WPF Panels

WPF Panels

Parent elements that support the arrangement of multiple children are known as *panels*. Here are the commonly used Panels:

- Grid
- Canvas
- StackPanel
- DockPanel

Grid

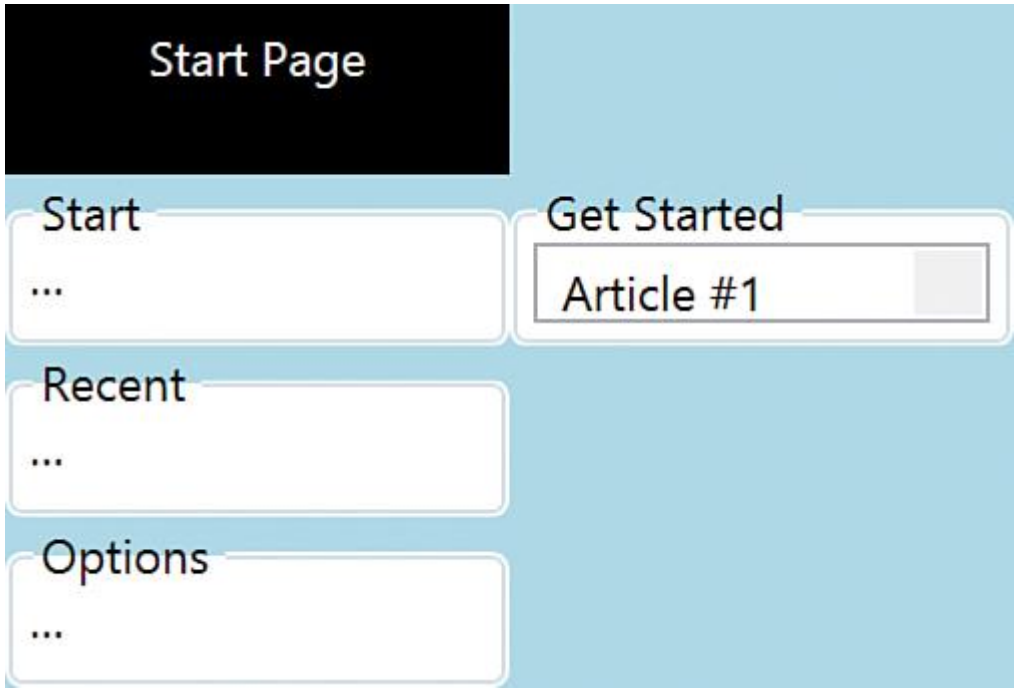
- Grid is the most versatile panel. It enables you to arrange its children in a multirow and multicolumn fashion

```
<Grid Background="LightBlue">

<!-- Define four rows: -->
<Grid.RowDefinitions>
  <RowDefinition/>
  <RowDefinition/>
  <RowDefinition/>
  <RowDefinition/>
</Grid.RowDefinitions>

<!-- Define two columns: -->
<Grid.ColumnDefinitions>
  <ColumnDefinition/>
  <ColumnDefinition/>
</Grid.ColumnDefinitions>
```

```
<!-- Arrange the children: -->
  <Label    Grid.Row="0" Grid.Column="0" Background="Black" Foreground="White"
            HorizontalContentAlignment="Center">Start Page</Label>
  <GroupBox Grid.Row="1" Grid.Column="0" Background="White"
            Header="Start">…</GroupBox>
  <GroupBox Grid.Row="2" Grid.Column="0" Background="White"
            Header="Recent">…</GroupBox>
  <GroupBox Grid.Row="3" Grid.Column="0" Background="White"
    Header="Options">…</GroupBox>
  <GroupBox Grid.Row="1" Grid.Column="1" Background="White"
    Header="Get Started">
    <ListBox>
      <ListBoxItem>Article #1</ListBoxItem>
      <ListBoxItem>Article #2</ListBoxItem>
      <ListBoxItem>Article #3</ListBoxItem>
      <ListBoxItem>Article #4</ListBoxItem>
    </ListBox>
  </GroupBox>
</Grid>
```



Grid

- RowSpan and ColumnSpan are set to 1 by default and can be set to any number greater than 1 to make an element span that many rows or columns.
  - adding this to the last GroupBox : *Grid.RowSpan="3"*
  - adding this to the Label : *Grid.ColumnSpan="2"*

Grid - Sizing the Rows and Columns

- Absolute sizing — Setting Height or Width to a numeric value representing device-independent pixels (like all other Height and Width values in WPF). Unlike the other types of sizing, an absolute-sized row or column does not grow or shrink as the size of the Grid or size of the elements changes.
- Autosizing — Setting Height or Width to Auto, which gives child elements the space they need and no more. For a row, this is the height of the tallest element, and for a column, this is the width of the widest element. This is a better choice than absolute sizing whenever text is involved to be sure it doesn't get cut off because of different font settings or localization.
- Proportional sizing (also called star sizing) —Setting Height or Width to special syntax to divide available space into equal-sized regions or regions based on fixed ratios. A proportional-sized row or column grows and shrinks as the Grid is resized.

Grid - Sizing the Rows and Columns

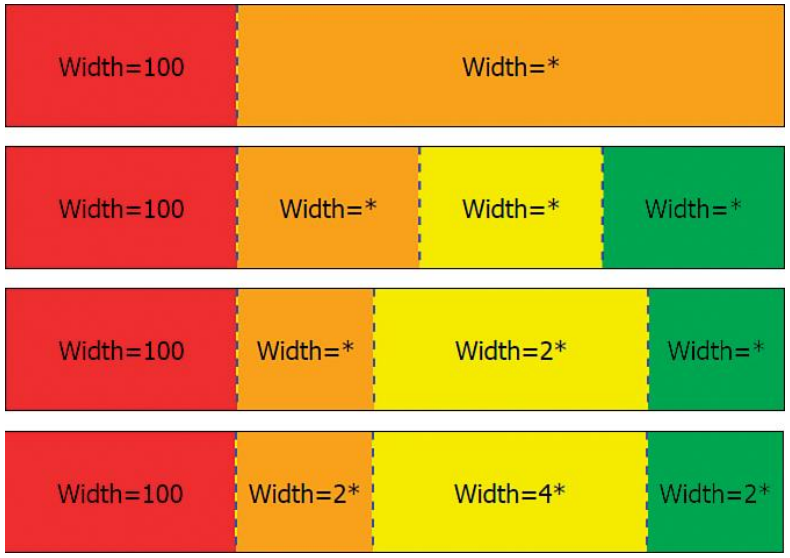
- Autosizing —making the first row and first column size to their content. This autosizing can be done by setting the appropriate RowDefinition's Height and ColumnDefinition's Width to the case-insensitive string Auto.

```
<!-- Define four rows: -->
<Grid.RowDefinitions>
  <RowDefinition Height="Auto"/>
  <RowDefinition/>
  <RowDefinition/>
  <RowDefinition/>
</Grid.RowDefinitions>

<!-- Define two columns: -->
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="Auto"/>
  <ColumnDefinition/>
</Grid.ColumnDefinitions>
```

Grid - Sizing the Rows and Columns

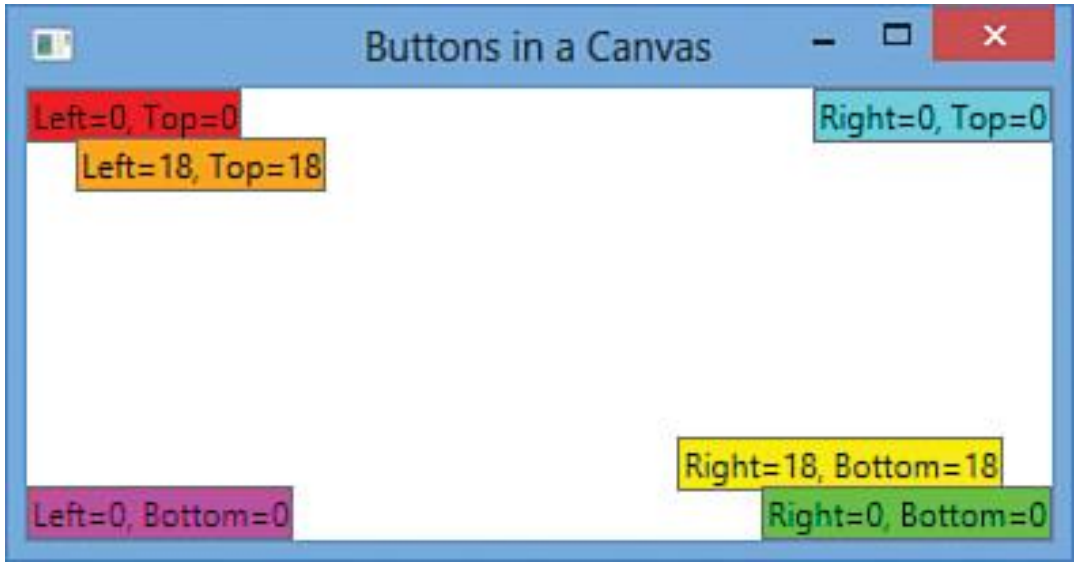
- Proportional sizing —It is done with star syntax that works as follows:
  - When a row's height or column's width is set to *, it occupies all the remaining space.
  - When multiple rows or columns use *, the remaining space is divided equally between them. The "remaining space" is the height or width of the Grid minus any rows or columns that use absolute sizing or autosizing.
  - Rows and columns can place a coefficient in front of the asterisk (like 2*) to take proportionately more space than other columns using the asterisk notation. A column with width 2* is always twice the width of a column with width * (which is shorthand for 1*) in the same Grid.



Canvas

- Canvas is a very basic panel. Only supports the “classic” notion of positioning elements with explicit coordinates.
- Position elements in a Canvas by using its attached properties: Left, Top, Right, and Bottom. In essence, you choose the corner in which to “dock” each element, and the attached property values serve as margins (to which the element’s own Margin values are added). If an element doesn’t use any of these attached properties (leaving them with their default value of Double.NaN), it is placed in the top-left corner of the Canvas (the equivalent of setting Left and Top to 0).

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        Title="Buttons in a Canvas">
  <Canvas>
    <Button Background="Red">Left=0, Top=0</Button>
    <Button Canvas.Left="18" Canvas.Top="18"
            Background="Orange">Left=18, Top=18</Button>
    <Button Canvas.Right="18" Canvas.Bottom="18"
            Background="Yellow">Right=18, Bottom=18</Button>
    <Button Canvas.Right="0" Canvas.Bottom="0"
            Background="Lime">Right=0, Bottom=0</Button>
    <Button Canvas.Right="0" Canvas.Top="0"
            Background="Aqua">Right=0, Top=0</Button>
    <Button Canvas.Left="0" Canvas.Bottom="0"
            Background="Magenta">Left=0, Bottom=0</Button>
  </Canvas>
</Window>
```



StackPanel

- StackPanel is a popular panel because of its simplicity and usefulness. As its name suggests, it simply stacks its children sequentially.
- With no attached properties for arranging children, only one way to customize the behavior of StackPanel—setting its Orientation property to Horizontal or Vertical. Vertical is the default Orientation.
- Example here shows simple Buttons, with no properties set other than Background and Content, in two StackPanels with only their Orientation set.

`Vertical` stacks elements from top to bottom.

`Horizontal` stacks elements from left to right.

DockPanel

- DockPanel enables easy docking of elements to an entire side of the panel, stretching it to fill the entire width or height. (unlike Canvas, which enables you to dock elements to a corner only.) DockPanel also enables a single element to fill all the remaining space unused by the docked elements.
- DockPanel has a Dock attached, so children can control their docking with one of four possible values: Left (the default when Dock isn't applied), Top, Right, and Bottom. The last child added to a DockPanel fills the remaining space unless DockPanel's LastChildFill property is set to false.

```
<DockPanel>
  <Button DockPanel.Dock="Top" Background="Red">1 (Top)</Button>
  <Button DockPanel.Dock="Left" Background="Orange">2 (Left)</Button>
  <Button DockPanel.Dock="Right" Background="Yellow">3 (Right)</Button>
  <Button DockPanel.Dock="Bottom" Background="Lime">4 (Bottom)</Button>
  <Button Background="Aqua">5</Button>
</DockPanel>
```