

در [مقاله قبلی](#) در مورد تعدادی از Layoutها صحبت کردیم و در این بخش به ادامه‌ی آن پرداخته و دو مبحث GridPanel و Custom Layout را بررسی می‌کنیم.

GridPanel

پنل پیش فرضی است که موقع ایجاد یک پروژه جدید WPF ایجاد می‌شود. چیدمان این نوع پنل به صورت سطر و ستون است و کارکرد آن بسیار مشابه جداول در HTML می‌باشد؛ با این تفاوت که در اینجا انعطاف پذیری بیشتری وجود دارد. هر سلول می‌تواند شامل چندین کنترل شود و یا هر کنترل می‌تواند چندین سلول را به خود اختصاص دهند و حتی می‌تواند روی کنترل‌های دیگر قرار بگیرند و همپوشانی کنترل‌ها را داشته باشیم.

تگ Grid Panel شامل دو تگ برای تعریف سطرها و ستون‌ها می‌باشد با استفاده از تگ Row Definition و Column Definition به تعیین تعداد سطر و ستون‌ها و اندازه آن‌ها می‌پردازیم:

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="*" />
    <RowDefinition Height="28" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="200" />
  </Grid.ColumnDefinitions>
</Grid>
```

گرید پنل بالا شامل 4 سطر و دو ستون است و تعیین اندازه آن‌ها توسط دو خاصیت Width و Height مشخص شده است که نحوه مقداردهی آن‌ها به صورت زیر است:

Fixed : یک مقدار ثابت، مثل سطر آخری که در کد بالا قرار می‌گیرد. این مقدار بر اساس یک واحد منطقی است و نه پیکسل که در [این مقاله](#) قبلاً بررسی کرده‌ایم.

Auto : به مقداری که احتیاج دارد فضایی را بخود اختصاص می‌دهد.

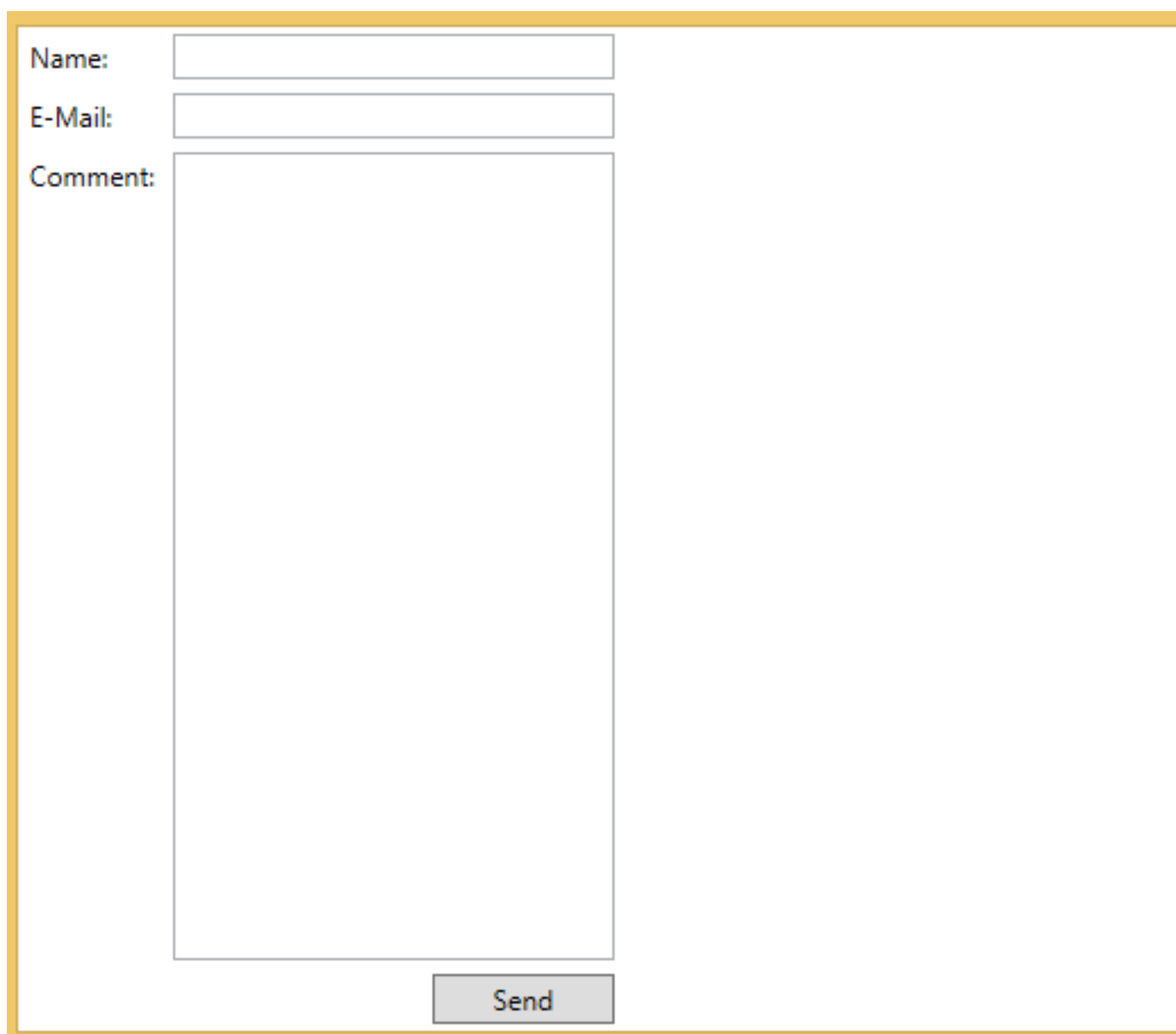
***** : هر آنچه از فضای موجود باقی مانده است را به خود اختصاص می‌دهد. علامت ستاره یک واحد نسبی است؛ به این صورت که می‌توانید مقدار فضا را به صورت زیر نیز بیان کنید. $3*$ و $2*$ به این معنی است که از پنج قسمت فضای باقیمانده سه قسمت و بعدی دو قسمت را به خود اختصاص می‌دهد. عبارت $*1*$ برابر است. عموماً با این علامت فضا را به شکل درصد بیان می‌کنند:

```
<ColumnDefinition Width="69%" /> <!-- Take 69% of remainder -->
<ColumnDefinition Width="31%" /> <!-- Take 31% of remainder -->
```

نحوه‌ی اضافه کردن المان‌ها به گرید به صورت زیر پس از تعیین تعداد سطرها و ستون‌ها انجام می‌گیرد و جایگاه هر المان در ستون یا سطر مربوطه توسط یک attached Dependency Property به نام‌های Grid.Column یا Grid.Row صورت می‌گیرد. خصوصیات Horizontal alignment و vertical Alignment هم برای تعیین موقعیت قرار گیری اشیاء در سلول به کار می‌روند و فاصله‌ی آن‌ها (کنترل‌ها) از لبه‌های گرید با margin محاسبه می‌شود.

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
```

```
<RowDefinition Height="*" />
<RowDefinition Height="28" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="Auto" />
  <ColumnDefinition Width="200" />
</Grid.ColumnDefinitions>
<Label Grid.Row="0" Grid.Column="0" Content="Name:" />
<Label Grid.Row="1" Grid.Column="0" Content="E-Mail:" />
<Label Grid.Row="2" Grid.Column="0" Content="Comment:" />
<TextBox Grid.Column="1" Grid.Row="0" Margin="3" />
<TextBox Grid.Column="1" Grid.Row="1" Margin="3" />
<TextBox Grid.Column="1" Grid.Row="2" Margin="3" />
<Button Grid.Column="1" Grid.Row="3" HorizontalAlignment="Right"
  MinWidth="80" Margin="3" Content="Send" />
</Grid>
```



تغییر اندازه در سمت کد هم می‌تواند توسط کدهای صورت گیرد.

```
Auto sized GridLength.Auto
Star sized new GridLength(1,GridUnitType.Star)
```

```
Fixed size new GridLength(100,GridUnitType.Pixel)
```

مثال:

```
Grid grid = new Grid();

ColumnDefinition col1 = new ColumnDefinition();
col1.Width = GridLength.Auto;
ColumnDefinition col2 = new ColumnDefinition();
col2.Width = new GridLength(1,GridUnitType.Star);

grid.ColumnDefinitions.Add(col1);
grid.ColumnDefinitions.Add(col2);
```

قابلیت تغییر اندازه‌ی سطر و ستون توسط کاربر

یکی از تگ‌های ویژه داخل گری، د تگ Grid Splitter است. برای قرارگیری تگ splitter ابتدا باید یک سطر یا ستون بین سطر و ستون‌هایی که می‌خواهید از یکدیگر جدا شوند ایجاد کنید و اندازه‌ی آن را auto تعیین کنید و سپس مانند بقیه‌ی اشیا توسط Grid.Column یا Grid.Row مانند کد زیر تگ splitter را به آن اختصاص دهید.

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <Label Content="Left" Grid.Column="0" />
  <GridSplitter HorizontalAlignment="Right"
    VerticalAlignment="Stretch"
    Grid.Column="1" ResizeBehavior="PreviousAndNext"
    Width="5" Background="#FFBCBCBC" />
  <Label Content="Right" Grid.Column="2" />
</Grid>
```

خاصیت ResizeBehavior مشخص می‌کند که ستون یا سطرهای کناری کدام باید تغییر اندازه داشته باشند.

مقدار پیش فرض این گزینه است و مشخص می‌کند سطر یا ستونی طرفی باید تغییر اندازه دهد که در Alignment آن آمده است	BasedOnAlignment
ستون یا سطر جاری به همراه ستون یا سطر بعدی	CurrentAndNext
ستون یا سطر جاری به همراه ستون یا سطر قبلی	PreviousAndCurrent
سطر یا ستون قبلی و بعدی که بهترین گزینه برای انتخاب است.	PreviousAndNext

خاصیت ResizeDirection جهت تغییر اندازه را مشخص می‌کند که شامل سه مقدار Row, Column و Auto است که مقدار پیش فرض آن auto است و نیازی به ذکر آن نیست و خود سیستم می‌داند که باید تغییر اندازه در چه جهتی صورت بگیرد.

ساخت Custom Layout یا یک پنل سفارشی (اختصاصی)

در این دو قسمت، شما با پنل‌های متفاوتی آشنا شدید که قابلیت‌های مفیدی داشتند؛ ولی گاهی اوقات هیچ کدام از این‌ها به کار شما نمی‌آیند و دوست دارید پنلی داشته باشید که مطابق میل شما عمل کند. برای ساخت یک پنل سفارشی یک کلاس می‌سازیم که از کلاس Panel ارث بری می‌کند. در اینجا دو متد برای Override کردن وجود دارند:

MeasureOverride: تعیین اندازه پنل بر اساس اندازه تعیین شده برای المان‌های فرزند و فضای موجود.

ArrangeOverride: مرتب سازی المان‌ها در فضای موجود نهایی.

کد نمونه:

```
public class MySimplePanel : Panel
{
    // Make the panel as big as the biggest element
    protected override Size MeasureOverride(Size availableSize)
    {
        Size maxSize = new Size();

        foreach( UIElement child in InternalChildren)
        {
            child.Measure( availableSize );
            maxSize.Height = Math.Max( child.DesiredSize.Height, maxSize.Height);
            maxSize.Width= Math.Max( child.DesiredSize.Width, maxSize.Width);
        }
    }

    // Arrange the child elements to their final position
    protected override Size ArrangeOverride(Size finalSize)
    {
        foreach( UIElement child in InternalChildren)
        {
            child.Arrange( new Rect( finalSize ) );
        }
    }
}
```

لینک‌های زیر تعدادی از پنل‌های سفارشی پر طرفدار هستند که بر روی اینترنت به اشتراک گذاشته شده اند:

[TreeMapPanel](#)

[Animating Tile Panel](#)

[Radial Panel](#)

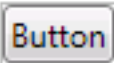
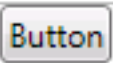
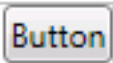
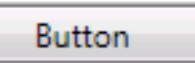

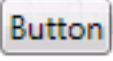









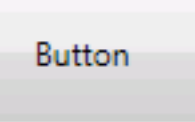
[Element Flow Panel](#)

[Ribbon Panel](#)

خواصی که باید در Layoutها با آنها بیشتر آشنا شویم:

Horizontal & Vertical Alignment

با دادن این خاصیت به کنترل‌های موجود، نحوه قرار گیری و موقعیت آن‌ها مشخص می‌گردد. جدول زیر بر اساس انواع موقعیت‌های مختلف تشکیل شده است:

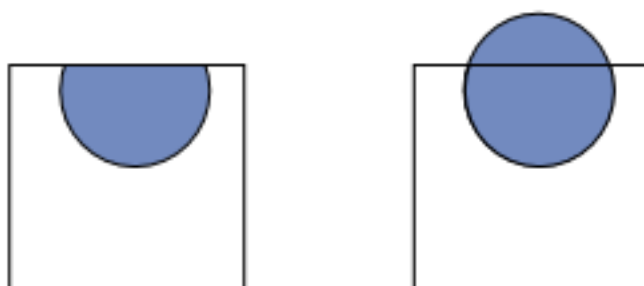
		HorizontalAlignment			
		Left	Center	Right	Stretch
VerticalAlignment	Top				
	Center				
	Bottom				
	Stretch				

Margin & Padding

این خاصیت‌ها حتماً برای شما آشنا هستند. خاصیت margin فاصله کنترل از لبه‌های Layout است و خاصیت padding فاصله محتویات کنترل از لبه‌های کنترل است.

Clipping

در صورتی که خاصیت ClipToBounds پنل برابر False باشد به این معناست که المان‌ها می‌توانند از لبه‌های پنل خارج شوند، در صورتی که برابر True باشد مقدار خارج شده نمایش نمی‌یابد.



ClipToBounds="True"

ClipToBounds="False"

Scrolling

موقعیکه از پنلی استفاده می‌کنید که با تمام شدن ناحیه‌اش روبرو شده‌اید ولی کنترل‌های داخلش هنوز ادامه دارند، نیاز به یک اسکرول به شدت احساس می‌شود. در این حالت می‌توان از ScrollViewer استفاده کرد.

```
<ScrollViewer>
  <StackPanel>
    <Button Content="First Item" />
    <Button Content="Second Item" />
  </StackPanel>
</ScrollViewer>
```

```
        <Button Content="Third Item" />
    </StackPanel>
</ScrollView>
```