**(10) 패널을 이용한 레이아웃의 구성**

실제로 WPF 애플리케이션은 창안에 많은 UI들이 다양한 형태로 그룹화 시킬 수 있다. 뿐만 아니라 배치한 UI요소들의 창이 리사이징 될 때 전체 크기를 조절할 수 있고 아니면 부분적으로 크기를 조절할 수도 있다. 이러한 기능을 지원하기 위해서 WPF는 여러가지 패널들을 지원하고 있다.  
  
이전 장에서 언급했듯이 Window에는 단 하나의 개체만 추가할 수 있다. 만약 Window 개체에 바로 Button 개체가 달랑 추가되어 있다고 가정해보자. 이 때 창의 사이즈가 변경되어도 버튼의 크기는 변경되지 않고 같은 곳에 계속 위치하게 될 것이다.

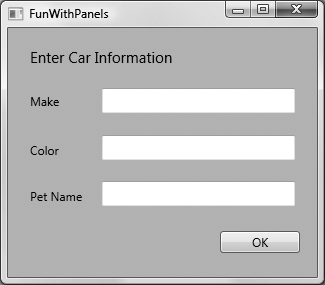
|  |
| --- |
| <!-- Window의 가운데에 출력되는 버튼 -->  <Window x:Class="MyWPFApp.MainWindow"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  Title="Fun with Panels!" Height="285" Width="325">  <Button Name="btnOK" Height = "100" Width="80">OK</Button>  </Window> |

만약 <Window> 개체 아래에 여러 개의 개체를 선언하게 되면 컴파일시에 마크업 에러가 발생하게 될 것이다. 그 이유는 앞에서 언급한 것처럼 Window 객체의 Content는 단 하나의 객체만 추가할 수 있기 때문이다.

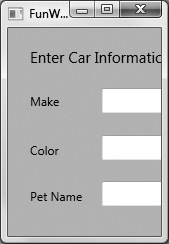
|  |
| --- |
| <!-- Content 개체 안에 여러 개체를 추가할 경우 에러가 발생하게 된다.-->  <Window x:Class="MyWPFApp.MainWindow"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  Title="Fun with Panels!" Height="285" Width="325">  <!--  두 개의 자식을 추가 -->    <Label Name="lblInstructions"  Width="328" Height="27" FontSize="15">Enter Car Information</Label>  <Button Name="btnOK" Height = "100" Width="80">OK</Button>  </Window> |

정리해 보자면 Window 타입은 단 하나의 객체만 참조할 수 있다는 것이고 만약 여러 개의 객체를 참조하고 싶다면 패널을 이용해야 한다는 것이다. 패널은 Window에 보여줄 UI 요소들을 모두 가지고 있을 것이고 패널을 위한 객체로는 Content 속성을 이용하게 될 것이다.  
  
  
**WPF의 기본 패널 컨트롤**  
  
System.Windows.Controls 네임스페이스는 다양한 패널 컨트롤들을 제공하고 있고 각각의 컨트롤들에서 자식 컨트롤들을 어떻게 배치시키는지 살펴보도록 하자. 패널을 이용하면 윈도우 크기를 조절할 때 UI 요소들을 어떻게 동작시킬 것인지를 설정할 수 있다.   
  
복잡한 사용자 인터페이스를 만들게 된다면 여러 개의 패널컨트롤들을 잘 섞어서 사용해야 한다. 게다가 패널 컨트롤은 ViewBox, TextBlock, TextFlow와 같은 문서적인 내용을 다루는 컨트롤들을 현재의 패널에 어떻게 배치할 것인지를 설정할 수 있다. 다음 [표29-3] 문서들은 WPF 패널 컨트롤들의 주 역할들에 대한 설명들을 보여주고 있다.

|  |  |
| --- | --- |
| Canvas | 디자인 시에 설정한 위치대로 항목들을 정확하게 배치시키는 가장 기본적인 패널이다. |
| DockPanel | 패널에서 지정한 방향으로 항목들을 위치시킨다. (Top, Bottom, Left, Right) |
| Grid | 표를 설정하여 그 표 안에 항목을 배치시키고 그 배율을 유지시킨다. |
| StackPanel | 가로 또는 세로 방향으로 항목들을 일렬로 정렬한다. |
| WrapPanel | 항목들을 왼쪽에서 오른쪽으로 순차적으로 배치시키고, 크기가 벗어나게 되면 다음 줄로 바꿔 배치한다 Orientation 속성 값에 따라 순서가 위에서 아래로 또는 오른쪽에서 왼쪽으로 지정할 수 있다. |

[표29-3] WPF의 기본 패널 컨트롤들  
  
그럼 이 패널들을 하나씩 자세하게 살펴보도록 하겠다. 이 패널들을 다루면서 다음 [그림29-16] UI들을 여러 패널들에 적용시켜 볼 것이고 창의 사이즈를 조절할 때 어떻게 변경되는지를 살펴보도록 하겠다.   
   
  
[그림29-16] 앞으로 살펴볼 UI 레이아웃  
  
  
**Canvas 패널 다루기**  
  
Canvas패널은 가장 단순하고 심플한 패널이다. Canvas 패널은 윈도우 폼 애플리케이션에서 다루던 패널과 거의 비슷하다고 느낄 수 있을 것이다. 간단하게 말해서 Canvas 패널은 UI 요소들을 절대적인 위치로 배치시킨다. 만약 사용자가 Canvas로 구성된 Window의 크기를 변경한다 하더라도 그 Canvas 안의 UI는 움직이지 않고 고정되어 있을 것이다.   
  
그럼 Canvas에 UI들을 추가하기 위해서 <Canvas> 태그를 열고 다음과 같은 요소들을 추가해 보도록 하자. 만약 Canvas를 UI 전체로 확장하고 싶다면 Height와 Width 속성을 지정하지 않으면 된다. 다음 [그림29-16]과 같은 UI를 만들기 위해서 다음과 같은 코드를 작성해 보도록 하자.

|  |
| --- |
| <Window  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  Title="Fun with Panels!" Height="285" Width="325">  <Canvas Background="LightSteelBlue">  <Button Canvas.Left="212" Canvas.Top="203" Name="btnOK" Width="80">OK</Button>  <Label Canvas.Left="17" Canvas.Top="14" Name="lblInstructions"  Width="328" Height="27" FontSize="15">Enter Car Information</Label>  <Label Canvas.Left="17" Canvas.Top="60" Name="lblMake">Make</Label>  <TextBox Canvas.Left="94" Canvas.Top="60" Name="txtMake"  Width="193" Height="25"/>  <Label Canvas.Left="17" Canvas.Top="109" Name="lblColor">Color</Label>  <TextBox Canvas.Left="94" Canvas.Top="107" Name="txtColor"  Width="193" Height="25"/>  <Label Canvas.Left="17" Canvas.Top="155" Name="lblPetName">Pet Name</Label>  <TextBox Canvas.Left="94" Canvas.Top="153" Name="txtPetName"  Width="193" Height="25"/>  </Canvas>  </Window> |

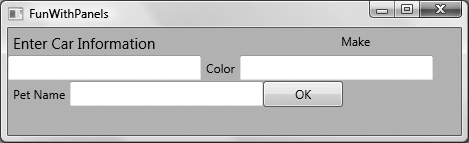
이 예제에서는 요소들 각각 앞에서 살펴보았던 첨부 속성을 이용해서 Canvas.Left와 Canvas.Top 값을 설정해주었고 이 요소들은 그 절대 좌표대로 패널에 위치될 것이다. 세로 위치는 Top과 Bottom 속성을 이용해서 설정할 수 있고 가로 위치는 Left와 Right 속성을 이용할 수 있다.   
  
이번 예제의 항목들은 <Canvas> 안에 배치되었기 때문에 창의 크기를 조절해 보면 크기는 고정되고 만약 창의 크기가 더 줄어들게 되면 다음 [그림29-17]처럼 잘리게 되는 것을 볼 수 있다.  
   
  
[그림29-17] Canvas를 이용한 절대 좌표  
  
캔버스에 선언되어 있는 순서는 좌표를 결정하는 것과는 무관하고 오로지 Canvas.Top, Canvas.Bottom, Canvas.Left, Canvas.Right 속성에 의해서 배치되게 된다. 만약 다음과 같이 마크업의 순서를 바꾼다 하더라도 동일한 화면을 볼 수 있을 것이다.

|  |
| --- |
| <Canvas Background="LightSteelBlue">  <TextBox Canvas.Left="94" Canvas.Top="153" Name="txtColor"  Width="193" Height="25"/>  <TextBox Canvas.Left="94" Canvas.Top="60" Name="txtPetName"  Width="193" Height="25"/>  <TextBox Canvas.Left="94" Canvas.Top="107" Name="txtMake"  Width="193" Height="25"/>  <Label Canvas.Left="17" Canvas.Top="14" Name="lblInstructions"  Width="328" Height="27" FontSize="15">Enter Car Information</Label>  <Label Canvas.Left="17" Canvas.Top="109" Name="lblColor">Color</Label>  <Label Canvas.Left="17" Canvas.Top="155" Name="lblMake">Pet Name</Label>  <Label Canvas.Left="17" Canvas.Top="60" Name="lblPetName">Make</Label>  <Button Canvas.Left="212" Canvas.Top="203" Name="btnOK" Width="80">OK</Button>  </Canvas> |

|  |
| --- |
| 만약 Canvas의 자식 개체에 첨부 속성을 이용한 위치가 지정되어 있지 않다면 그 개체는 제일 왼쪽 위에 위치하게 된다. |

Canvas 컨트롤이 자식 개체들을 정렬하는 패널에 있어서 기존부터 다뤄오던 개념과 비슷해서 선호할 수도 있지만 상당히 제한적인 부분이 많이 있다. 무엇보다도 만약 스타일이나 템플릿을 이용해서 폰트의 크기를 조절하고자 한다면 Canvas에서는 동적으로 사이즈를 조절하는 것이 불가능하다. 그리고 사용자가 창 크기를 Canvas 패널보다 작게 조절했을 경우에 그 자식 개체들은 잘리게 될 수도 있을 것이다.   
  
아마도 Canvas 클래스를 이용하는 최고의 상황은 그래픽 컨텐트를 배치하기 위한 용도가 될 것이다. 예를 들어 만약 XAML을 이용해서 그림판과 같이 이미지를 만드는 프로그램을 만든다면 선이나 도형 그리고 문자열과 같은 위치들이 동일하게 유지되어야 할 것이다. Canvas는 다음 장에서 그래픽 렌더링 서비스를 만들게 될 때 다시 한번 다루게 될 것이다.   
  
  
**WrapPanel 다루기**  
WrapPanel은 창의 크기가 변경에 따라서 컨텐트를 동적으로 위치시키게 된다. WrapPanel 안에 여러 개체들을 추가할 경우 Canvas처럼 Top, Bottom, Left, Right와 같은 도킹 값을 지정하지 않는다. 하지만 각각의 자식 개체들은 Height와 Width값을 자유롭게 지정할 수 있다.   
WrapPanel은 패널 특정 영역에 위치 값을 설정하지 않기 때문에 선언한 순서대로 배치되게 된다. 즉, 가자 위에 선언한 개체는 위쪽에 나타나는 것이고 가장 아래에 선언한 개체는 제일 뒤에 배치될 것이다. 그럼 다음과 XAML을 작성해보도록 하자.

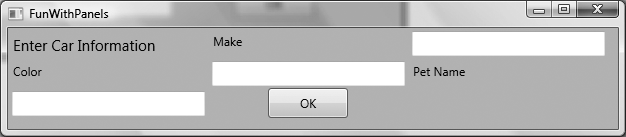
|  |
| --- |
| <WrapPanel Background="LightSteelBlue">  <Label Name="lblInstruction" Width="328"  Height="27" FontSize="15">Enter Car Information</Label>  <Label Name="lblMake">Make</Label>  <TextBox Name="txtMake" Width="193" Height="25"/>  <Label Name="lblColor">Color</Label>  <TextBox Name="txtColor" Width="193" Height="25"/>  <Label Name="lblPetName">Pet Name</Label>  <TextBox Name="txtPetName" Width="193" Height="25"/>  <Button Name="btnOK" Width="80">OK</Button>  </WrapPanel> |

마크업 코드를 보면 각각의 개체들에 Width와 Height를 지정해주었고 한번 실행해보면 각각의 개체들은 왼쪽에서부터 차례대로 출력되는 것을 볼 수 있다.   
   
  
[그림29-18] HTML 페이지처럼 배치되는 WrapPanel의 배치  
  
기본적으로 WrapPanel은 왼쪽에서 오른쪽으로 배치되지만 만약 Orientation 속성을 Vertical로 설정한다면 위에서 아래로 배치되는 것을 볼 수 있을 것이다.

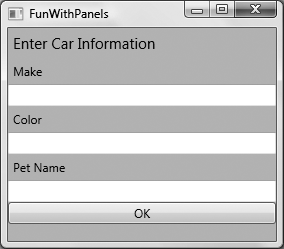
|  |
| --- |
| <WrapPanel Background="LightSteelBlue" Orientation ="Vertical"> |

WrapPanel 뿐만 아니라 다른 패널들은 ItemWidth와 ItemHeight 값을 설정해줄 수 있다. 이 값은 컨트롤의 기본 크기를 지정하는 것이다. 이 값은 자식 개체들의 기본 영역의 크기를 지정해 주기 위한 값이다. 다음 마크업 코드를 살펴보자.

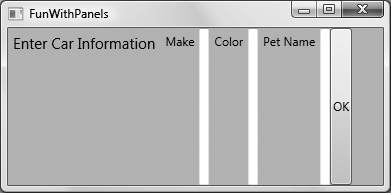
|  |
| --- |
| <WrapPanel Background="LightSteelBlue" ItemWidth ="200" ItemHeight ="30">  <Label Name="lblInstruction"  FontSize="15">Enter Car Information</Label>  <Label Name="lblMake">Make</Label>  <TextBox Name="txtMake"/>  <Label Name="lblColor">Color</Label>  <TextBox Name="txtColor"/>  <Label Name="lblPetName">Pet Name</Label>  <TextBox Name="txtPetName"/>  <Button Name="btnOK" Width ="80">OK</Button>  </WrapPanel> |

이 코드에서 Button의 Width를 80으로 지정하였지만 실제로 실행해 보면 다음 [그림29-19]처럼 기본 영역을 차지하고 있는 것을 볼 수 있다.   
   
  
[그림29-19] WrapPanel은 항목의 Hegiht와 Width를 지정해줄 수 있다.  
  
[그림29-19]에서 볼 수 있듯이 WrapPanel은 창의 사이즈가 자주 변경되는 창에서는 일반적으로 잘 사용하지 않을 것이다. WrapPanel을 다루기 위한 가장 좋은 방법은 자식으로 다른 패널을 추가한 후에 고정하고 싶은 개체들을 그 패널에 넣는 것이다. 그렇다면 사이즈가 조절되어도 다른 패널에 있는 요소의 위치는 변경되지 않을 것이다.  
  
  
**StackPanel 다루기**  
WrapPanel 컨트롤처럼 StackPanel 컨트롤은 개체들을 자동으로 정렬시시고 Orientation 속성을 이용해서 배치할 방향을 수평이나 수직으로 설정할 수 있다. 하지만 WrapPanel과의 차이점은 사용자가 창의 크기를 조절해도 StackPanel은 전혀 영향을 받지 않는다는 것이다. 오히려 StackPanel은 자식 개체들의 크기를 늘려 자신의 영역만큼 확장시킨다. 예를 들어 다음과 같은 XAML을 작성하여 출력해보면 [그림29-20]처럼 그려지는 것을 볼 수 있다.

|  |
| --- |
| <StackPanel Background="LightSteelBlue">  <Label Name="lblInstruction"  FontSize="15">Enter Car Information</Label>  <Label Name="lblMake">Make</Label>  <TextBox Name="txtMake"/>  <Label Name="lblColor">Color</Label>  <TextBox Name="txtColor"/>  <Label Name="lblPetName">Pet Name</Label>  <TextBox Name="txtPetName"/>  <Button Name="btnOK">OK</Button>  </StackPanel> |

   
[그림29-20] 수직으로 정렬시킨 StackPanel  
  
만약 Orientation 속성을 Horizontal로 설정한다면 다음 [그림29-21]처럼 출력되는 것을 볼 수 있을 것이다.

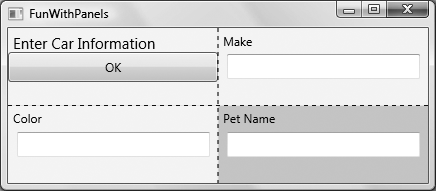
|  |
| --- |
| <StackPanel Background="LightSteelBlue" Orientation ="Horizontal"> |

   
[그림29-21] 수평으로 정렬시킨 StackPanel  
  
WrapPanel과 마찬가지로 StackPanel은 잘 사용하지 않을 것이다. StackPanel은 오히려 자식 패널을 가지고 있는 부모패널로 더 적당하다.  
  
  
**Grid 다루기**  
  
WPF에서 제공되는 모든 패널들 중에서 Grid는 굉장히 유용한 패널이다. HTML 테이블처럼 Grid는 셀을 나눌 수 있고 각각의 컨텐트들을 배치할 수 있다. 만약 Grid를 정의하고 싶다면 다음과 같은 순서대로 설정하면 된다.

|  |
| --- |
| 1. 각각의 열(Column)들을 정의한다.  2. 각각의 행(Row)들을 정의한다. 3. 첨부 속성 문법을 이용해서 각각의 셀에 특정 컨텐트를 할당한다. |

만약 어떤 행이나 열을 정의하지 않으면 <Grid>는 기본적은 하나의 셀로 동작된다.(행 1, 열 1) <Grid>에 자식 개체를 추가하지 않을 경우 (행 0, 열 0) 으로 설정된다.  
  
먼저 행과 열을 정의하는 이 단계는 <Grid.ColumnDefinitions>와 <Grid.RowDefinitions> 개체를 이용해서 설정해 주어야 한다. 그리고 그 개체는 각각 <ColumnDefinition> 개체들과 <RowDefinition> 개체들을 포함하고 있어야 한다. 각각의 셀(Cell) 안에는 실제로 닷넷 객체들이 할당되기 때문에 각각의 모양이다 객체 각각의 동작들을 정의하는 것이 가능하다. 다음 마크업 코드는 <Grid>에 UI 요소들을 배치하는 예제를 보여주고 있고 [그림29-2]는 이 코드를 실행했을 때의 화면을 보여주고 있다.

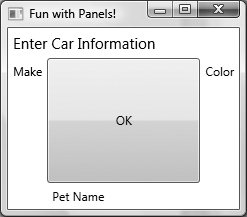
|  |
| --- |
| <Grid ShowGridLines ="True" Background ="AliceBlue">   <!-- 행/열 지정 -->   <Grid.ColumnDefinitions>     <ColumnDefinition/>     <ColumnDefinition/>   </Grid.ColumnDefinitions>   <Grid.RowDefinitions>     <RowDefinition/>     <RowDefinition/>   </Grid.RowDefinitions>   <!-- 그리드 셀에 개체들을 추가하자. -->   <Label Name="lblInstruction" Grid.Column ="0" Grid.Row ="0"   FontSize="15">Enter Car Information</Label>   <Button Name="btnOK" Height ="30" Grid.Column ="0" Grid.Row ="0" >OK</Button>   <Label Name="lblMake" Grid.Column ="1" Grid.Row ="0">Make</Label>   <TextBox Name="txtMake" Grid.Column ="1" Grid.Row ="0" Width="193" Height="25"/>   <Label Name="lblColor" Grid.Column ="0" Grid.Row ="1" >Color</Label>   <TextBox Name="txtColor" Width="193" Height="25" Grid.Column ="0" Grid.Row ="1" />   <!-- Just to keep things interesting, add some color to the pet name cell -->   <Rectangle Fill ="LightGreen" Grid.Column ="1" Grid.Row ="1" />   <Label Name="lblPetName" Grid.Column ="1" Grid.Row ="1" >Pet Name</Label>   <TextBox Name="txtPetName" Grid.Column ="1" Grid.Row ="1"   Width="193" Height="25"/> </Grid> |

각각의 개체들은 Grid.Row와 Grid.Column이라는 첨부 속성을 이용해서 자신의 행과 열을 지정해주게 된다. 만약 지정해주지 않을 경우 기본적으로 Grid.Column="0"과 Grid.Row="0"이 선언되기 때문에 제일 왼쪽 위에 배치된다는 것을 알아두자.  
   
  
[그림29-22] Grid 패널의 동작  
  
  
**Grid를 위한 GridSplitter 컨트롤**  
  
Grid 컨트롤은 또한 GridSplitter라는 컨트롤을 제공해주고 있다. 일반적으로 스플리터(Spliter)라는 컨트롤은 사용자가 행이나 열의 크기를 조절할 수 있는 기능을 제공해주고 있다. 때문에 셀 안에는 크기 조절이 가능한 개체가 있을 것이고 그 크기는 우리의 설정에 의해서 조절될 것이다. GridSplitter 컨트롤은 Grid를 위한 스플리터 컨트롤로 아주 쉽게 사용할 수 있다. GridSplitter 컨트롤은 첨부 속성을 이용해서 행이나 열을 지정할 수 있다. 여기서 알아 두어야 할 것은 Width와 Height 속성 중에 하나의 값은 반드시 지정해 주어야 한다는 것이다. 그럼 첫 번째 컬럼(Grid.Column = "0")에 스플리터를 적용해 보도록 하자.

|  |
| --- |
| <Window xmlns="[**http://schemas.microsoft.com/winfx/2006/xaml/presentation**](http://schemas.microsoft.com/winfx/2006/xaml/presentation)" xmlns:x="[**http://schemas.microsoft.com/winfx/2006/xaml**](http://schemas.microsoft.com/winfx/2006/xaml)" Title="FunWithPanels" Height="191" Width="436">   <Grid Background ="AliceBlue">     <!-- 기본컬럼 -->     <Grid.ColumnDefinitions>       <ColumnDefinition Width ="Auto"/>       <ColumnDefinition/>     </Grid.ColumnDefinitions>     <!-- 첫 번째 셀에 라벨추가 -->     <Label Name="lblLeft" Background ="GreenYellow"     Grid.Column="0" Content ="Left!"/>     <!-- 스플리터의 정의 -->     <GridSplitter Grid.Column ="0" Width ="5"/>     <!-- 두 번째 셀에 라벨 추가 -->     <Label Name="lblRight" Grid.Column ="1" Content ="Right!"/>   </Grid> </Window> |

먼저 스플리터를 가지고 있는 셀의 Width 속성은 Auto로 지정해 두었다는 것을 알아두자. 그리고 <GridSplitter>에 첨부속성을 이용해서 Column을 지정해 주었다. 이 코드를 실행시켜보면 5픽셀 크기의 스플리터가 보일 것이고 각각의 라벨들의 크기를 조절할 수 있을 것이다. 만약 Height와 Width 속성을 지정하지 않으면 셀 전체로 채워지게 된다는 것을 알아두자. [그림29-23]을 살펴보자.  
   
  
[그림29-23] Grid 컨트롤의 스플리터  
  
  
**DockPanel 다루기**  
  
DockPanel은 일반적으로 여러 패널들을 가지고 있는 마스터 패널로 주로 사용된다. DockPanel의 자식 개체들은 Canvas 컨트롤처럼 DockPanel의 첨부속성을 이용해서 자신의 방향을 지정하게 된다. 기본적으로는 제일 왼쪽 위에 위치하게 된다. 그럼 간단하게 다음과 같은 XAML을 작성한 후에 결과를 확인해 보도록 하자. 결과는 [그림29-24]와 같이 컨트롤들이 배치되는 것을 볼 수 있을 것이다.

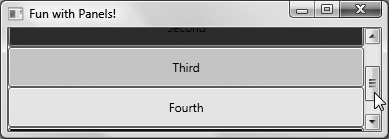
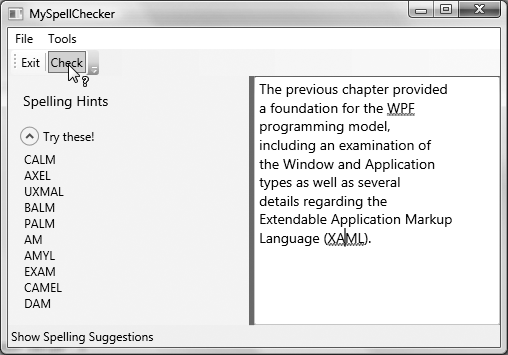
|  |
| --- |
| <DockPanel LastChildFill ="True">  <!-- 패¡¨￠널øI 안uE의C 개Æⓒø체¨u들ie -->  <Label DockPanel.Dock ="Top" Name="lblInstruction"  FontSize="15">Enter Car Information</Label>  <Label DockPanel.Dock ="Left" Name="lblMake">Make</Label>  <Label DockPanel.Dock ="Right" Name="lblColor">Color</Label>  <Label DockPanel.Dock ="Bottom" Name="lblPetName">Pet Name</Label>  <Button Name="btnOK">OK</Button>  </DockPanel> |

  
[그림29-4] 간단한 DockPanel

|  |
| --- |
| 만약 DockPanel에 같은 방향으로 여러 개체들이 추가하게 된다면 그 개체들은 지정한 방향에 선언되어 있는 순서대로 배치될 것이다. |

DockPanel 컨트롤의 장점은 사용자가 창의 크기를 조절할 때 각각의 자식 컨트롤들이 지정된 방향에 계속 위치시킬 수 있다는 것이다. 또한 DockPanel 컨트롤은 LastChildFill 속성을 지원해주고 있다. 만약 이 값을 true로 설정하면 그 자식 컨트롤이 DockPanel.Dock을 설정하지 않아도 그 컨트롤은 전체를 채우게 될 것이다.  
  
  
**패널 컨트롤에 스크롤 넣기**  
  
WPF에서는 <ScrollViewer> 개체를 지원해주고 있기 때문에 이 개체를 이용하면 자동으로 스크롤을 넣어 동작시킬 수 있다. 다음 XAML을 살펴보자.

|  |
| --- |
| <ScrollViewer>  <StackPanel>  <Button Content ="First" Background = "Green" Height ="40"/>  <Button Content ="Second" Background = "Red" Height ="40"/>  <Button Content ="Third" Background = "Pink" Height ="40"/>  <Button Content ="Fourth" Background = "Yellow" Height ="40"/>  <Button Content ="Fifth" Background = "Blue" Height ="40"/>  </StackPanel>  </ScrollViewer> |

이렇게 XAML을 작성한 후에 출력해보면 [그림29-25]와 같이 정의되는 것을 볼 수 있을 것이다.   
   
  
[그림29-25] ScrollViewer 개체의 적용  
  
예상했던 것처럼 각각의 패널 컨트롤들은 각각의 자식 개체들을 배치하기 위한 여러 멤버들을 제공하고 있다. 이와 관련해서 WPF 컨트롤은 Padding과 Margin이라는 속성을 지원하고 있다는 것을 알아두자. Padding 속성은 컨트롤 안쪽의 공간을 얼마나 확보할 것인지 결정하는 값인 반면에 Marin은 컨트롤 바깥쪽의 여백을 얼마나 확보할 것인지에 대한 값을 나타낸다.   
  
지금까지 WPF의 주요한 패널들을 살펴보았고 WPF는 다양한 방법으로 컨트롤들을 배치시킬 수 있다는 것을 보았을 것이다. 이제 다음에는 이러한 패널들을 이용해서 매인 윈도우 창을 만들어 볼 것이다. 여기서는 앞에서 스펠링 체크 프로그램과 같은 기능을 추가할 것이고 또한 메뉴, 상태바, 툴바와 같은 컨트롤들을 지정해 줄 것이다.  
  
  
**패널을 이용한 윈도우 프레임 구성하기**  
이번에 만들어 볼 프로그램은 기존에 만들었던 MySpellChecker 응용프로그램을 확장해서 보다 완벽하게 만들어 볼 것이다. 여기서는 기본적인 레이아웃을 구성할 것이고 기본적인 기능들을 구성해 볼 것이다.   
  
우리가 만들 프로그램의 구성을 살펴보자면 먼저 가장 위쪽에 메뉴와 툴바를 추가할 것이고 창 제일 아래에는 상태바를 추가할 것이다. 상태바는 사용자가 특정 메뉴 항목을 선택했을 때 알려줄 메시지들을 보여줄 것이고 메뉴나 툴바의 경우 애플리케이션을 닫거나 틀린 단어에 대한 추천 단어를 보여주는 기능을 제공할 것이다. 다음 [그림29-26]은 “XAML”이란 단어에 대한 추천 단어를 보여주고 있다.   
   
  
[그림29-26] 패널을 이용해서 설정한 윈도우의 UI   
  
툴바에서 제공하는 2개의 버튼은 이미지가 아닌 간단한 텍스트 문자로 보여주고 있다. 이 프로그램이 직접 학습용으로 만들어 보는 프로그램이기 때문에 텍스트를 이용했지만 리소스를 이용해서 버튼을 이미지로 보여줄 수 있다. 이 내용은 30장에서 보다 자세하게 살펴볼 것이다. 또한 마우스로 Check 버튼을 클릭하면 마우스 버튼 커서를 변경해줄 것이고 상태바에는 유용한 UI 메시지들을 보여줄 것이다.   
UI를 만들기 전에 <Grid> 대신 <DockPanel>을 추가해 보도록 하자.

|  |
| --- |
| <Window x:Class="MySpellChecker.MainWindow"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  Title="MySpellChecker" Height="331" Width="508"  WindowStartupLocation ="CenterScreen" >  <!-- This panel establishes the content for the window -->  <DockPanel>  </DockPanel>  </Window> |

**메뉴 구성하기**  
WPF에서는 Menu라는 컨트롤을 제공하고 있고 MenuItem이라는 개체들을 추가할 수 있다. XAML을 이용해서 메뉴를 구성할 경우 각각의 MenuItem에 서브 메뉴를 클릭할 때 발생하는 다양한 Click 이벤트들을 추가할 것이다. 이번 예제에서는 File과 Tools 라는 두 개의 메뉴를 추가할 것이고 Exit와 Spelling Hins라는 서브 메뉴를 추가할 것이다. 그리고 각각의 서브 메뉴에 Click 이벤트를 생성할 것이고 뒤에서 상태바를 설정할 때 사용하기 위한 MouseEnter와 MouseExit 이벤트 또한 구현할 것이다. 그럼 <DockPanel>에 다음과 같은 마크업 코드를 할당해 주도록 하자.

|  |
| --- |
| <Menu DockPanel.Dock ="Top"  HorizontalAlignment="Left" Background="White" BorderBrush ="Black">  <MenuItem Header="\_File" Click ="FileExit\_Click" >  <Separator/>  <MenuItem Header ="\_Exit" MouseEnter ="MouseEnterExitArea"  MouseLeave ="MouseLeaveArea" Click ="FileExit\_Click"/>  </MenuItem>  <MenuItem Header="\_Tools">  <MenuItem Header ="\_Spelling Hints" MouseEnter ="MouseEnterToolsHintsArea"  MouseLeave ="MouseLeaveArea" Click ="ToolsSpellingHints\_Click"/>  </MenuItem>  </Menu> |

우리는 메뉴에 DockPanel.Dock 설정으로 Top을 설정해 주었다. 그리고 <Separator> 개체를 Exit메뉴 위에 추가했기 때문에 메뉴위에 얇은 수평 라인을 볼 수 있을 것이다. 또한 각각의 MenuItem 개체의 Header 값을 보면 특정 단어 앞에 언더바를 지정해서 값을 할당해주고 있는 것을 볼 수 있을 것이다. 이것은 사용자가 Alt키를 이용할 경우에 단축키를 지정해 주기 위해서 이다.   
  
메뉴 구성을 마무리 하기 위해서 위에서 지정한 다양한 이벤트들을 구현해 보도록 하겠다. 먼저 File -> Exit 메뉴를 클릭했을 때 발생하는 FileExit\_Click() 메서드를 구현해 보도록 하자. 이 메서드에서는 Application.Current.Shutdown()을 이용해서 애플리케이션을 간단하게 종료시켜 주면 된다. 그리고 서브메뉴 각각 MouseEnter와 MouseExit 이벤트들을 구현하여 상태바를 업데이트 하는 부분은 뒷부분에서 추가하도록 하겠다. 마지막으로 Tools->Spelling Hints 메뉴를 클릭했을 때의 ToolsSpellingHints\_Click() 메서드 또한 뒷부분에서 구현하도록 하겠다. 지금은 다음과 같이 뼈대만 만들어 두도록 하자.

|  |
| --- |
| public partial class MainWindow : System.Windows.Window  {  public MainWindow()  {  InitializeComponent();  }  protected void FileExit\_Click(object sender, RoutedEventArgs args)  {  // 애uO플A리￢￠c케E이I션uC 종¨u료¤a  Application.Current.Shutdown();  }  protected void ToolsSpellingHints\_Click(object sender, RoutedEventArgs args)  {  }  protected void MouseEnterExitArea(object sender, RoutedEventArgs args)  {  }  protected void MouseEnterToolsHintsArea(object sender, RoutedEventArgs args)  {  }  protected void MouseLeaveArea(object sender, RoutedEventArgs args)  {  }  } |

**툴바 추가하기**  
WPF 에서 제공하고 있는 Toolbar 컨트롤은 메뉴를 대신해서 사용할 수도 있는 컨트롤이다. 그럼 다음과 같은 XAML을 <Menu> 뒤에 추가해 보도록 하자.

|  |
| --- |
| <ToolBar DockPanel.Dock ="Top" >  <Button Content ="Exit" MouseEnter ="MouseEnterExitArea"  MouseLeave ="MouseLeaveArea" Click ="FileExit\_Click"/>  <Separator/>  <Button Content ="Check" MouseEnter ="MouseEnterToolsHintsArea"  MouseLeave ="MouseLeaveArea" Click ="ToolsSpellingHints\_Click"  Cursor="Help" />  </ToolBar> |

이번에 추가한 툴바에서는 2개의 버튼을 가지고 있다. 이 버튼은 앞에서 만들었던 메뉴 버튼과 같은 이벤트를 지정해 주었다. 이렇게 두 개의 이벤트를 동시에 이용하는 것이 가능하다. 이 툴바가 일반적인 버튼을 이용하고 있기는 하지만 ToolBar는 ContentControl에서 파생된 컨트롤이기 때문에(“Is-a” 구조) 원하는 모양으로 마음대로 변경할 수 있다는 것을 알아두자. (콤보박스, 이미지 등) 이 컨트롤에서 다른 점은 Check 버튼은 Cursor 속성을 이용해서 마우스의 커서를 다르게 변경해 주고 있다는 것이다.

|  |
| --- |
| 툴바 컨트롤은 <ToolBarTray> 개체를 이용해서 툴바를 도킹하거나 드래그 드랍 하는 등의 동작을 구현할 수 있다. 보다 자세한 정보를 보고 싶다면 .NET Framework 3.5 SDK 문서를 참고하도록 하자. |

**상태바 구현하기**  
상태바는 <DockPanel>에서 가장 아래에 위치시킬 것이고 하나의 지금까지 다루어 본적이 없는 <TextBlock> 컨트롤을 추가할 것이다. 이 TextBlock 컨트롤은 TextBox처럼 특정 문자열을 고정시키기 위해서 사용되고 있고 진하게 하거나 밑줄을 긋는 것과 같은 여러 가지 효과들을 낼 수 있다. 상태바가 특별한 기술을 필요로 하고 있지 않지만 여기서 사용한 TextBlock 타입은 작은 문장을 나타내거나 할 경우에 여러 방면으로 유용한 컨트롤이다. 그럼 툴바 아래에 다음과 같은 XAML을 추가해보도록 하자.

|  |
| --- |
| <StatusBar DockPanel.Dock ="Bottom" Background="Beige" >  <StatusBarItem>  <TextBlock Name="statBarText">Ready</TextBlock>  </StatusBarItem>  </StatusBar> |

이렇게 작성하고 비주얼 스튜디오 디자이너를 열어보면 [그림29-27]과 같은 화면을 볼 수 있을 것이다.   
  
   
[그림29-27] 스펠링 검사UI  
  
  
**디자인 UI의 마무리**  
그럼 마지막으로 지금까지의 UI에 2개의 컬럼을 가지고 있는 Grid와 스플리터를 추가해 보도록 하겠다. 왼쪽에는 추천 단어 리스트를 보여주는 Expander 컨트롤을 만든 후에 <StackPanel>을 이용해서 정렬시킬 것이다. 오른쪽에는 멀티라인들을 지원하는 TextBox 컨트롤을 만들고 스펠링 체크 속성을 지정해줄 것이다. 그리고 <Grid> 컨트롤은 <DockPanel> 안에서 왼쪽에 위치 되도록 설정할 것이다. 그럼 다음 XAML을 추가해 보도록 하자.

|  |
| --- |
| <Grid DockPanel.Dock ="Left" Background ="AliceBlue">   <!-- 행/열의 정의 -->   <Grid.ColumnDefinitions>     <ColumnDefinition />     <ColumnDefinition />   </Grid.ColumnDefinitions>   <GridSplitter Grid.Column ="0" Width ="5" Background ="Gray" />   <StackPanel Grid.Column="0" VerticalAlignment ="Stretch" >     <Label Name="lblSpellingInstructions" FontSize="14" Margin="10,10,0,0">       Spelling Hints     </Label>     <Expander Name="expanderSpelling" Header ="Try these!" Margin="10,10,10,10">       <!-- 프로그램에서 추가할 리스트-->       <Label Name ="lblSpellingHints" FontSize ="12"/>     </Expander>   </StackPanel>   <!-- 문장들을 작성할 텍스트 박스 -->   <TextBox Grid.Column ="1"   SpellCheck.IsEnabled ="True"   AcceptsReturn ="True"   Name ="txtData" FontSize ="14"   BorderBrush ="Blue">   </TextBox> </Grid> |

**마지막 코드**  
이렇게 해서 UI 디자인을 모두 끝냈다. 그럼 이제 남은 일은 UI에서 지정했던 이벤트들을 작성하는 것이다. 여기서는 주석을 이용해서 코드 설명을 대신하도록 하겠다.

|  |
| --- |
| public partial class MainWindow : System.Windows.Window {     protected void ToolsSpellingHints\_Click(object sender, RoutedEventArgs args)     {         string spellingHints = string.Empty;         // 현재 Caret의 위치로 에러를 가져온다.          SpellingError error = txtData.GetSpellingError(txtData.CaretIndex);         if (error != null)         {             // 제안하는 단어 만들기             foreach (string s in error.Suggestions)             {                 spellingHints += string.Format("{0}\n", s);             }             // Expander안에 있는 Label에 제안 단어 보여주기             lblSpellingHints.Content = spellingHints;             // Expander의 확장             expanderSpelling.IsExpanded = true;         }     }     protected void MouseEnterExitArea(object sender, RoutedEventArgs args)     {         statBarText.Text = "Exit the Application";     }     protected void MouseEnterToolsHintsArea(object sender, RoutedEventArgs args)     {         statBarText.Text = "Show Spelling Suggestions";     }     protected void MouseLeaveArea(object sender, RoutedEventArgs args)     {         statBarText.Text = "Ready";     } } |

자, 여기까지 프로그램 코드를 작성해 보았다. 여기에 몇 줄의 코드를 더 추가하면 우리는 더 많은 기능을 제공해 줄 수 있을 것이다. 이 기능을 추가하기 위해서 컨트롤 명령(Control Command)에 대해서 이해하고 있어야 하기 때문에 이 내용에 대해서 살펴보도록 하자.