter.

Accepts the **KeyEventHandler**() delegate, whose method must have the following signature:

private void **ObjectName\_\_KeyDown(object sender, KeyEventArgs e)**

Example for adding the delegate method:

this.objectName.KeyDown +=

new KeyEventHandler(this.ObjectName\_\_KeyDown);

Important Fields in sender (the ListView object)

ListView lv = (ListView)sender;

**lv.SelectedItems.Count**: how many rows in the list are selected

**lv.SelectedItems**: the array of currently selected **LIstViewItem**. Normally you are checking if they pressed Enter on a row in the ListView and you only want to process the one row which was selected (**SelectedItems**[0])

Important Fields in KeyEventArgs

**e.KeyCode**: gets or sets the character just pressed allowing you to change, suppress or pass-through each character

e.KeyCode == Keys.Enter

**e.SuppressKeyPress**: a boolean to indicate whether the control should suppress the keypress. If it is set to true, then .NET will not process the keypress (ie. the keyboard buffer will be cleared). Note that this is different from the KeyPress event handling which uses e.Handled.

#### ItemActivate

Occurs when a row is double-clicked on

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.ItemActivate +=

new EventHandler(this.ObjectName\_\_ItemActivate);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_ItemActivate(object sender, EventArgs e)**

Important Fields in sender

ListView lv = (ListView)sender;

**lv.SelectedItems**: the array of currently selected **LIstViewItem**. We will usually only want SelectedItems[0]

Important Fields in EventArgs

None.

#### SelectedIndexChanged

Occurs when a new row is selected either via a single mouse click or the arrow keys. It will be called twice (when the currently selected row is deselected and when the new row is selected)

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.SelectedIndexChanged +=

new EventHandler(this.ObjectName\_\_SelectedIndexChanged);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_ SelectedIndexChanged (object sender, EventArgs e)**

Important Fields in sender

ListView lv = (ListView)sender;

**lv.SelectedItems**: the array of currently selected **LIstViewItem**. We will usually only want SelectedItems[0]

Important Fields in EventArgs

None.

# CheckedListBox

The ListView control with CheckBoxes = true gives the same functionality as this control.

# ListBox

The ListView control with one column gives the same functionality as this control.

# RadioButton (EditPerson)

Only one RadioButton can be selected on a Form unless "containers" such as the GroupBox, Panel, TabControl or SplitContainer are used. If those other containers are used, then one RadioButton can be selected per container on the form.

### Important Fields

**(Name)**: the field name within the Form Class

**Checked**: boolean to indicate if the radio button is checked or not. Your code should default a selection if the associated data does not exist yet.

**Location**: x, y

**Size**: width, height

**TabIndex**: the order of the field when tabbing through the form

**Text**: the text to show beside the radio button

**Visible**: boolean to control visibility

**Enabled**: boolean to control whether the RadioButton is enabled for entry

**Tag**: System.Object which can be used to store application-specific data

### Important Events

#### CheckedChanged

Occurs when the Checked status changes. Note that this will be called when the currently selected RadioButton unchecks and the new RadioButton becomes checked.

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.CheckedChanged +=

new EventHandler(this.ObjectName\_\_CheckedChanged);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_CheckedChanged(object sender, EventArgs e)**

Important Fields in sender

RadioButton rb = (RadioButton)sender;

**rb.Checked**: the current checked state of the RadioButton control.

Important Fields in EventArgs

None.

# GroupBox (EditPerson)

The GroupBox allows you to group controls together and enable or make visible the whole group of them just by enabling or making visible the group box. It also allows you to make allow multiple RadioButtons to be selected on the form.

### Important Fields

**(Name)**: the control name within the Form Class

**Location**: x, y

**Size**: width, height

**TabIndex**: the order of the group of controls when tabbing through the form. Note that the contained controls then have their own TabIndex order within the GroupBox.

**Text**: the text to show on the GroupBox border

**Visible**: boolean to control visibility of the GroupBox and all contained controls

**Enabled**: boolean to control whether all contained controls are enabled

**Tag**: System.Object which can be used to store application-specific data

# Panel

The Panel is identical to the GroupBox but does not display a title in the border.

### Important Fields

**(Name)**: the control name within the Form Class

**Location**: x, y

**Size**: width, height

**TabIndex**: the order of the group of controls when tabbing through the form. Note that the contained controls then have their own TabIndex order within the Panel.

**Visible**: boolean to control visibility of the Panel and all contained controls

**Enabled**: boolean to control whether all contained controls are enabled

**Tag**: System.Object which can be used to store application-specific data

# TabControl (EditPerson)

### Important Fields

(**Name**): the field name within the Form Class

**Dock:** normally you want the TabControl to **fill** the form (the middle rectangle)

**Items**: string[] to display in the list to choose from. Can be programmatically set using Items.Add()

**Location**: x, y

**Size**: width, height

**TabIndex**: the order of the field when tabbing through the form

**TabPages**: the collection of TabPages that comprise the TabControl (click […])

(Name): the name of the TabPage object

Text: the text to display on the tab

**Visible**: boolean to control visibility

**Enabled**: boolean to control whether the ComboBox is enabled for entry

**Tag**: System.Object which can be used to store application-related data

### Important Events

#### SelectedIndexChanged

Occurs when the user changes the TabPage value

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.SelectedIndexChanged +=

new EventHandler(this.ObjectName\_\_SelectedIndexChanged);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_SelectedIndexChanged(object sender, EventArgs e)**

Important Fields in sender

TabControl tc = (TabControl)sender;

**tc.SelectedIndex**: the 0-based index of the selected item

**tc.SelectedTab**: the selected TabPage (Name)

Important Fields in EventArgs

None.

# OpenFileDialog (MyEditor)

### Important Fields and Methods

**(Name)**: the control name within the Form Class

**DefaultExt:** the default extension of the filename to open

**FileName:** the selected filename and full path

**Title:** the title to show in the Open File Dialog window

**Tag**: System.Object which can be used to store application-specific data

ShowDialog(): displays the interactive File Open Dialog

if (openFileDialog.ShowDialog() == DialogResult.OK)

{ filePath = openFileDialog.FileName; }

# SaveFileDialog (MyEditor)

### Important Fields and Methods

**(Name)**: the control name within the Form Class

**DefaultExt:** the default extension of the filename to save

**FileName:** the selected filename and full path

**Title:** the title to show in the Open File Dialog window

**Tag**: System.Object which can be used to store application-specific data

**ShowDialog():** displays the interactive File Open Dialog

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{ filePath = saveFileDialog.FileName; }

# ColorDialog (MyEditor)

### Important Fields and Methods

**(Name)**: the control name within the Form Class

**Color:** the selected color

**Tag**: System.Object which can be used to store application-specific data

**ShowDialog():** displays the interactive File Open Dialog

if (colorDialog.ShowDialog() == DialogResult.OK)

{ textbox.ForeColor = colorDialog.Color; }

# Timer (MyEditor)

### Important Fields and Methods

**(Name)**: the control name within the Form Class

**Interval:** the number of milliseconds to elapse

**Tag**: System.Object which can be used to store application-specific data

**Start():** start the timer

**Stop():** stop the timer

### Important Events

#### Tick

Occurs when the timer's Interval millisecond value has elapsed. By default the timer restarts after firing each Tick event.

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.Tick +=

new EventHandler(this.ObjectName\_\_Tick);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_Tick(object sender, EventArgs e)**

Important Fields in sender

None.

Normally you would be checking other fields, such as whether a ProgressBar has reached 0.

Important Fields in EventArgs

None.

# ContextMenuStrip (MyEditor)

# MenuStrip (MyEditor)

# ToolStrip (MyEditor)

# ToolTip (EditPerson)

### Important Fields and Methods

**(Name)**: the control name within the Form Class

**Tag**: System.Object which can be used to store application-specific data

**Show(string text, object control, int x, int y [, int duration]):** displays the tooltip in the control at (x,y) for duration ms. If duration is omitted, then the tooltip persists until another tooltip.Show() is called, or tooltip.Hide() is called.

**Hide():** hides the tooltip

# WebBrowser (EditPerson)

Displays a web page or calls a URL to accomplish a task (such as logging in to a server)

## Registry Setting to Enable Internet Explorer v12

Note that .NET defaults to using Internet Explorer v7 (released in 2006). Include the following code in the application form's constructor for .NET to use Internet Explorer v12. Add this code directly after InitializeComponent();

try

{

// Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; WOW64; Trident / 7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729; wbx 1.0.0)

Microsoft.Win32.RegistryKey key = Microsoft.Win32.Registry.LocalMachine.OpenSubKey(

@"SOFTWARE\\WOW6432Node\\Microsoft\\Internet Explorer\\MAIN\\FeatureControl\\FEATURE\_BROWSER\_EMULATION",

true);

key.SetValue(Application.ExecutablePath.Replace(Application.StartupPath+"\\",""), 12001, Microsoft.Win32.RegistryValueKind.DWord);

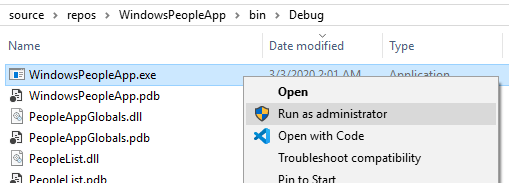
key.Close();

}

catch

{

}

After compiling with this code, you will have to run the executable as an Administrator to add the required registry setting. The most efficient way to do this is to go to the bin\debug folder that contains the exe file for the application. For example, you may have added this code to EditPerson.dll, but WindowsPeopleApp.exe is the executable that calls EditPerson.dll, so right-click on WindowsPeopleApp.exe to **Run as administrator.**

### Important Fields and Methods

**(Name)**: the control name within the Form Class

**Location**: x, y

**Size**: width, height

**Url**: the Uri of the webpage (use Navigate() instead)

this.scheduleWebBrowser.Url = new Uri("c:\\temp\\schedule.html");

**Visible**: boolean to control visibility of the Panel and all contained controls

**Enabled**: boolean to control whether all contained controls are enabled

**ScriptErrorsSuppressed**: boolean to control whether script or security errors or warnings are displayed. Default it false. Normally you will want to set to true to avoid popup security dialog boxes.

**Tag**: System.Object which can be used to store application-specific data

**Navigate**(): a more reliable way to load a website

this.homepageWebBrowser.Navigate(this.homepageTextBox.Text);

**Refresh**(): refresh the current webpage (any DOM modifications will be lost)

You can also add the html dynamically at runtime using the Document.Write() method (GifFinder):

**Document.Write(string html)**: dynamically write html to the web Document

**Refresh**(): be sure to call Refresh after changing the html

### Important Events

#### DocumentCompleted

Occurs when the web control completes loading the web page into memory. This method allows manipulating the DOM of a webpage to change contents and handle events.

Accepts the **WebBrowserDocumentCompletedEventHandler**() delegate.

Example for adding the delegate method:

this.objectName.DocumentCompleted +=

new WebBrowserDocumentCompletedEventHandler(

this.ObjectName\_\_DocumentCompleted);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_DocumentCompleted(object sender,**

**WebBrowserDocumentCompletedEventArgs e)**

Important Fields in sender

WebBrowser wb = (WebBrowser)sender;

// get a single element using the id attribute

HtmlElement htmlElement = wb.Document.GetElementById(htmlId);

// get a collection (array) of elements using the tag

HtmlElementCollection htmlElementCollection =

wb.Document.GetElementsByTagName("a");

Important Fields in EventArgs

e.Url: the URL of the website which has been loaded.

# HtmlElement / HtmlElementCollection / DOM Manipulation (PE-20, EditPerson, Discord, Sherlock)

HtmlElement can reference each element on a web page and can be accessed like a control to add event handlers and change the contents.

**Code Examples**

**Add a click event to #link1**

htmlElement = webBrowser1.Document.GetElementById("link1");

htmlElement.Click += new HtmlElementEventHandler(LinkHtmlElement\_\_Click);

**Change all text to a shade of blue**

htmlElement = webBrowser1.Document.Body;

htmlElement.Style += "color: #0000a4;";

**Append a new footer element to the end of the page**

htmlElement = webBrowser1.Document.CreateElement("footer");

htmlElement.InnerText = "This is my newly created footer!";

htmlElement.SetAttribute("id", "footer");

htmlElement.SetAttribute("class", "footerClass");

htmlElement.SetAttribute("title", "Footer is Here");

webBrowser1.Document.Body.AppendChild(htmlElement);

**Replace the HTML of an element and remove the click event delegate**

// Completely replace the HTML of an element with OuterHtml

htmlElement.OuterHtml = "<p>Stop clicking me!</p>";

htmlElement.Click -= LinkHtmlElement\_Click;

**Insert a new anchor element inside a new paragraph**

htmlElement = webBrowser.Document.CreateElement("p");

htmlElement.InnerHtml = "<a href=\"http://www.rit.edu\">RIT</a>";

**Insert a new anchor element relative to #footer**

htmlElement = webBrowser1.Document.CreateElement("a");

htmlElement.InnerText = "RIT";

htmlElement.SetAttribute("href", "http://www.rit.edu");

HtmlElement footerElement = webBrowser1.Document.GetElementById("footer");

// possible options are:

// HtmlElementInsertionOrientation.**BeforeBegin**: insert new element before this element

// HtmlElementInsertionOrientation.**AfterBegin**: insert new element within and at the start of this element

// HtmlElementInsertionOrientation.**BeforeEnd**: insert new element within and at the end of this element

// HtmlElementInsertionOrientation.**AfterEnd**: insert new element after this element

footerElement.InsertAdjacentElement(HtmlElementInsertionOrientation.AfterEnd, htmlElement);

**Get an array of elements by the tag definition**

// get all paragraph elements

HtmlElementCollection htmlElementCollection = webBrowser.Document.GetElementsByTagName("p");

foreach( HtmlElement htmlElement1 in htmlElementCollection )

{

// set the text inside <p></p> to "blah blah blah"

htmlElement1.InnerText = "blah blah blah";

}

**Add MouseDown and MouseOver (hover) Event Handlers to an Element**

MouseDown with the left button will display a context menu.

MouseOver will display a tooltip with Course Description and Review.

// set the inner text to the course code

htmlElement.InnerText = course.courseCode;

// set the button to red to show the time is scheduled

htmlElement.Style += "background-color:red;";

// add a MouseDown (click) event handler to the html button

htmlElement.MouseDown += new HtmlElementEventHandler(this.HtmlElement\_\_MouseDown);

// add a MouseOver (hover) event handler to the html button

htmlElement.MouseOver += new HtmlElementEventHandler(this.HtmlElement\_\_MouseOver);

private void HtmlElement\_\_MouseDown( object sender, HtmlElementEventArgs e)

{

// MouseDown can check which mouse button was clicked.

// The "Clicked" event is limied to the left mose button.

// if the left mouse button was pressed

if( e.MouseButtonsPressed == MouseButtons.Left)

{

// set class-scoped variable htmlElement to reference the element that was clicked

// so that it can be modified

this.htmlElement = (HtmlElement)sender;

// show the context menu

this.scheduleContextMenuStrip.Show(this.scheduleWebBrowser,

e.MousePosition.X + 5, e.MousePosition.Y + 15);

}

}

private void HtmlElement\_\_MouseOver(object sender, HtmlElementEventArgs e)

{

// show the tooltip when hovering over a button

Course course;

this.htmlElement = (HtmlElement)sender;

// load the course associated with the course code in this button

course = Globals.courses[this.htmlElement.InnerHtml];

if( course != null)

{

// show the tooltip

this.toolTip.Show($"Description: {course.description}\nReview: {course.review}",

this.scheduleWebBrowser, e.MousePosition.X + 5, e.MousePosition.Y + 15, 1500);

}

}

# PictureBox (EditPerson)

The PictureBox displays an image from the hard drive or a URL, or an image can be imported into the application.

### Important Fields and Methods

**(Name)**: the control name within the Form Class

**Image:** an image can be imported into the application using the […] button

**ImageLocation:** this can be set to a URL or filepath to load an image at runtime

**Location:** x, y

**Size**: width, height

**SizeMode**: how to fit the image into the control

**Normal:** display image at actual size in the control, cropping if necessary

**StretchImage:** fix the image exactly into the height and width of the control

**AutoSize:** autosize the control to be the exact size of the image

**CenterImage:** center the image at actual size in the control, cropping if necessary

**Zoom:** zoom the image to fit in the control, maintaining height-width ratio

**Visible**: boolean to control visibility of the GroupBox and all contained controls

**Enabled**: boolean to control whether all contained controls are enabled

**Tag**: System.Object which can be used to store application-specific data

**BringToFront**(): bring the control to the front and overlay any other controls

**SendToBack**(): send the control to the back of any overlaying controls

### Important Events

Click, MouseHover, MouseEnter and MouseLeave are useful events for PictureBoxes

#### MouseHover

Occurs when the mouse pauses over a control.

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.MouseHover +=

new EventHandler(this.ObjectName\_\_MouseHover);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_MouseHover(object sender, EventArgs e)**

Important Fields in sender

If you want to display a ToolTip associated with the control that is hovered over:

toolTip.Show("tooltip text", sender);

Important Fields in EventArgs

None.

#### MouseEnter

Occurs when the mouse enters the region of a control.

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.MouseEnter +=

new EventHandler(this.ObjectName\_\_MouseEnter);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_MouseEnter(object sender, EventArgs e)**

Important Fields in sender

If you want to double the size of the image when the mouse moves over it:

PictureBox pb = (PictureBox) sender;

pb.Height \*= 2;

pb.Width \*= 2;

Important Fields in EventArgs

None.

#### MouseLeave

Occurs when the mouse leaves the region of a control.

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.MouseLeave +=

new EventHandler(this.ObjectName\_\_MouseLeave);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_MouseLeave(object sender, EventArgs e)**

Important Fields in sender

If you want to halve the size of the image when the mouse leaves it:

PictureBox pb = (PictureBox) sender;

pb.Height /= 2;

pb.Width /= 2;

Important Fields in EventArgs

None.

# ProgressBar (MyEditor)

The ProgressBar allows you to display a scrolling bar to indicate progress or a time limit.

### Important Fields

**(Name)**: the control name within the Form Class

**Maximum:** the max value

**Minimum:** the minimum value (usually 0)

**Value:** the current value to display

**Style:** How the progress bar is animated: Continuous/Blocks/Marquee

**Size**: width, height

**Visible**: boolean to control visibility of the GroupBox and all contained controls

**Enabled**: boolean to control whether all contained controls are enabled

**Tag**: System.Object which can be used to store application-specific data

### Important Events

**None**

# SplitContainer (MyEditor)

The SplitContainer allows you to group controls together and enable or make visible the whole group of them just by enabling or making visible the group box. It also allows you to make allow multiple RadioButtons to be selected on the form. Part of the container can be made to be a fixed size, which is useful when you want to doc a RichTextBox or WebBrowser control to fill the whole form and include a ToolStrip.

### Important Fields

**(Name)**: the control name within the Form Class

**Location**: x, y

**Size**: width, height

**Dock**: specify how the control should be placed in the form. The center rectangle is "Fill" which means to fully fill the form and resize with the form.

**IsSplitterFixed:** determines whether the splitter can be moved

**Orientation:** orientaion of the splitter: horizontal or vertical

**TabIndex**: the order of the group of controls when tabbing through the form. Note that the contained controls then have their own TabIndex order within the GroupBox.

**Visible**: boolean to control visibility of the GroupBox and all contained controls

**Enabled**: boolean to control whether all contained controls are enabled

**Tag**: System.Object which can be used to store application-specific data

### Important Events

**None**

# StatusStrip (MyEditor)

Places a status bar at the bottom of the form. Normally used for adding a ProgressBar or StatusLabel.

### Important Fields

**(Name)**: the control name within the Form Class

### Important Events

**None**

# CheckBox

Works the same as the RadioButton except that any number of CheckBoxes can be selected on a Form at once.

### Important Fields

**(Name)**: the field name within the Form Class

**Checked**: boolean to indicate if the checkbox is checked or not. Your code should default a selection if the associated data does not exist yet.

**Location**: x, y

**Size**: width, height

**TabIndex**: the order of the field when tabbing through the form

**Text**: the text to show beside the checkbox

**Visible**: boolean to control visibility

**Enabled**: boolean to control whether the CheckBox is enabled for entry

**Tag**: System.Object which can be used to store application-specific data

### Important Events

#### CheckedChanged (see RadioButton)

# DateTimePicker (EditPerson)

A control to allow selecting a DateTime value.

### Important Fields

**(Name)**: the control name within the Form Class

**Format**: how the DateTime will be formatted in the control (Long/Short/Time/Custom)

**CustomFormat**: if Format==Custom, then define the custom format to use. Press F1 in the property to see the online documentation with all possible formatting options.

**MinDate**: the earliest date that this control can display. The absolute minimum is defined by the static variable DateTimePicker.MinimumDate, which is 1/1/1753, and which is the default value for MinDate. This could cause runtime errors since the default value of a DateTime field is 1/1/1. If you try to copy a DateTime variable to DateTimePicker.Value, the app will raise a runtime exception if the variable is less than MinDate.

### Important Events

#### ValueChanged

Occurs when the DateTime value changes.

Accepts the empty **EventHandler**() delegate because the event is limited to only the current control.

Example for adding the delegate method:

this.objectName.ValueChanged +=

new EventHandler(this.ObjectName\_\_ValueChanged);

The EventHandler delegate method must have the following signature:

private void **ObjectName\_\_ValueChanged(object sender, EventArgs e)**

Important Fields in sender

If you want to show a blank field if the Value == MinDate:

DateTimePicker dtp = (DateTimePicker) sender;

if( dtp.Value == dtp.MinDate )

{

dtp.CustomFormat = " ";

}

else

{

dtp.CustomFormat = "MMM d, yyyy";

}

Important Fields in EventArgs

None.

# MonthCalendar