Lesson 09:  MyWPF on C:\X\CSChap09.sln

WPF Forms

[TicTac](#TicTac)

[MouseDown](#MouseDown)

[animation](#animation)

[WPFBlazor](#WPFBlazor)

[GitHub](#GitHub)

[Commandline](#commandline)

[**FileHandling**](#FileHandling)

[Exception](#Exception)

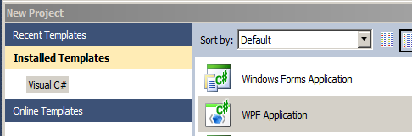
**Windows Presentation Foundation (WPF)**

<https://youtu.be/exmSDeSEKlY> 22:11 Up to drawing ellipse.

[startXaml](#startXamal) [xaml](#xaml)  [Binding](#Binding)

* File, New Project.

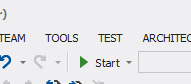
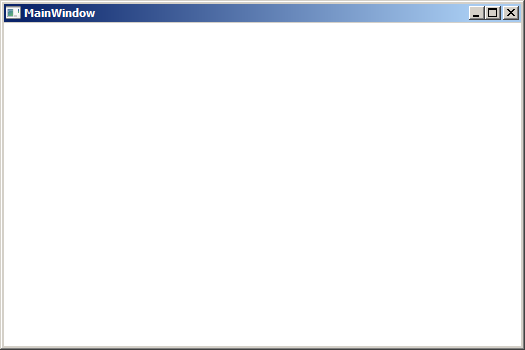
WPF App(.NET Framework).



* Choose WPF Application. Call it **MyWPF** and click OK.

You may wish to immediately run the program ie:

* Click Run.



Our main window form appears:

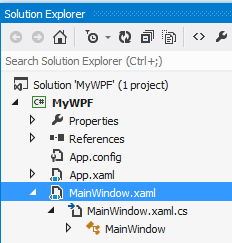
Close the window or click



to stop.

If your design window (See next page) is not showing…

* …double-click on **MainWindow.xaml**.



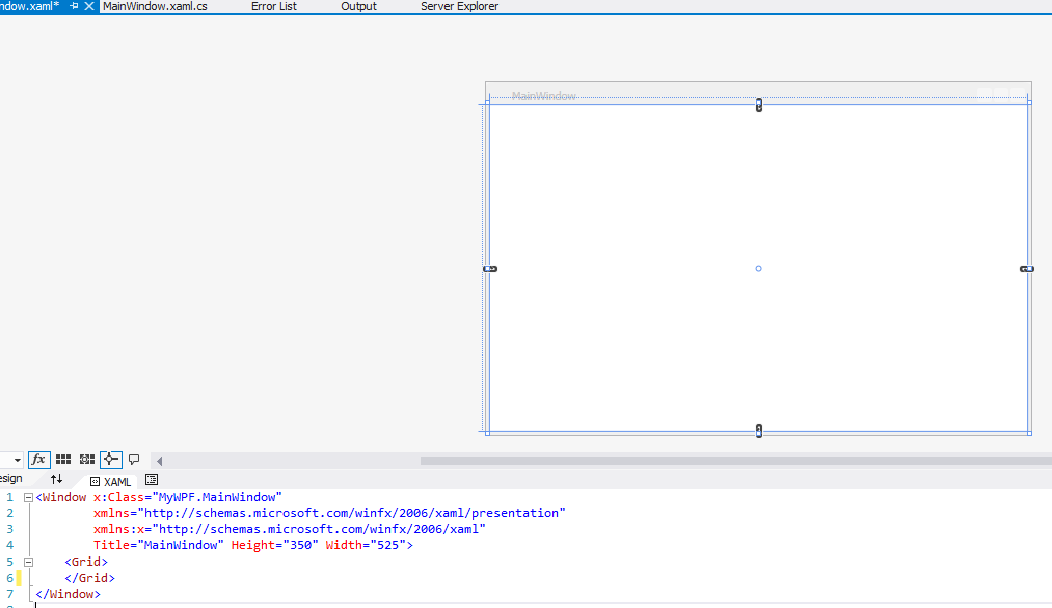
For source code projects: GitHub: <https://github.com/edthehorse/jack>

Download (From Add File) and Unzip CSChapt10.

Code at <https://github.com/edthehorse/jack/upload/main>

Download zip and unzip it

* Note that in our design view the Window form was produced automatically.



* A Grid is also produced by default in the window. This is the default container for our controls - buttons and text boxes etc.)

xmlns="http://sc

XAML (Extended Application Markup Language) is just a variant of XML. Like XML it is simply a text file. (Great for emailing your form to another developer!)

Like XML (and HTML) it consists of opening and closing tags

eg <Grid

</Grid>

<Grid>

</Grid>

</Window>

* Notice that the XAML has been produced automatically - in particular for this Grid.
* MyWPF is the name I have given to the project. (You will need to be very wary of this if copying code from elsewhere into your project.)
* These 2nd two lines of XAML starting with xmlns are also produced automatically.

They specify the XAML namespace (xmlns). Basically, they contain the proper specification conventions for our XAML code.

Don’t edit them.

* There are two ways we can edit the Heightand Width of our Main Window for example.

1. By directly editing the XAML code! and 2. Dragging the handles (on the very outside of our window:

Watch the XAML change as you drag! Try it.

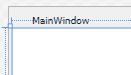
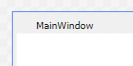
Actually, there is a third way: Programmatically as we shall see later.



Take care that you are not dragging the grid. The grid and the Main Window are close together.

Main Window Grid

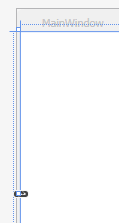
(The UI (User Interface) and its xaml is called the “**View**”.)



Note: If you click on their tag names in the xaml then the corresponding “graphic” will be respectively selected in the design window.



We can also resize the **Grid** (as well as the MainWindow):



* If you very carefully locate the Grid which is just inside the main window and drag it, you will see its XAML automatically change.
* The Name of the window MainWindow is prefixed by the name of our namespace MyWPF. This is usually the name of our project (but need not be).

<Window x:Class="MyWPF.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="MainWindow" Height="350" Width="525">

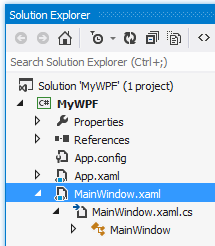
<Grid Margin="149,0,0,0">

</Grid>

</Window>

Take a look at the first line of the XAML.

Class specifies the name of the class which is automatically produced - representing our window ( MainWindow.) This class can be found in this file **MainWindow.xaml.cs**. Double-click on it to open it. See below for the code.



* You may wish to view **App.config** and **App.xaml**. They don’t really concern us at the moment but **App.xaml** contains the name of the startup form StartupUri="MainWindow.xaml"

We can also place some “global” settings here eg StaticResource – see later.

* MainWindow is derived from Window.

Window is in the System.Windows namespace hence:

using System.Windows; at the top of the code

Time to reflect on what’s going on here. We have designed a form. XAML has been produced automatically for us. Also, C# has automatically rendered a class MainWindow in this case corresponding to our XAML Class="MyWPF.MainWindow.

namespace MyWPF

{

public partial class MainWindow : Window

{

public MainWindow()

{

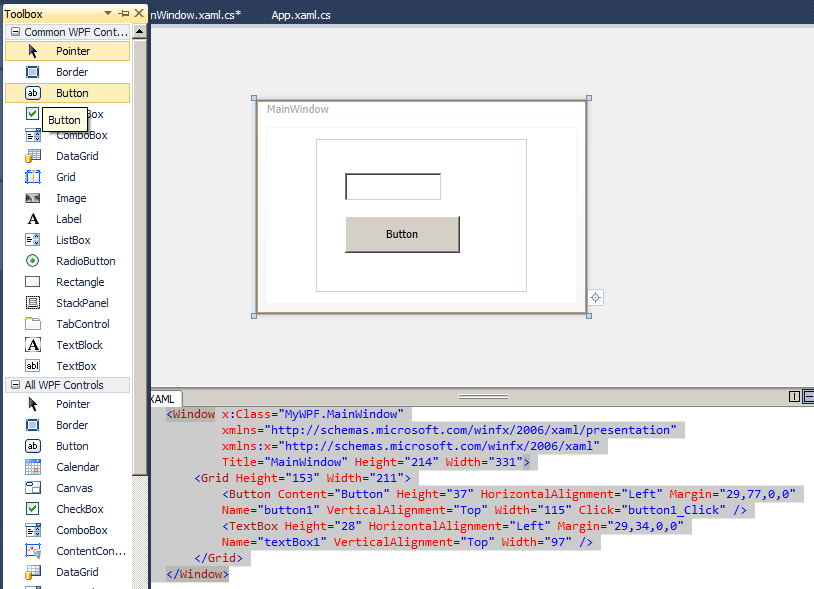
InitializeComponent();

}

}

}

**Controls**



Make sure that your Toolbox is visible (View, Toolbox from the menu.)

Choose Common WPF Controls.

* Choose the **MainWindow.xaml** tab and drag a Button and a TextBox onto the grid.
* Run it now to show that it is working so far.

We could use a Label instead of a TextBox -because we won’t be typing anything into our TextBox.

You should see the XAML for the button and the text box produced dynamically!

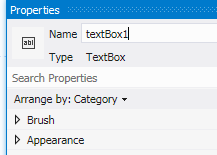
Common mistake: Make sure you have the correct control selected when you assign a property!

1. With the text box selected,

from the menu choose View,

Properties. or you can

2. Name it textBox1.



just press F4.)

.

You should see the corresponding XAML code appear.

We could instead have changed it in the XAML by typing

Name="textBox1" as above.

Take a look at the button XAML:There was no Name until we named it.

<Button Content="Button" HorizontalAlignment="Left" Height="49"

Margin="82,168,0,0" VerticalAlignment="Top" Width="155"/>

(The Name will only appear in the XAML if we *change* the name - as we did for the textbox.)

**Add an Event Handler** for the button as follows:

* IN Design View, double-click on the button. This Click event handler appears.



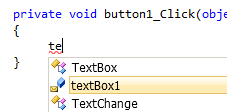
The name of the button is button1 (Nowadays it is Button1!) by default, so the name of the event will be

private void button1\_Click etc.)

* Type this into the button1\_Click event:

(We could also create an event handler stub by going to Properties of the Button, selecting Events and then double-clicking on the Click event.)

* Note that as we start to type, we are given the name of the Text Box textBox1. Double-click on the name (or just type the full stop) to accept it and continue to type the code below.



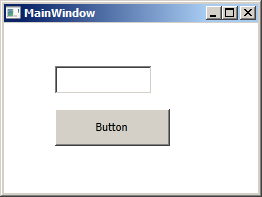
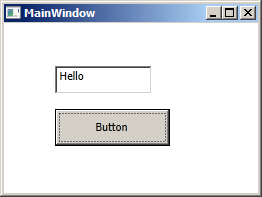
private void button1\_Click(object sender, RoutedEventArgs e)

{

textBox1.Text = "Hello";

}

* Build and Run.
* Hello appears.
* Click the button.



Once again take note where our code was automatically produced – in the MainWindow.xml.cs file – in our MainWindow class.

namespace myWPF

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

textBox1.Text = "Hello";

}

}

}

We are drawing controls onto a form. The XAML is produced automatically, and the corresponding code is produced automatically in the corresponding Main Window .cs file!

Delete our button and text box controls. The XAML will be conveniently automatically deleted as well! (We could just as well have just deleted the XAML instead!) but note that we must especially delete the event handler code ourselves manually as well!

private void button1\_Click(object sender, RoutedEventArgs e)

{

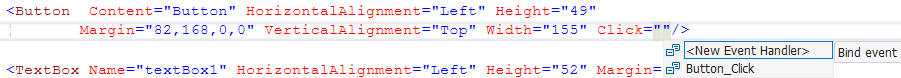
textBox1.Text = "Hello";

}

Alternatively-------------------------------------------------------------------

We *could* also create a new event by going to the XAML

and typing (anywhere!) in the here Click=



whereupon we are prompted for a <New Event Handler> etc.

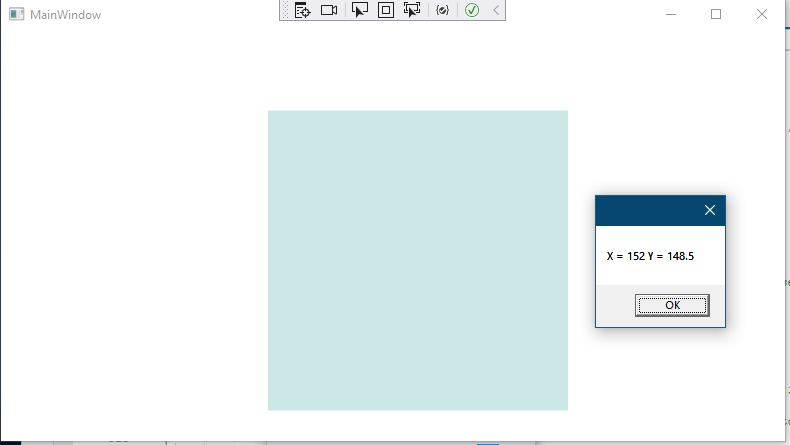
and… quite miraculously the code is automatically generated in the corresponding .cs file:

private void Button\_Click(object sender, RoutedEventArgs e)

{

}

MouseDown Event MyWPF on CsChsp08.sln



If we click on a grid…

We get the pixels left & down. (The grid is 300x300.)

<Grid x:Name="GameGrid" Height="300" Width="300" Margin="50,50,0,0"

MouseDown="GameGrid\_MouseDown" Background="#FFCCE8E6" >

</Grid>

(Apparently we need a grid BackGround for this to work!)

private void GameGrid\_MouseDown(object sender, MouseButtonEventArgs e)

{

Point P = e.GetPosition(GameGrid);

MessageBox.Show($"X = {P.X.ToString()} Y = {P.Y.ToString()}");

}

(We will use this event later in our TicTac game.)

**XAML**

<Grid Height="200" />

Grid is an Element

Height is an Attribute

The above could alternatively be written as:

<Grid Height="200">

</Grid>

In both of these cases, note that a value is written in quotes eg "200".

We could also write this as:

<Grid>

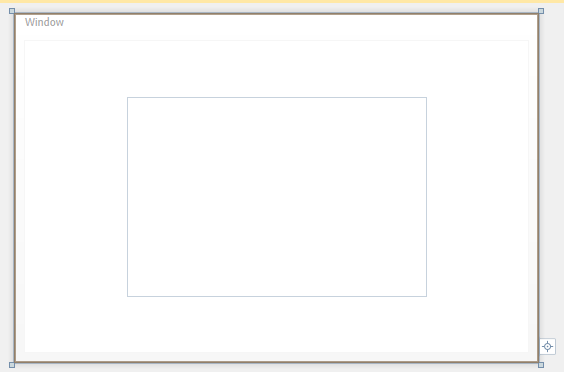
<Grid.Height>

But now the value need not be a string!

200

</Grid.Height>

</Grid>



In all cases the height of our grid will be 200.

Ref: [https://www.tutorialspoint.com/wpf/wpf\_**xaml**\_overview.htm](https://www.tutorialspoint.com/wpf/wpf_xaml_overview.htm)

<https://www.tutorialspoint.com/xaml/xaml_quick_guide.htm>

Or better Microsoft.

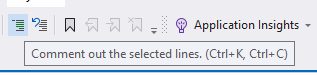
https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml/?view=netdesktop-7.0



To Comment Code:

<!-- -->

Select the code that you want to comment.



Click the Comment button.

Or use the control keys as suggested.

The selected code has become commented:

<Grid

<!-- <TextBlock Name="textBlock1" Margin="0,185,140,46" FontSize="20">

The <Bold> cat</Bold> sat on the mat

</TextBlock>-->

</Grid>

Hint: You will find that you can’t comment within tags. So - Do it *outside* the tag and you can then drag it in or beside the tag.

To Uncomment: Ctrl-K Ctrl-U

**Grid Properties:** A grid consists of cells.

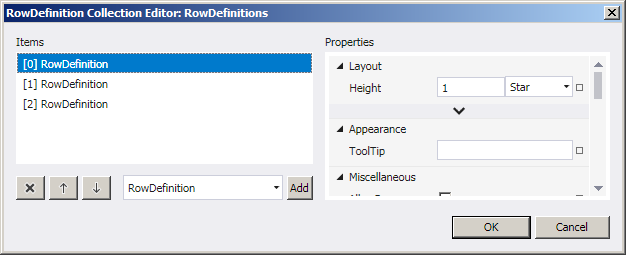
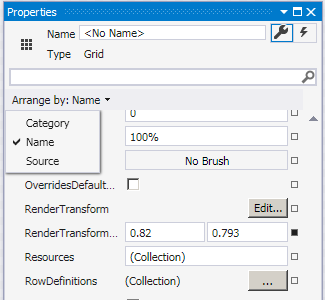
A 1 x 1 grid is created by default ie 1 cell & 1 columns ie a single rectangle.

* With the grid selected view the Properties. (Press F4)

If you can’t see your design Window double-click on **MainWindow.xaml**.

Finding properties is not always easy. Probably best to categorize them by name by choosing **Arrange by Name**.

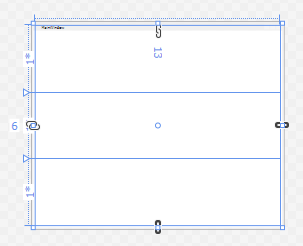
* Click RowDefinitions ellipses button. (Since we may specify many row widths we will specify a C*ollection*.



* Click Add 3 times.

Star means that the heights will adjust automatically at design-time.

We get 3 rows:



(It is probably easier to just *type* the xaml as below!)

and the corresponding XML:

<Grid HorizontalAlignment="Left" Height="533" Margin="6,-13,0,0"

The <Grid.RowDefinitions> tags indicates that we have a *collection* (of rows here).

VerticalAlignment="Top" Width="653">

<Grid.RowDefinitions>

<RowDefinition/>

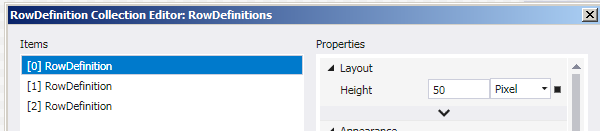
<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

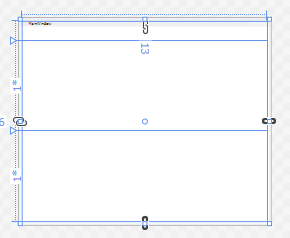
</Grid>

To make a row fixed width:



* Choose Pixel etc.

50 pixel fixed height.



Hint: You can click just outside the margin to set a row (& column).

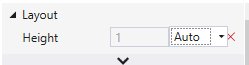
and the corresponding xaml:

<RowDefinition Height="50"/>

<RowDefinition/>

<RowDefinition/>

Auto



Will accommodate size of controls.

Variable width is denoted by an asterisk(\*)

Other Containers:

Our XAML produced so far has an opening and closing Grid elements by default:

<Grid>

</Grid>

The grid is a type of container which can hold controls.There are other containers that our main window can have: (Actually, we are only allowed to place only one control directly onto our main window ie if we do *not* have a container.)

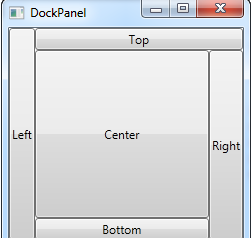
We can also have a StackPanel, Canvas etc see below.

They all have different behaviours when the Window is resized etc.

We have seen that with a Grid, child elements can be arranged in tabular form

<https://wpf-tutorial.com/panels/introduction-to-wpf-panels/> **Introduction to WPF panels**

# [The DockPanel](https://wpf-tutorial.com/panels/wrappanel/)

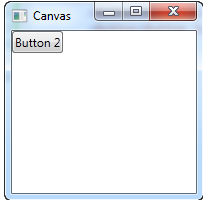
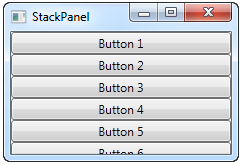
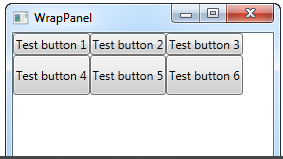


Use the DockPanel whenever you need to dock one or several controls to one of the sides

 [The Canvas control](https://wpf-tutorial.com/panels/canvas/)

The **WrapPanel** will position each of its child controls next to the other, horizontally (default) or vertically, until there is no more room, where it will wrap to the next line and then continue..

 [The Canvas control](https://wpf-tutorial.com/panels/canvas/)



# The WrapPanel control

# [The StackPanel control](https://wpf-tutorial.com/panels/stackpanel/)

|  |  |
| --- | --- |
|  | In a stack panel, child elements can be arranged in a single line, either horizontally or vertically, based on the orientation property. |

 [The Canvas control](https://wpf-tutorial.com/panels/canvas/)

Canvas allows you to assign specific coordinates to each of the child controls, giving you total control of the layout. Nothing will appear til you give some coordinates.

the child elements can be positioned explicitly using coordinates that are relative to the **Canvas** any side such as left, right, top and bottom.

[The Canvas control](https://wpf-tutorial.com/panels/canvas/)

(Used often.)

[UniformGrid](https://wpf-tutorial.com/panels/introduction-to-wpf-panels/#aelm390)  UniformGrid in CSChap09.sln

<UniformGrid x:Name="GameGrid"

The UniformGrid is just like the Grid but all rows and columns will automatically have the same size.

Width="300" Height="300"

Rows="3" Columns="3" >

<UniformGrid.Background>

<ImageBrush ImageSource="Assets/Grid.png"/>

</UniformGrid.Background>

</UniformGrid>

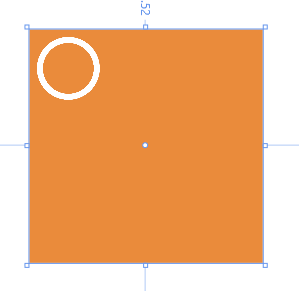
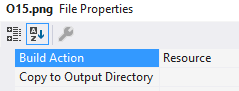
Unfortunately, the grid lines are not visible.

**Exercise**: Place an image at the top left (so we might then be able to make out make the top left cell.)

Use: <Image Source="Assets\O15.png"></Image>

and place the image O15.png in an /Assets folder.

Make sure that image is designated as a Resource:



(We will use this event later in our TicTac game.)

Using 3rd party XAML to make a control:

As well as drawing controls using the ToolBox as we have just done, we can simply cut and paste code for them! For example, we may cut and paste from this site:

Visit <http://www.c-sharpcorner.com/beginners/>

Mahesh Chand CSharpcorner (v good)



### [Accordion in WPF Toolkit](http://www.c-sharpcorner.com/uploadfile/dpatra/accordion-in-wpf-toolkit/default.aspx)

### [AutoCompleteTextBox in WPF](http://www.c-sharpcorner.com/uploadfile/rahul4_saxena/autocompletetextbox-in-wpf/default.aspx)

### [AutoComplete Folder TextBox in WPF](http://www.c-sharpcorner.com/UploadFile/lycj/wpf-autocomplete-folder-textbox/)

### [Area Chart in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/area-chart-in-wpf/default.aspx)

### [Bar Chart in WPF](http://www.c-sharpcorner.com/UploadFile/mahesh/bar-chart-in-wpf/)

### [Button Control in WPF](http://www.c-sharpcorner.com/uploadfile/dbeniwal321/button-control-in-wpf/default.aspx)

### [Border in WPF](http://www.c-sharpcorner.com/UploadFile/mahesh/border-in-wpf/)

### [Canvas in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/canvas-in-wpf/default.aspx)

### [CheckBox in WPF](http://www.c-sharpcorner.com/UploadFile/mahesh/wpf-checkbox/)

eg Choose Ellipse in WPF and then cut and paste the code from here between our Grid tags as shown below:

<Grid>

<Ellipse Width="200" Height="100"

Fill="Blue" Stroke="Black" StrokeThickness="4" />

</Grid>

WPFApp1 in CSAChap09.sln

### [Calendar in WPF](http://www.c-sharpcorner.com/UploadFile/mahesh/wpf-calendar-control/)

### [Closable Tab Control in WPF](http://www.c-sharpcorner.com/uploadfile/dpatra/closable-tab-control-in-wpf/default.aspx)

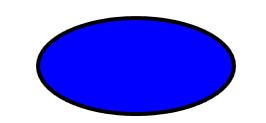
### [Column Chart in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/column-chart-in-wpf/default.aspx)

### [ComboBox in WPF](http://www.c-sharpcorner.com/UploadFile/mahesh/wpf-combobox/)

### [DatePicker in WPF](http://www.c-sharpcorner.com/UploadFile/mahesh/wpf-datepicker/)

### [DockPanel in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/dockpanel-in-wpf/default.aspx)

### [Drawing Brush in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/wpf-drawing-brush/default.aspx)



### [Ellipse in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/ellipse-in-wpf/default.aspx)

### [Expander Control in WPF](http://www.longhorncorner.com/uploadfile/raj1979/expander-control-in-wpf/default.aspx)

### [Focus Manager in WPF](http://www.c-sharpcorner.com/uploadfile/dpatra/focus-manager-in-wpf/default.aspx)

### [GridView in WPF](http://www.c-sharpcorner.com/UploadFile/mahesh/gridview-in-wpf/)

### [Grid in WPF](http://www.c-sharpcorner.com/UploadFile/mahesh/grid-in-wpf/)

### [Hyperlink in WPF](http://www.c-sharpcorner.com/UploadFile/mahesh/hyperlink-in-wpf/)

### [Icon in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/windows-icon-in-wpf/default.aspx)

### [ImageBrush in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/using-imagebrush-in-wpf/default.aspx)

### [Image Viewer in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/image-viewer-in-wpf/default.aspx)

### [Label in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/wpf-label/default.aspx)

### [Line Chart in WPF](http://www.c-sharpcorner.com/UploadFile/mahesh/line-chart-in-wpf/)

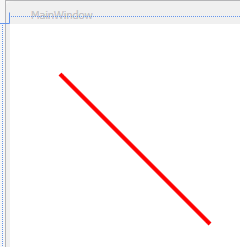
### [Line in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/line-in-wpf/default.aspx) eg Choose [Line in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/line-in-wpf/default.aspx)

### [ListBox in WPF](http://www.c-sharpcorner.com/uploadfile/mahesh/listbox-in-wpf/default.aspx)

Place the Line code with this (Between the <Grid> tags.) as shown:

WPfApp1

<Grid Margin="0,0,19,-1">



<Line

X1="50" Y1="50" Result:

X2="200" Y2="200"

Stroke="Red"

StrokeThickness="4" />

</Grid>

Using code to make a control: WPfApp1 See <https://youtu.be/exmSDeSEKlY> end.

We can achieve exactly the same thing (ie placing a line) with code!

First delete the above XAML that you just added. Copy this code from the previous web page or copy it from here and paste it as shown below in MainWindow.xaml.cs.

public partial class MainWindow : Window

{

The code will be called from the MainWindow() constructor ie when the program is run and the window appears.

(Alternatively we could make a button and call CreateALine() from there. )

public MainWindow()

{

InitializeComponent();

CreateALine();

}

public void CreateALine()

{

// Create a Line

If there are such inconsistencies when copying and pasting code, we will get an error message:

"The name 'InitializeComponent' does not exist in the current context".

Line redLine = new Line();

redLine.X1 = 50;

redLine.Y1 = 50;

redLine.X2 = 200;

redLine.Y2 = 200;

// Create a red Brush

SolidColorBrush redBrush = new SolidColorBrush();

redBrush.Color = Colors.Red;

using System.Windows.Media;

using System.Windows.Shapes;

// Set Line's width and color

redLine.StrokeThickness = 4;

Note how when we name a control it is given the prefix x.

Recall that x was the name of our class’s namespace:

xmlns:x="http://schemas. etc /xaml"

redLine.Stroke = redBrush;

// Add line to the Grid.

LayoutRoot.Children.Add(redLine);

}

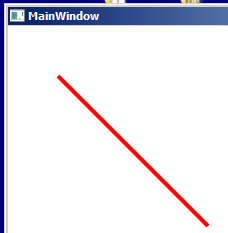
}

* LayoutRoot is the name that the author has given to his Grid.

From the Grid’s properties change the name of your grid to LayoutRoot.

(You should now see this in your XAML: <Grid x:Name="LayoutRoot">.)

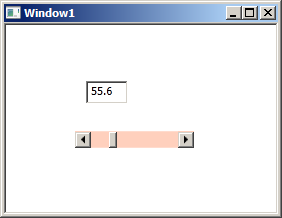
Run the code. You should get a red line as previously.



<https://learn.microsoft.com/en-us/dotnet/desktop/wpf/graphics-multimedia/shapes-and-basic-drawing-in-wpf-overview?redirectedfrom=MSDN&view=netframeworkdesktop-4.8>

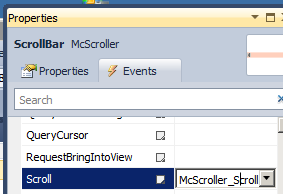
1. **Exercise**. Follow the article [**ScrollBar in WPF**](http://www.c-sharpcorner.com/UploadFile/mahesh/SBar06292009105859AM/SBar.aspx) Scroll in CSChap09.sln

* Write some event code which will show the value of the scroll bar in a label as the scroll bar button is moved.



**Hint**: First ascertain the Name of the scroll bar and then use its Value property as an indication of the scroll button's position horizontally…

…and create a Scroll event stub:



* Change the Content property of the label ie change its "caption". (See over)

More XAML:

The Content property

Content replaces the “Caption“ property of the old Windows Forms and Buttons.

<Grid>

<Button Content="Press" Margin="78,19,290,146" />

</Grid>

Placing images etc on a button

But Content is more than just a caption….

… eg it could be an image or even a StackPanel of a command button as below:

<Grid>

<!-- Setting the Content property using property element syntax -->

<Button Height="80" Width="100">

<Button.Content>

<StackPanel>

<Ellipse Fill="Red" Width="25" Height="25"/>

<Label Content ="OK!"/>

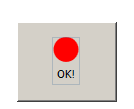
</StackPanel>

Anything that is contained between the control name tags ( Button in this case) is content. This is true for any controls that have a Content property.

</Button.Content>

</Button>

</Grid>



* We can safely remove the <Button.Content> and </Button.Content> tags since this is what is implicitly between Button element tags - in this case so there is no need to include it.

Good XAML Reference: <http://www.blackwasp.co.uk/XAML.aspx> (v good does styles etc)and

<https://wpf-tutorial.com/xaml/basic-xaml/>

Begin: A page is usually used in navigation-style WPF applications, whereas a window is used for standard windows and [dialog boxes](http://www.blackwasp.co.uk/WPFWindowShowDialog.aspx)

**Markup Extensions**

Markup extensions are dynamic placeholders for attribute values in XAML. **They resolve the value of a property at runtime**. Markup extensions are surrounded by curly braces (Example: Background="{StaticResource NormalBackgroundBrush}"). WPF has some built-in markup extensions, but you can write your own, by deriving from MarkupExtension. These are the built-in markup extensions:

eg Styles

<Window x:Class="myWPF.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Height="350" Width="525">

<Window.Resources>

<Style TargetType= "Button" x:Key = "myButtonStyle">

<Setter Property = "FontSize" Value = "16"/>

<Setter Property = "Foreground" Value = "Red"/>

</Style>

</Window.Resources>

<Grid>

<Button Margin="168,119,250,152" Style= "{StaticResource myButtonStyle}">

Help

</Button>

StaticResource means we are referencing a resource and myButtonStyle is the key for that resource.

</Grid>

</Window>

Every button created will have this style if it uses:

Style= "{StaticResource myButtonStyle}"



**References:**



<https://www.youtube.com/watch?v=2Dcq0ZI-2-w&list=PLI-F7ydFjNpC2lJ6UI-ZA2FnwR3fiP1IZ&index=5>

Advantage of WPF over Forms:

1. XAML which is a text file specification of a form (see later) provides a more expedient way of producing a form.

2. The many skills necessary to produce rich forms eg GDI, Direct X, Media Player are unnecessary with WPF. These are taken care of with WPF without requiring these extra programming skills.

<https://www.youtube.com/watch?v=0rcHiV8qS9g&list=PLI-F7ydFjNpC2lJ6UI-ZA2FnwR3fiP1IZ&index=4>

BINDING: **Binding** CSChap09.sln on QW .

*Recall* how we bound a TextBox to Command Button.

**MainWindow.xaml**

Consider this after “Events”.

Only difference is that Button Click is called from Xaml.

<Window x:Class="myWPF.MainWindow"

etc

<Button Click="Button\_Click" etc/>

<TextBox x:Name="textBox1" etc/>

etc

</Window>

Button\_Click" is bound. Behind the scenes binding!

**MainWindow.xaml,cs**

namespace myWPF

{

etc

private void Button\_Click(object sender, RoutedEventArgs e)

{

textBox1.Text = "Hello";

}

etc

}

These 2 files are independent! (Loose coupling.)

Developers could work upon each independently.

BINDING cont’d: **Binding Between TextBox Controls** **Binding1** project in Lesson9.sln on QW.

Make a form below with a text box and a label. testWPF.sln C:\Chap09 on QW

The following example shows a label whose Content is bound to the Text of the textbox. When you type text into the text box and the binding markup extension automatically updates the content of the label. ie **no code required**.

Draw twoTextBoxes.

<Grid>

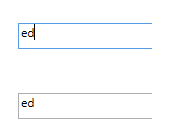
<TextBox Name = "TB1" Width="150" Margin="287,50,363,335" />

<TextBox Name = "TB2" Width="150" Margin="287,120,363,265"

Text = "{Binding Text, ElementName=TB1}"/>

Mode=TwoWay is the default. Type it in here (preceded with a comma) – it won’t make any difference.

</Grid>



* *Now if we type some text into the upper text box it immediately appears in the lower textbox*

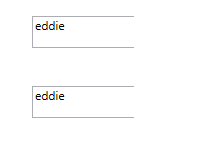
Now type some text into the lower text box. It is not immediately updated in the upper text box.

We need to click into the upper text box ie give it the focus before it is updated.

To remedy this, include this code:

<TextBox Name = "TB2" Width="150" Margin="287,120,363,265" Text = "{Binding Text, ElementName=TB1 , UpdateSourceTrigger=PropertyChanged}"/>

Now any change in either text box is immediately! reflected in the other



Binding between Form and a Control (TextBox)

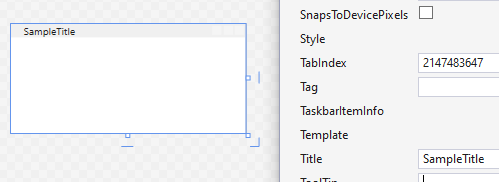
**The Windows’ (Form’s) DataContext.** SimpleFormDC in Chap09

Take a look at the Form’s properties:

Make sure you select the (outer) Window and then press F4.

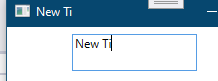
A **DataContext** of a Form is a code representation of its properties.

Other controls eg text box have DataContexts as well.



We wish to bind the Title property to a Control (textbox) on the form so that….

… when we type in the textbox, the Title will be dynamically updated.



Currently, the Form’s properties are not available for Binding. To make them so we must set the Form’s DataContext to these properties like so:

public MainWindow()

{

In the context! Of the Form, this represents the Form itself.

InitializeComponent();

this.DataContext = this;

}

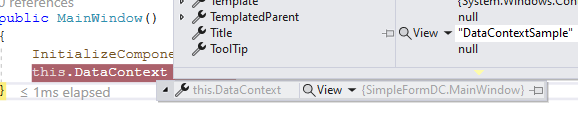
In the XAML we can now bind the Title property to the textbox to the Title property of the Form’s data context like so:

<Grid>

<TextBox Text="{Binding Title, UpdateSourceTrigger=PropertyChanged}" Margin="65,7,129,81" />

</Grid>

It is interesting to single-step this line to see the DataContext assignment before and after.



Binding other Form Properties. <https://wpf-tutorial.com/data-binding/using-the-datacontext/>

eg the Window’s Height and Width:

<Grid >

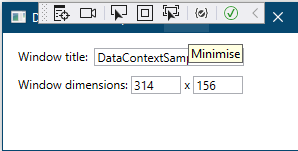
<TextBox Text="{Binding Title, UpdateSourceTrigger=PropertyChanged}" Height =" 45" Margin="66,7,193,207" />

<TextBox HorizontalAlignment="Left" Height="33" Margin="26,106,0,0" TextWrapping="Wrap" Text="{Binding Width}" VerticalAlignment="Top" Width="80"/>

<TextBlock HorizontalAlignment="Left" Height="33" Margin="129,106,0,0" TextWrapping="Wrap" Text="x" VerticalAlignment="Top" Width="51"/>

<TextBox HorizontalAlignment="Left" Height="33" Margin="154,0,0,0" TextWrapping="Wrap" Text="{Binding Height}" VerticalAlignment="Center" Width="50"/>

</Grid>



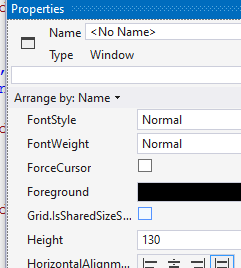
Resize and the text boxes update

**Summary**:

Simply set the form’s datacontext to the form! And all its (binding) properties are available.

ie this.DataContext = this; // Need this cause the window(form) does not default to its own DataContext.

and the text boxes will be bound to all of the Windows properties.eg Binding Opacity



------------------------------------------------------

More on **DataContext** binding

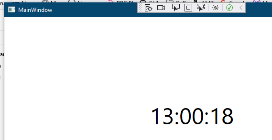
<https://www.youtube.com/watch?v=w1bSbSfes6o>

UnderstandingINotifyPropertyChanged.sln

Timer WPFTimer on C:\X\CSCHAP09 solution.

For Timer see: <https://www.c-sharpcorner.com/UploadFile/mahesh/timer-in-wpf/>

.



<Grid>

<Label Name="lblTime" FontSize="48" HorizontalAlignment="Center" VerticalAlignment="Center" />

</Grid>

public MainWindow()

Calls into the Framework which calls back to our mytimer\_Tick.

{

InitializeComponent();

DispatcherTimer timer = new DispatcherTimer();

timer.Interval = TimeSpan.FromSeconds(1);

timer.Tick += mytimer\_Tick; // Goes off every tick.

timer.Start();

}

void mytimer\_Tick(object? sender, EventArgs e)

{

lblTime.Content = DateTime.Now.ToLongTimeString();

}

(Label here is in the scope of MainWindow.)

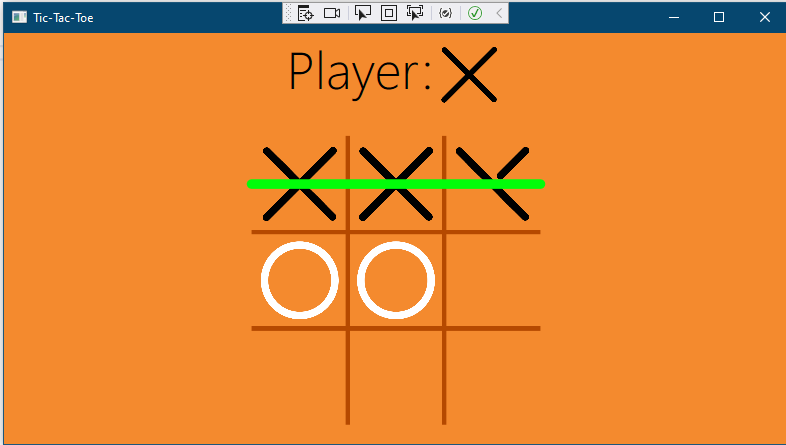
WPF TicTacToe See WPFTicTac in Chap09.sln

<https://www.youtube.com/watch?v=OHRWRpT9WcE>

Download Zip

GitHub Repository: [https://github.com/OttoBotCode/Tic-Ta...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbTRvTEVZdTh2TzhVTVdMZXUwcmdDRmhGRnVtQXxBQ3Jtc0ttOTdiN2dQUzk4ZTEtUlFzMmRGZ1FmYl9xaHpWcVk0dm1LamkxTEJuNm1JQVVzYkZ3VktheGlWb1NMQjRFNzV6QjJBSW5fVVYxd3FXa04zZllwM1FES2tsdmtOa2RybXh4YzdtZld1OWVZem1jaVEtVQ&q=https%3A%2F%2Fgithub.com%2FOttoBotCode%2FTic-Tac-Toe&v=OHRWRpT9WcE)

or mine: <https://github.com/edthehorse/jack>

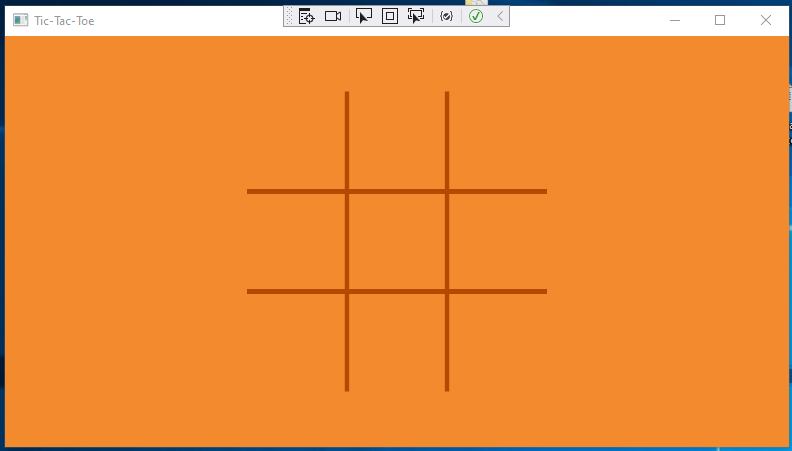


To make a Basic TicTac grid:

See WPFTicTac1 in Chap09.sln.

A “hash” image is superposed onto a 3x3 UniformGrid which is inside a Canvas.

The grid.png image should be placed in an Assets folder.



<Grid>

<Canvas

Width="300" Height="300" Margin="20">

<UniformGrid x:Name="GameGrid"

Width="300" Height="300"

Rows="3" Columns="3" >

<UniformGrid.Background>

<ImageBrush ImageSource="Assets/Grid.png"/>

</UniformGrid.Background>

</UniformGrid>

</Canvas>

</Grid>

Also, the Background of the Window is set to a custom colour.

Background="{StaticResource BackgroundColor}"

… and in App.xaml insert:

<Application.Resources>

<SolidColorBrush x:Key="BackgroundColor" Color="#f48a2e"/>

</Application.Resources>

**async & await** See next chapter.

eg

async void OnGameEnded

……….

await ShowLine says wait for me to render the line on the UI (ie the grid).

How His game board is animated.

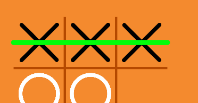
1st just draw



ie Just run the game and this appears.

Not from code/animation – XAML!

Then: End of Game: **-> Animations**:

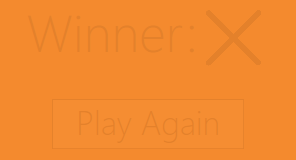
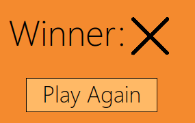


1. Line is animated.

2.FadeOut End-of-game screen and



3.Fade in/out the Winner screen and



3.Then fade in new game.



See commenting WPFTicTac on Chap09.sln

//Summary of C:\Users\User\Desktop\XandYOz\X\WPFTicTac

//XAML sets up the grid. and

//<StackPanel x:Name = "TurnPanel"

//< Image x: Name = "PlayerImage"

//< Canvas x: Name = "GameCanvas"

//< UniformGrid x: Name = "GameGrid"

//< Line x: Name = "Line"

//etc are referred to by the .cs

In the code:

//Set up a 3x3 Image holder grid corresponding to the array GameGrid)

//The Grid fades out after game is complete into the “Play Again ?” screen and then onto a new grid.

//The other animation is the animation of the X & O’s using keyframes.

//Upon clicking on the grid with say an X then both the array GameGrid and the corresponding grid square are updated.

**Enums**

public class WinInfo

{

public WinType Type { get; set; }// Row,column, diag , none

public int Number { get; set; }// No. of the row/column.

}

where

public enum WinType

{

Row, Column, MainDiagonal, AntiDiagonal

}

Also

public enum Player

{

None, X, O

}

public Player[,] GameGrid { get; private set; } // contains X,O,none

eg

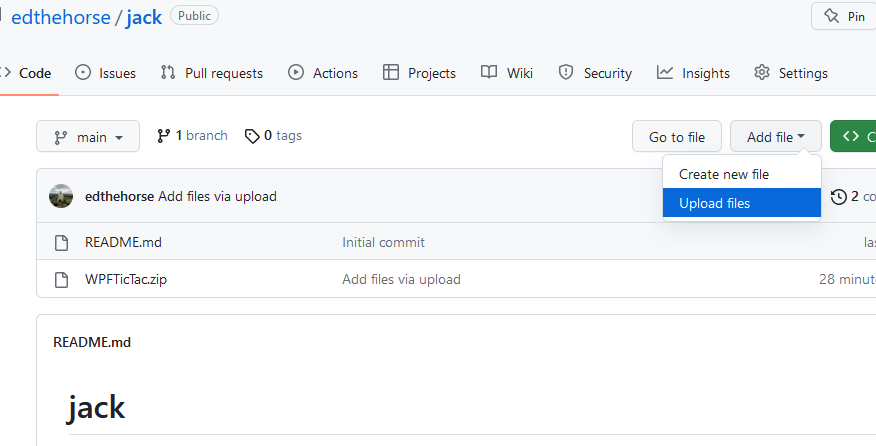
{ { X O X} ,

{ X O X} ,

{ X O X} }

GitHub

Zip the project before uploading.



https://github.com/edthehorse/jack

Tuples

<https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-tuples>

(int, int)[] squares

foreach ((int r, int c) in squares) // All the 9 squares. Tuple?

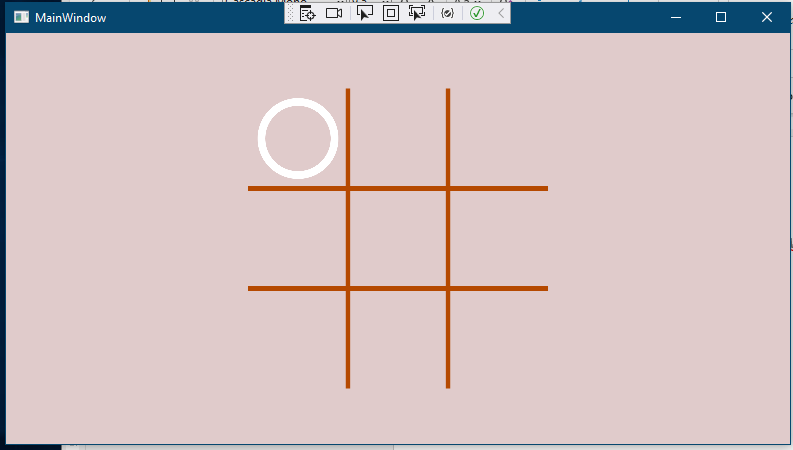
{

Dictionary

See Chapt 10.

Exercise WPFTicTacBare on CSChap09.

Using the WPFTicTac code, make a basic program which places an ”O” wherever we click on the grid.



We will place the “O” image on the grid - without animation.

Hints:

Copy some of his xaml code from MainWindow.xaml to make the UniformGrid.

Use his image O15.png which is placed in a (new) Assets folder.

Use his array of Image controls:

private readonly Image[,] imageControls = new Image[3, 3];

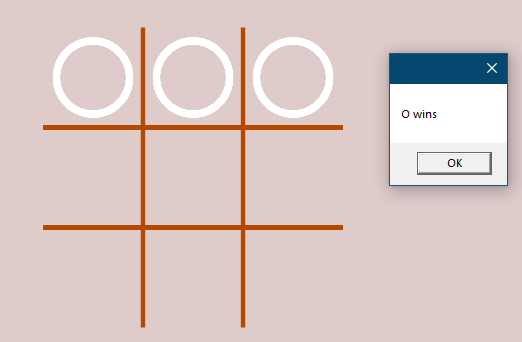
Use his MouseDown event to ascertain the row & column in his 3x3 grid.

private void GameGrid\_MouseDown(object sender, MouseButtonEventArgs e)

{

imageControls[row, col].Source = new BitmapImage(new Uri("pack://application:,,,/Assets/O15.png"));

To Test For a Win WPFTicTacBare1 on CSChap09



**Exercise**: For the moment, just test the top row!

Hint: In the MouseDown use

if (imageControls[row, col].Source != null)

In practice this is not a good technique ie testing for null image etc.

Better: Is to use an array which mirrors the state of the image controls. He uses public Player[,] GameGrid // contains X,O,none in order to do this

where Player is an enum:

public enum Player

{ None, X, O }

Exercise: Check for all winning combinations.

For the winning logic see <https://www.youtube.com/watch?v=gTt1iqVs0_U>

(He uses a simple console app and he uses a ONE dimensional array)

WPFTicTacBare1 in C:\X\CSChap09

**Exercise:** WPFTicTacBare2 in C:\X\CSChap09

Use WPFTicTacBare1 to make a (non-animated) game.

No need to check if there’s already an O or X already on a square that we click.

Just MessageBoX “Winner” for the moment if either X or O wins.

To Animate

**Exercise**: Animate the placement of the O in the top left:

See WPFAnimateXO in C:\X\Chap09.sln see comments is OK.

Basically objKeys holds 16 images of O.

objKeys.Duration = TimeSpan.FromSeconds(2); // Total animation duration.

Uri oUri = new Uri($"pack://application:,,,/Assets/O{i}.png"); // Respective locations of O1.png etc. 16 of them.

BitmapImage oImg = new BitmapImage(oUri); // Create respective bitmaps.

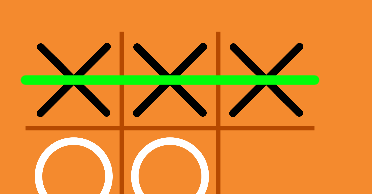
then imageControls[0, 0].BeginAnimation(Image.SourceProperty, objKeys);

evenly spaces the time between images.

To Draw an Animated Line:

C:/X/**WPFTicTac** on Laptop. See comments

C:/X/**WPFTicTac** on QW PC. See comments See [WPFAnimationLesson](#WPFAnimationLesson)



Called by an event.

return (new Point(0, y), new Point(GameGrid.Width, y)); // Return the line ends coords (0,50) (300,50).

private async Task ShowLine(WinInfo winInfo) // Draws an animated line (at top).

{

(Point start, Point end) = FindLinePoints(winInfo); // tuple

// 0,50 300,50

Line.X1 = start.X; // 0

Stage1. Set the line start,

Line.Y1 = start.Y; // 50

DoubleAnimation x2Animation = new DoubleAnimation // Animate the line X.

{

Duration = TimeSpan.FromSeconds(.25), // Lasts for 1/4 sec.

From = start.X, // 0

Stage2. Set the animation parameters.,

To = end.X // 300

};

DoubleAnimation y2Animation = new DoubleAnimation // Animate the line Y.

{

Duration = TimeSpan.FromSeconds(.25),

From = start.Y, // 50

To = end.Y // 50

};

Line.Visibility = Visibility.Visible;

Line.BeginAnimation(Line.X2Property, x2Animation); // Animate the line.

Stage3. Call the animation.

Line.BeginAnimation(Line.Y2Property, y2Animation);

await Task.Delay(x2Animation.Duration.TimeSpan); // .25 sec

}

Where Line is defined:

<Line x:Name="Line"

Stroke="{StaticResource LineColor}"

StrokeThickness="10"

StrokeStartLineCap="Round"

StrokeEndLineCap="Round"

Visibility="Hidden"/>

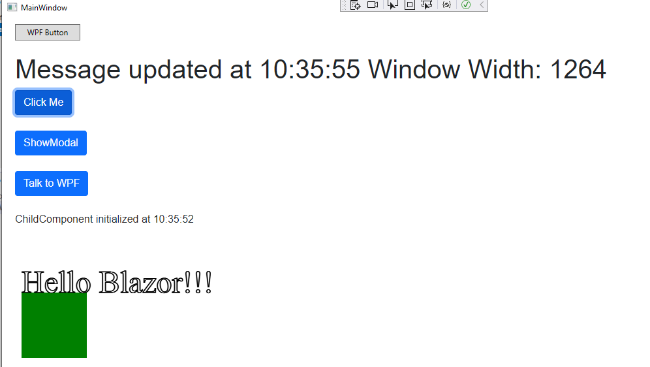
BeginAnimation(PropertyToBeAnimated, TimeLineForTheAnimation)

WPFBlazor



<https://www.youtube.com/watch?v=v1NBBZz5izs>

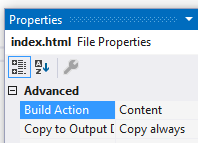
<https://github.com/carlfranklin/WpfBlazor>



Introduces a browser to WPF

“WPF can do more things than Blazor can do.”

He makes a WPF app!!



Copy Always is important.

C:

\X\WpfBlazor-master on QW see comments C:

\X\WpfBlazor-master on laptop

The WPF : MainWindow.xaml:

<!--PUT THE WPF PAGE ON THE WEB!!-->

<blazor:BlazorWebView

HostPage="wwwroot/index.html"

etc

</blazor:BlazorWebView>

serviceCollection.AddBlazorWebView();

**File Handling** page 526 C# 21 Days =FileHandling on C:\X\CSChap09 (uncomment)

<https://github.com/edthehorse/jack>

C#Ch8FileHandling.avi: <https://youtu.be/5sTlToVY7eQ>

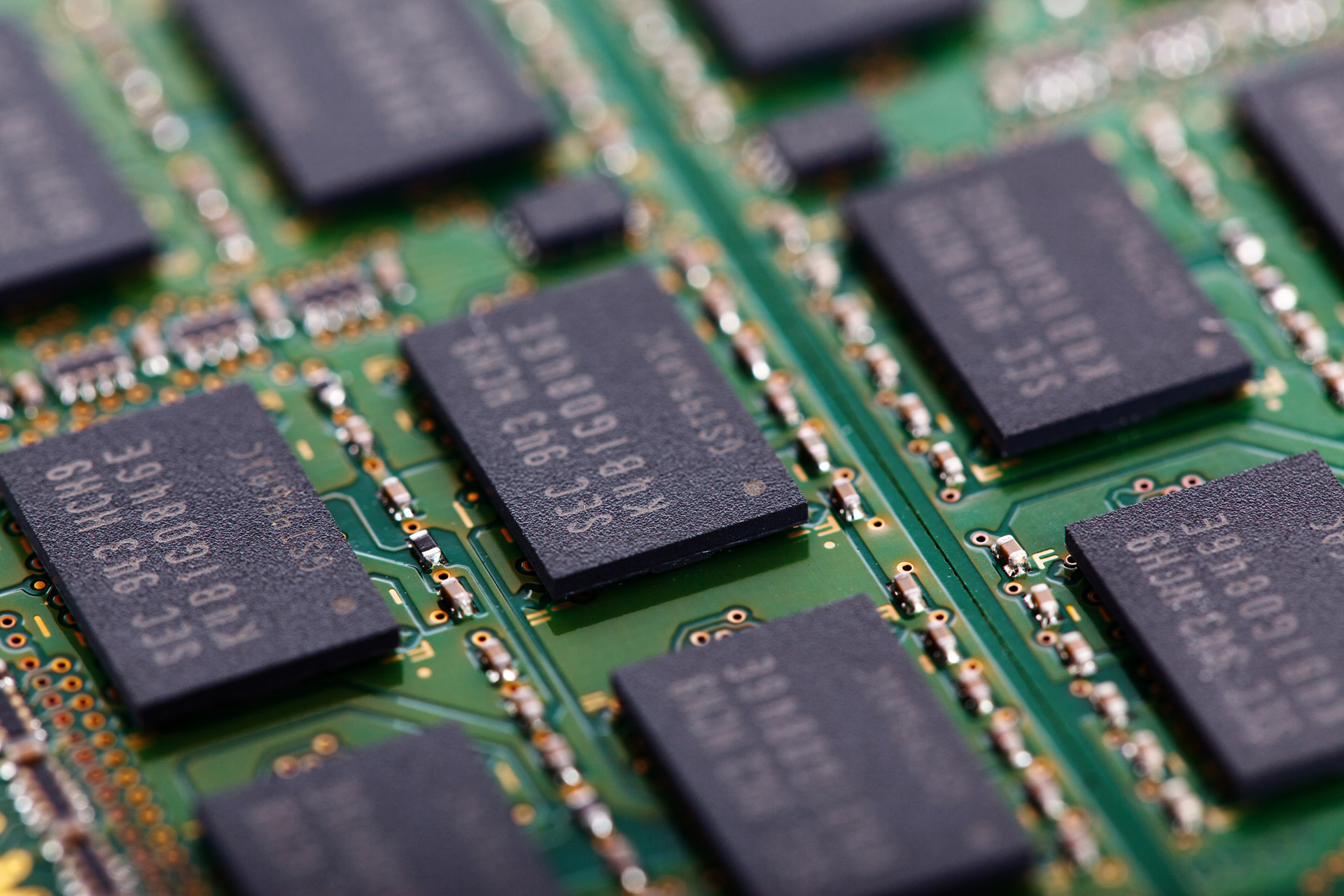
-> binary files

Sometimes we need to save our data onto disk. For example, we might save some lines of text or perhaps a set of numbers or even a picture (bitmap).

What is a Stream?

An object oriented representaion of our (flat) file. See below:

output



write

input

read

StreamWriter is a class. Right-click on it and take a look. eg Find the Close method of that class.

Memory Hard disk

Text Files Writing:

File is a class located within the System.IO namespace.

CreateText is a static method of the File class. It creates a text file.

using System;

using System.IO;

public class Writing

{

public static void Main()

{

StreamWriter myFile = File.CreateText("C:\\temp\\text.txt"); myFile.WriteLine("about time");

Writes to the file.

myFile.Close();

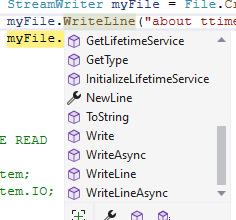
}



}

Beware: When we open a file for output, it will write over any data that was there before.

* Confirm this – save the file as above but with different data.



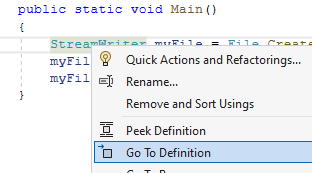
As mentioned, the stream (myFile in this case) is an object-oriented representation of the file (of characters).

<- Accordingly, it has properties and methods.

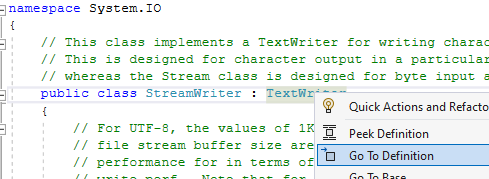
<-property (spanner)

<-method (box)

Interestingly if you wish to see the actual code for eg Writeline you can find the (overloaded) methods by



Right-click on StreamWriter and choose Go to Definition but …



… right-click on its base class TextWriter and then do a Ctrl-F to find WriteLine…

.. to find the overloaded WriteLine methods eg

public virtual void WriteLine()

{

Write(CoreNewLine);

}

public virtual void WriteLine(char[] buffer, int index, int count)

{

Write(buffer, index, count);

WriteLine();

}

**Text Files Reading**:

You may wish to view the file File,Open etc (or use NotePad to view the file.)

– or double-click on it.

using System;

using System.IO;

class Test

{

public static void Main()

{

Peek is a method which returns the next character to be read in the form of an integer. If there are no more characters, then -1 is returned.

string path = @"c:\temp\text.txt";

StreamReader sr = new StreamReader(path);

{

while (sr.Peek() >= 0)

{



Console.WriteLine(sr.ReadLine());

}

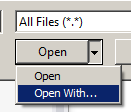
}

}

}

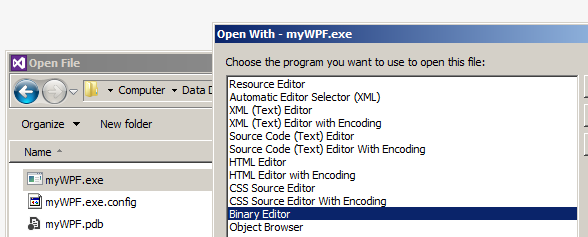
# To View Our File as Binary

If you have full version Visual Studio: To view our hex executable file use:



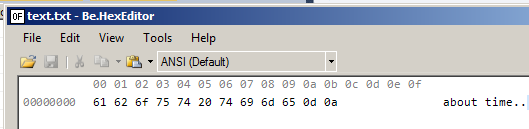
File, Open, File… then

* Open With…



* Choose Binary Editor.

To view the ascii list visit:<http://www.ascii.cl/>



The two bytes 0d 0a indicate the end of a text file.

These bytes are the ascii equivalents of the string. eg hex 61 is 97 which is a.

Write and read FileHandling on C:\X\CSChap09 (uncomment)

using System;

using System.IO;

class Test

{

public static void Main()

{

string path = @"c:\text.txt";

try

{

if (File.Exists(path))

{

File.Delete(path);

}

using (StreamWriter sw = new StreamWriter(path))

{

sw.WriteLine("about time");

}

using (StreamReader sr = new StreamReader(path))

{

while (sr.Peek() >= 0)

{

Console.WriteLine(sr.ReadLine());

}

}

}

catch (Exception e)

{

Console.WriteLine("The process failed: {0}", e.ToString());

}

finally



{

}

}

}

Peek is a method which returns the next character to be read in the form of an integer. If there are no more characters then -1 is returned.

**Binary Files** BinaryFiles (WPF) on CSCHap09

using is just a method of making sure that an object automatically goes out of scope when the using block finishes.

Streams are representations of “flat” files as objects of the StreamWriter class.

StreamWriter and StreamReader classes are used to create stream objects which are then used to transfer data to and from the opened file.

We saw for text files that letters eg a and b are written as their ascii equivalent – and so are the numbers eg 1 is written as 49 (or hex 31). But for a binary file 1 is written as 1 ie numbers as numbers.

Writing

* Run this:

To add a console window to a WPF:

Right click on the project, "Properties", "Application" tab, change "Output Type" to "Console Application", and then it will **also** have a console.

using System;

using System.IO;

class MyStream

{

public static void Main()

{

FileStream myFile = new FileStream("C:\\temp\\binFile", FileMode.CreateNew);

BinaryWriter bwFile = new BinaryWriter(myFile);

// Write the data

See page 532 C# 21 days.

for (int i = 0; i < 10; i++)

{

bwFile.Write(i);

}

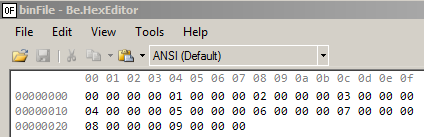
Close the stream as well as the file itself.

bwFile.Close();

myFile.Close();

}

}



Each number eg 2 is written as a four byte integer.

To avoid an error if the file already exists, we could use at the start:

string path = @"c:\temp\binFile";

if (File.Exists(path))

ie Delete it if it exists. Take care!

{

File.Delete(path);

}

Reading Binary File <https://www.dotnetperls.com/binaryreader>

To read back our binary file:

using System;

using System.IO;

class MyStream

{

public static void Main()

{

string path = @"c:\temp\binFile";

FileStream myFile = new FileStream(path, FileMode.Open);

BinaryReader br = new BinaryReader(myFile);

// Read data

for (int i = 0; i < (int)br.BaseStream.Length/sizeof(int); i++)

{

int b = br.ReadInt32();



br.BaseStream.Length/sizeof(int)

will give us the *number* of integers.

Console.WriteLine(b);

}

br.Close();

}

}

Note: FileHandling is better handled using Async Access

see <CSLesson10ver7.docx> AsyncWriteFile link etc.

**Running Programs from the Command Line** CommandLineDemo on CSChap09 on C:\X

Make a simple program to print hello to the screen as below. (Make **exp.sln**). For convenience locate the solution in a convenient folder eg **c:\temp**.

using System;

namespace exp

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("hello");

**DOS Commands**: (Not case sensitive.)

**cd\** will change to the root directory.

**cd temp** will change to the temp directory.

**dir** will list what’s in the current directory.

The up-arrow key will retrieve the previous DOS command.

}

}

}



To Run it from the Command Line.

* Open the Command Prompt window.

(You may need to do type **cmd** in the Windows run box to find it) and navigate to the file as follows:

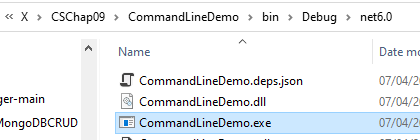
* Type **cd\** and **<Enter>** to change to **c:** (the root directory).

CD C:\X\CSChap09\CommandLineDemo\bin\Debug\net6.0

Then:

* (You might also like to double-click on the file to run it.)

Our program executes.



We now wish to pass some arguments to our executable so that we can say hello to whoever we choose.

We should also have:

using System;

namespace exp

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello " + args[0].ToString());

}

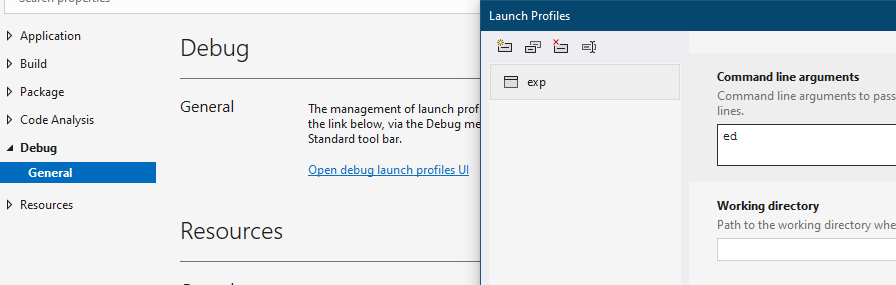
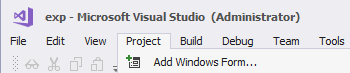
}}

**ed** is concatenated.



**To Set the Command Line Arguments from the IDE**

Choose Project and the exp Properties… at the bottom of the menu drop down.



Type the command line argument here.

Now if we run from Visual Studio (not from the command line) we get

**Hello ed**

appears as before.



**Exception Handling** Exceptions on CSChap09 Chapter 9 C#21 days

An Exception is a run-time error. ie an error which occurs when the program is running.

namespace Exceptions

{

internal class Program

{

static void Main(string[] args)

{

Whenever there is the possibility that some code may produce an error at run-time, then we place it in a try block with…

int y = 6; int x = 0;

try

{

int z = y/x;

Console.WriteLine(x.ToString());

… a corresponding catch block which will deal with that error.

}

catch (Exception e)

{

Console.WriteLine($"ERROR: {e.Message}");

}

finally

{

Console.WriteLine("Always do this error or not.");

}

Console.WriteLine("And this");

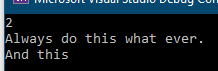
}

}

}



Change the above line to int y = 6; int x = 3;



Result:

Try just

Console.WriteLine($"ERROR: {e}");

We get the information about the error in its totality:



In the above, note the specific error System.DivideByZeroException.

We could use that to catch the error specifically:

catch (DivideByZeroException e)

{

// etc

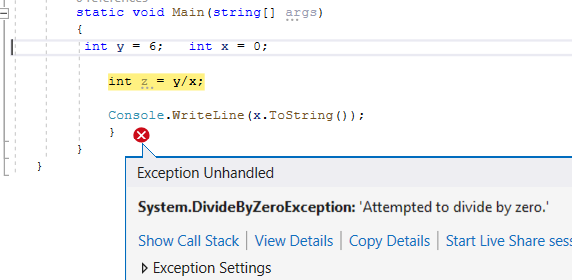
}

StackTrace will just tell you where the error occurred

Console.WriteLine($"ERROR: {e.StackTrace}");



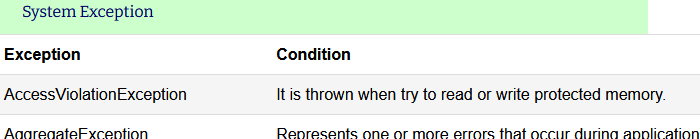
Try it without any error handling:



:

A list of exceptions can be found here:

<https://www.completecsharptutorial.com/basic/complete-system-exception.php>



Strings which are possible null.

using System;

class TryIt2

{

public static void Main()

{

String? s = "ed";

Console.WriteLine(s!.ToUpper());

}

}



? warns that s may be null.

The null forgiving operator ! makes the s not-null.

**Multiple Catch with a Single Try**

* Place *both* of these catch statements in your code.

using System;

class TryIt

{

public static void Main()

{

int[] arr = new int[3];

(arr[2]) is zero.)

try

{

Console.WriteLine(3/arr[2]);

}

If you don’t need to access its properties you can just have:

catch (DivideByZeroException)

catch (DivideByZeroException e)

{

Console.WriteLine(e.Message);

}

Generally the more specific error catches are placed first.

catch (IndexOutOfRangeException e)

{

Console.WriteLine(e.Message);

}

catch (Exception e)

This will catch any OTHER exception which may occur

{

Console.WriteLine(e.Message);

}

}

}

This “general” catch block must come AFTER the *specific* catch blocks.



**throw**

Throw your own error! This can be useful for debugging.

using System;

class Program

{

public static void Main()

{

try

{

throw (new DivideByZeroException());

}

catch (DivideByZeroException e)

{

Console.WriteLine(e.Message);

}

}

}