1. WPF 기초 2
01. WPF 맛보기 3
02. XAML 8
03. 컨텐츠 모델 13
2. 레이아웃 19
01. Canvas 패널 20
02. StackPanel 23
03. WrapPanel 27
04. DockPanel 29
05. Grid 31
3. 사용자 인터페이스 39
01. 마우스 이벤트 40
02. 키보드 이벤트 45
03. 라우트된 이벤트 47
4. 컨트롤 52
01. 기본 컨트롤 53
02. 리스트 형태의 컨트롤 66
5. 데이터 바인딩 72
01. 일반적인 UI의 데이터 다루기 73
02. 데이터 바인딩을 이용한 UI 다루기 89

<장제목>

1. WPF 기초

</장제목>

WPF(Windows Presentation Foundation)는 MS 의 새로운 프리젠테이션 프레임워크로 기존 프레임워크에서 제공하던 GDI, GDI+, HTML 등을 포함할 뿐만 아니라 다양하고 새로운 여러 기능과 프레임워크를 제공한다.

WPF 의 주요 특징을 살펴보면 컨텐츠라 부르는 여러 요소들(컨트롤, 텍스트, 그래픽 등)을 하나의 모델로 통합하여 관리할 수 있다. 이들은 내부적으로 DirectX 기반으로 개발되어 DirectX 기술의 좋은 특징을 가질 뿐만 아니라 외형적으로는 DirectX 기반인지 알지 못하며 WinForm 과 같은 다른 .Net 환경의 기술들처럼 사용하기 쉽고 생산성 높게 개발할 수 있다.

WPF 는 컨텐츠를 각각의 레이아웃 알고리즘으로 자동 배열하는 컨테이너 요소를 제공하며 이러한 컨테이너 요소는 컨텐츠 종류에 상관없이 자동으로 훌륭하게 레이아웃 구성할 수 있도록 돕는다. 또 컨텐츠를 시각화하기 위해 데이터 바인딩, 컨트롤 템플릿, 애니메이션을 지원한다.

WPF 는 3D 환경에 컨텐츠를 표현하기 위해 3D 와 관련된 카메라, 모델, 조명, 텍스처, 변형 등 여러 요소들을 제공한다. 2D, 3D 그래픽뿐만 아니라 스트리밍 기능, 비디오 기능, 플로 문서 기능 등 다양한 기능들을 WPF 는 자체적으로 제공한다.

<절제목>

01. WPF 맛보기

</절제목>

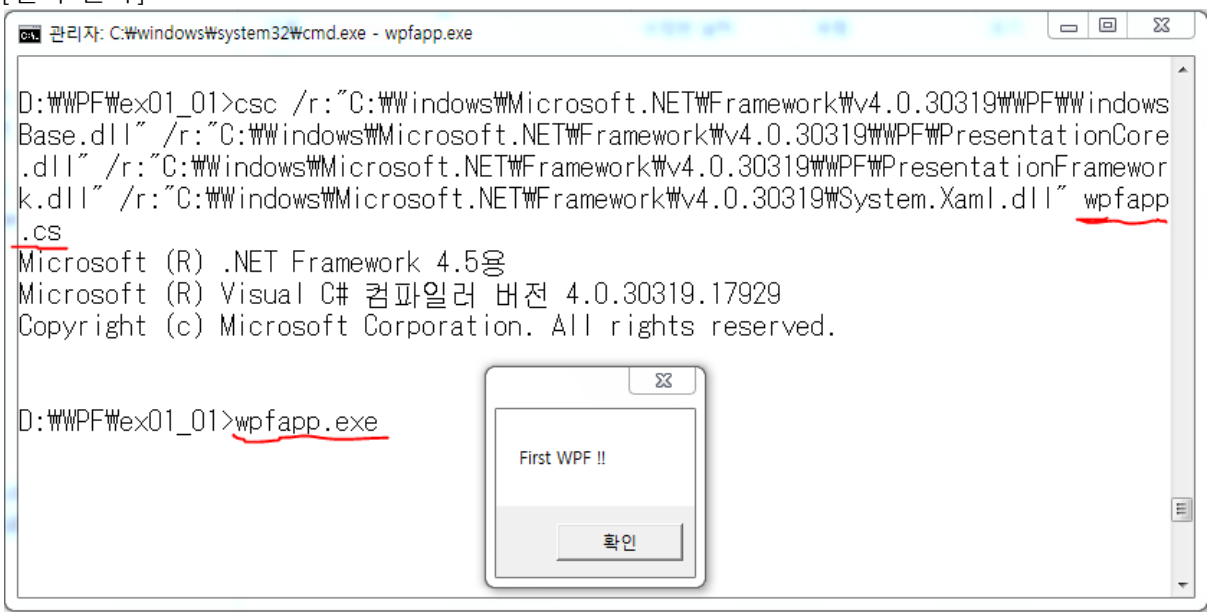
다음 예제01-01은 가장 처음으로 시작하는 WPF 시작 프로그램으로 메시지 박스를 출력한다.

<코드>
[예제 01-01] WPF 시작 프로그램

```
using System;
using System.Windows; //WPF 최상위 네임스페이스

namespace ex01_01
{
    class WpfApp
    {
        static void Main()
        {
            MessageBox.Show(" First WPF !! ");
        }
    }
}
```

</코드>

<출력>
[출력 결과]



</출력>

WPF 응용프로그램을 만들기 위해 WPF 주요 어셈블리 WindowsBase.dll(WPF API 의 가장 기본적인 형식들의 정의로 스레딩, 보안, 변환기, 종속 속성, 라우트된 이벤트 등의 기능), PresentationCore.dll(GUI 레이아웃과 관련된 형식 정의), PresentationFramework.dll(런타임에 XAML 문서를 읽고 쓰는 기본 기능과 Application 과 Window 클래스가 상호 작용 기능)와 System.Xaml.dll(XAML 과 관련된 프로그래밍 기능으로 .Net Framework 4.0 이상 버전에서는 포함)를 참조 추가해야 하며 System.dll 은 csc.rsp 에 의해 자동으로 추가되었다.

사실 위 예제는 응용프로그램 컴파일을 위한 예제로 WPF 의 기능을 사용하지 않았다. 다음 예제는응용프로그램의 뼈대를 이루는 WPF 의 Application 객체와 기본 UI 요소인 Window 객체를 사용한 WPF 응용프로그램이다.

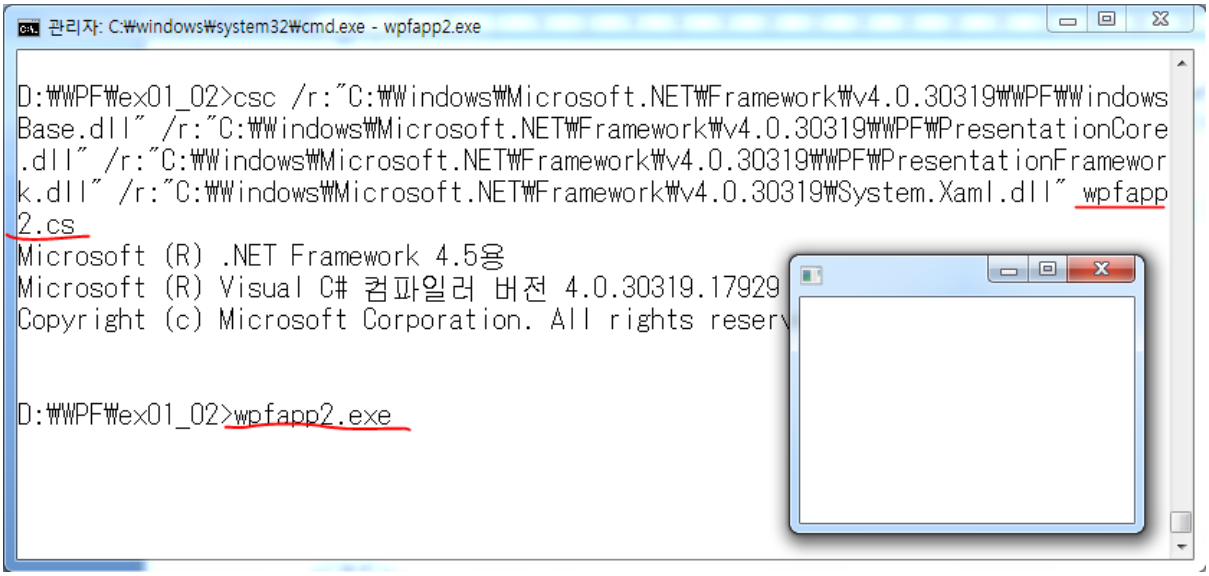
<코드>
[예제 01-02] WPF의 Application, Window 객체 사용

```
using System;
using System.Windows;
namespace ex01_02 {

   class MyApp : Application {

      [STAThread]
      static void Main(string[] args) {
         MyApp app = new MyApp();
         app.Startup += app.MyApp_StartingUp;
         app.Run();
      }

      void MyApp_StartingUp(object sender, StartupEventArgs e)
      {
         Window window = new Window();
         window.Show();
      }
   }
}
```

</코드>

WPF 에서 응용프로그램이란 app 객체 자체를 말하며 app 객체는 싱글톤 객체로 응용프로그램 서비스를 담당한다. app.Startup 이벤트는 app.Run 이 호출될 때 발생하는 이벤트이며 이때 window 객체를 생성하고 화면에 보인다. window 객체는 자동으로 WPF 응용프로그램의 주 윈도우이며 최상위 운도우로 응용프로그램 객체의 MainWindow 속성에 대입된다. 여기서는 간단한 실행 예만을 보이고 뒤쪽에서 다시 공부한다. [STAThread]는 COM 컴포넌트와 관련된 특성(attribute)로 단일 쓰레드 UI 작업과 호환되도록 초기화한다는 특성이다.

<결과>
[출력 결과]



</결과>

결과 실행 방법은 이전과 같다.

다음 예제는 단순한 버튼 컨트롤을 윈도우 위에 위치시킨 예제로 버튼의 이벤트를 처리하여 메시지 박스를 뛰우도록 하였다.

<코드>
[예제 01-03]

```
using System;
using System.Windows;
using System.Windows.Controls;

namespace ex01_03 {

    class MyWindow : Window
    {
        public MyWindow()
        {
            Button btn = new Button();
            btn.Click += btn_Click;
            btn.Width = 100;
            btn.Height = 25;
            btn.Content = "클릭";

            AddChild(btn);

            Title = "WPF Window";
        }
        void btn_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("버튼 클릭!");
        }
    }

    class MyApp : Application {
```

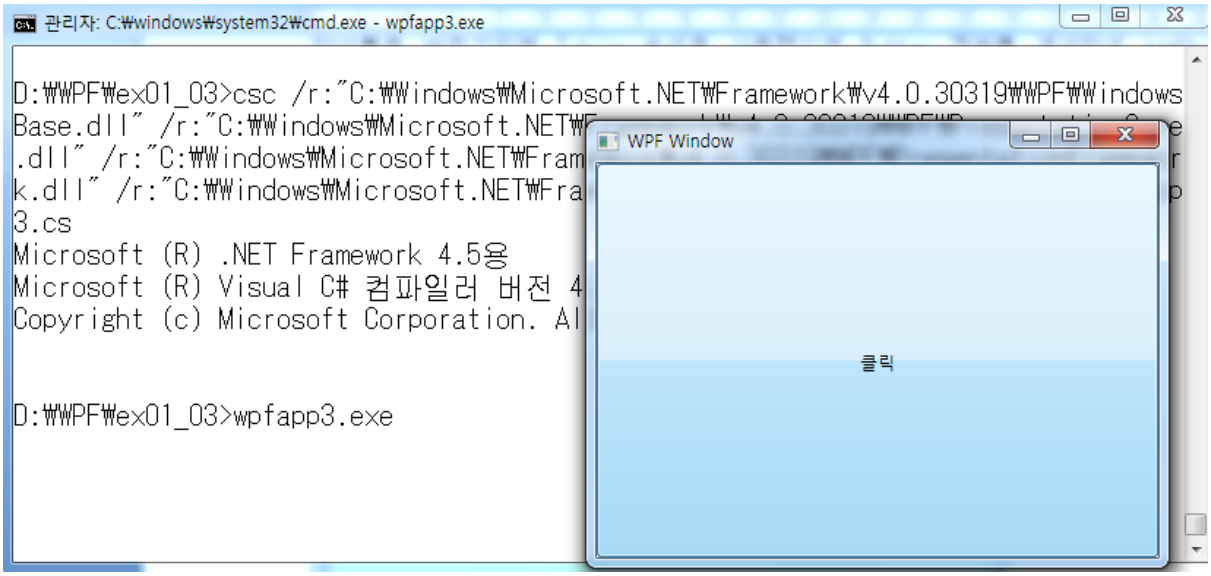
```
        [STAThread]
        static void Main(string[] args) {
            MyApp app = new MyApp();
            app.Startup += app.MyApp_StartingUp;
            app.Run();
        }

        void MyApp_StartingUp(object sender, StartupEventArgs e)
        {
            MyWindow window = new MyWindow();
            window.Show();
        }
    }
}
```
</코드>

<결과>
[출력 결과]

</결과>

윈도우의 기능을 확장하기위하여 Windows 클래스를 상속받아 MyWindow 클래스를 만들고 타이틀을 바꾸기위해 Titile 속성을 사용했으며 Button 객체를 생성하여 AddChlid() 메소드로 자식 요소로 선택한다. 한 가지 눈여겨 볼 부분은 Button 의 크기를 설정할 뿐 버튼의 위치는 설정하지 않았지만 Window 의 정 중앙에 놓인다. 이것은 컨텐츠 모델과 레이아웃 시스템에 대해 설명하는 장에서 다루게 된다.

만약 Button 에 폭과 높이를 설정하지 않는다면 어떻게 될까?
콘텍츠 모델의 기본 동작에 의해서 아래 그림처럼 버튼이 윈도우에 꽉 차게 그려진다.


관리자: C:₩windows₩system32₩cmd.exe - wpfapp3.exe
D:₩WPF₩ex01_03>csc /r:"C:₩Windows₩Microsoft.NET₩Framework₩v4.0.30319₩WPF₩WindowsBase.dll" /r:"C:₩Windows₩Microsoft.NET₩
.dll" /r:"C:₩Windows₩Microsoft.NET₩Fram
k.dll" /r:"C:₩Windows₩Microsoft.NET₩Fra
3.cs
Microsoft (R) .NET Framework 4.5용
Microsoft (R) Visual C# 컴파일러 버전 4
Copyright (c) Microsoft Corporation. Al
D:₩WPF₩ex01_03>wpfapp3.exe
WPF Window
클릭

<절제목>

02. XAML

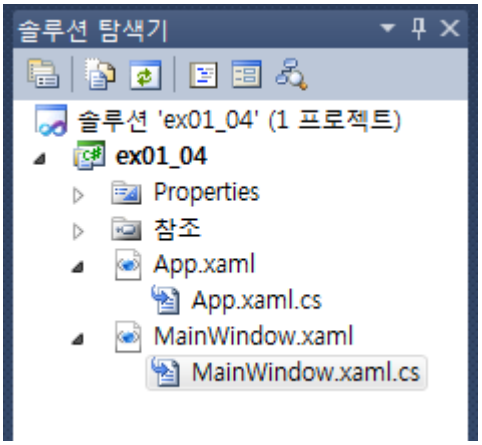
</절제목>

일반적으로 프로그램 개발은 사용자 인터페이스(UI) 작업과 프로그램 로직 작업으로 나눌 수 있는데 UI 작업은 그래픽 디자이너가 담당하고 프로그램 로직은 프로그래머가 담당하여 협업을 통해 프로그램 개발을 진행한다. 또한 이렇게 나누어 작업하면 서로 독립적으로 작업을 진행할 수 있고 프로그램 유연성을 높여 프로그램 생산성이 좋아진다. 많은 프레임워크나 프로그램 언어는 위와 같은 작업이 용이하도록 발전해 왔다.

WPF 는 이러한 협업(UI 와 프로그램 로직)을 쉽게 개발할 수 있는 프로그램 모델을 제공한다. UI 작업은 대부분 XAML 이란 선언적 형태의 언어를 사용하고 프로그램 로직은 .Net 프로그램 언어를 사용하여 작업할 수 있다. XAML 은 XML 을 기반의 언어이다. 또한 WPF 는 UI 작업을 XAML 뿐만 아니라 .Net 프로그램 언어로도 작업할 수 있으며 XAML 은 XAML 컴파일러를 이용하여 이미 각각 매핑되어 정의한 .Net 형식(type)으로 변환되어 결과물인 .Net 바이너리(.dll, .exe)로 결합된다.

한마디로 WPF 에서는 프로그램 로직상에서 Window 와 같은 클래스 인스턴스를 직접 생성하여 조작할 수 있지만 모든 인스턴스를 XAML 을 사용하여 생성하고 프로그램할 수 있으며 특히 UI 와 관련된 작업은 XAML 을 사용하여 개발하는 것이 좀 더 이해하기 쉽고 프로그램 생산성을 높이는데 유리하다.

다음은 Visual Studio 2010 에서 기본적으로 생성한 WPF 프로젝트로 앞쪽 예제에서 생성했던 Window 클래스의 정의 일부(MainWindow.xaml)와 Application 클래스의 정의 일부(App.xaml)를 XAML 코드를 저장하는 파일 목록을 보여준다. 나머지 클래스 정의는 C# 비하인드 코드(App.xaml.cs, MainWindow.xaml.cs)에 partial 클래스로 구현되어 있다.



다음 예제는 주 윈도우 클래스 MainWindow 의 XAML 코드 파일로 MainWindow.xaml 의 코드다.

<코드>
[예제 01-04] MainWindow.xaml 파일

```
<Window x:Class="ex01_04.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>

    </Grid>
</Window>
```

</코드>

Window 클래스의 일부분을 XAML 로 정의한 예제이다.

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

은 XML 문법의 기본 네임 스페이스이며

"http://schemas.microsoft.com/winfx/2006/xaml/presentation" 를 기본 네임 스페이스로 사용한다는 의미다.(기본 네임 스페이스는 XAML 코드에서 접두어 없이 클래스나 속성, 이벤트 등을 사용한다.)

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

는 XML 문법의 "http://schemas.microsoft.com/winfx/2006/xaml" 을 'x' 라는 네임 스페이스로 XAML 코드에서 사용한다는 것이다. (클래스, 속성, 이벤트 등에 x 네임스페이스를 사용해야 한다.)

x:Class="ex01_04.MainWindow"

속성은 이 Window 요소(여기서 요소는 XML 용어다.)가 ex01_04.MainWindow 의 partial 클래스의 일부 정의라는 것을 나타내는 속성이다.

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

속성은 이 Window 속성의 Title, Height, Width 의 속성 값을 의미한다.

Grid 요소는 panel 이라고 부르는 컨테이너 요소로 레이아웃 시스템 설명에서 자세히 다룬다.

다음은 응용 프로그램 Application 클래스를 XAML 로 구현한 코드로 UI 와 관련된 형식 외에도 XAML 코드로 표현할 수 있음을 볼 수 있다.

<코드>
[예제 01-05] App.xaml 파일

```
<Application x:Class="ex01_04.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

</코드>

여기서 주요 속성이 StarupUri 로 이 응용 프로그램의 주 윈도우 클래스의 XAML 코드 파일(StartupUri="MainWindow.xaml")을 지정한다.

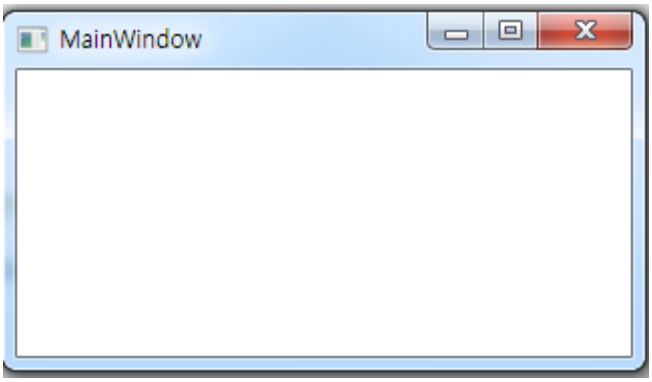
x:Class="ex01_04.App" 속성은 이 Application 요소가 ex01_04.App 클래스의 partial 클래스 정의를 XAML 로 표현했다는 것을 나타낸다. Application.Resources 요소는 XAML 문법 중 속성 요소 문법으로 다음에 설명한다.

다음 예제는 Application 클래스와 MainWindow 클래스의 C# 코드로 나머지 partial 클래스를 보인 것이다. 주요 내용이 모두 XAML 에 구현되어 아무 내용도 없는 것을 볼 수 있다.

<코드>
[예제 01-06]

```
namespace ex01_04
{
    public partial class App : Application
    {
    }
}
namespace ex01_04
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

</코드>
<결과>
[출력 결과]



</결과>

VisualStudio 2010 에서 작성 및 빌드해서 실행한 기본 WPF 응용 프로그램이다. InitializeComponet()는 XAML 에 정의된 Window 클래스에 정의된 값들을 사용하여 윈도우 속성을 초기화 한다. 또 Application XAML 코드는 외형적으로 프로그램 시작점인 Main() 함수가 없는 것처럼 보이지만 obj 폴더의 App.g.cs 에 다음과 같이 정의 되어 있다.

```
[System.STAThreadAttribute()]
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
        public static void Main() {
```

```
            ex01_04.App app = new ex01_04.App();
            app.Run();
        }
```

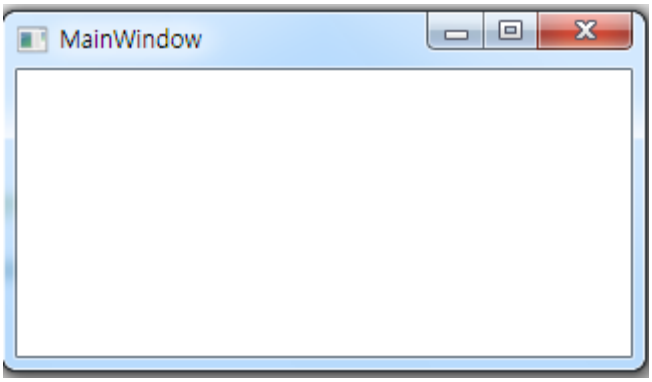
C# 코드에서 윈도우(MainWindow) 인스턴스를 직접 생성하고 싶다면 아래 예제와 같이 직접 코드를 작성할 수 있다.

<코드>
[예제 01-07] App.xaml 파일과 App.xaml.cs 파일

```
<Application x:Class="ex01_07.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             Startup="App_Startup">
    <Application.Resources>

    </Application.Resources>
</Application>
```

```
namespace ex01_07
{
    public partial class App : Application
    {
        private void App_Startup(object sender, StartupEventArgs e)
        {
            Window win = new MainWindow();
            win.Show();
        }
    }
}
```

</코드>
<결과>
[출력 결과]



</결과>

Application 요소의 Startup 이벤트에서 App_Startup 핸들러 메소드를 지정하고 비하인드 코드에서 윈도우 인스턴스를 생성하고 보이도록 구현하였다.

다음 예제는 Application 클래스의 XAML 을 제거하고 직접 Main 함수에서 Application 객체를 생성하고 Startup 이벤트를 이용하여 Window 객체를 C# 코드로 생성하는 예제다.

<코드>
[예제 01-08]

```
namespace ex01_08
{
    public partial class App : Application
    {
        [STAThread]
        private static void Main(string[] args)
        {
            App app = new App();
            app.Startup += new StartupEventHandler(app.App_Startup);
            app.Run();
        }
        private void App_Startup(object sender, StartupEventArgs e)
        {
            Window win = new MainWindow();
            win.Show();
        }
    }
}
</코드>
```

결과는 위와 같고 주 윈도우와 관련된 XAML 코드와 비하인드 코드는 그대로 이용한다. 또 코드의 new StartupEventHandler 대리자는 생략가능하다.

XAML 에 대한 설명도 프로그램을 진행하며 하나하나 설명할 것이므로 이 절에서는 XAML 이 C# 코드와 어떻게 어울려 WPF 응용프로그램을 생성하는지 정도만 알고 있으면 되겠다.

<절제목>

03. 컨텐츠 모델

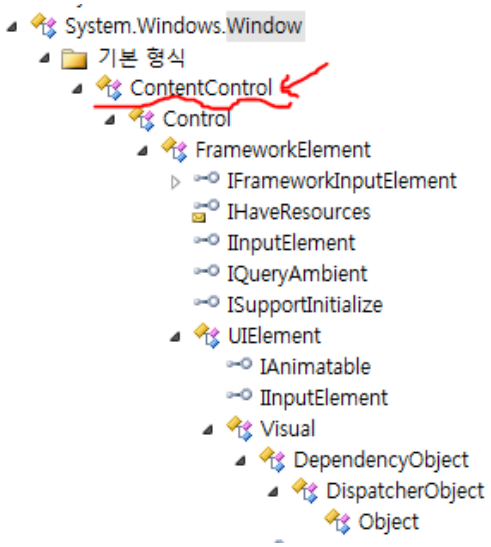
</절제목>

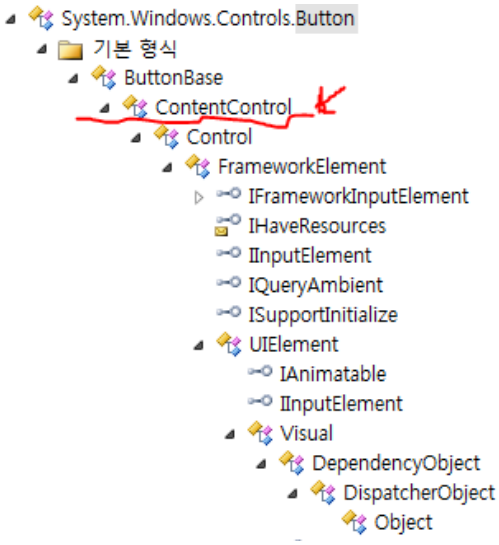
WinForm 과 같은 컨트롤들은 각 컨트롤이 수행하는 '동작'과 '외관'이 정해져 있으며 이런 컨트롤의 외관을 변경하는 것은 간단한 문자열을 변경하거나 비트맵을 그리는 수준이였다. 만약 사용자가 원하는 수준의 복잡한 외관을 원할 경우는 사용자 정의 컨트롤을 직접 작성하고 개발해야 하는 어려움이 있었다.

WPF 에서는 사용자 정의 컨트롤을 사용하지 않고도 컨트롤의 외관을 아~주 유연하게 변경할 수 있는데 이 특징의 핵심이 컨텐츠 모델이며 WPF 의 대부분의 컨트롤(ContentControl 로부터 파생된 요소)은 Content 라는 속성을 가지며 이 속성의 형식이 object 형식이므로 일반 문자열이나 .Net 객체라면 어떤 데이터든지 컨텐츠로 가질 수 있다. 이 말은 곧 WPF 의 컨트롤의 외관은 무한하게 변경 가능하다는 것을 의미한다. ContentControl 로부터 파생한 요소는 컨텐츠를 가질 수 있으며 단 하나의 컨텐츠만을 갖는다. 단 하나의 컨텐츠만을 갖는다 하더라도 다음에 공부할 Panel 을 사용해 여러 컨텐츠를 갖는 효과를 만들 수 있기 때문에 무수히 많은 컨텐츠를 가질 수 있다고 보아도 된다. 또 컨텐츠로 포함된 요소가 ContentControl 로부터 파생한 컨텐츠 컨트롤이라면 또 무수히 많은 컨텐츠를 포함할 수 있으므로 무한한 컨텐츠를 포함한다는 것이다.

마지막으로 WPF 는 사용자 정의 컨트롤을 제공할 뿐만 아니라 컨트롤 템플릿이라는 개념을 제공하며 다양하게 컨트롤을 변형할 수 있다.

다음은 Window 클래스와 Button 클래스 모두 ContentControl 로 부터 파생되므로 컨텐츠 컨트롤이라는 것을 보여주는 클래스 상속 구조 그림다.



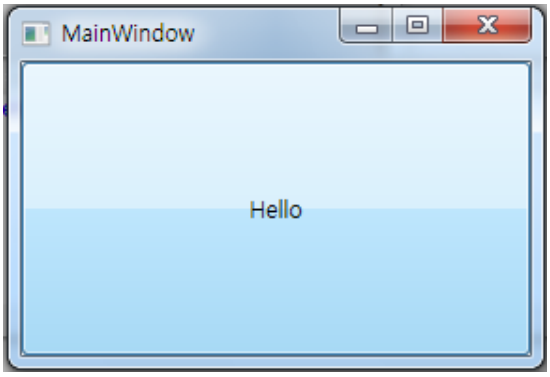


다음 예제는 Button의 컨텐츠로 "Hello" 문자열을 갖는 코드다.

<코드>
[예제 01-09] Button의 컨텐츠 문자열

```
<Window x:Class="ex01_09.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">
    <Button Content="Hello">
    </Button>
</Window>
```

</코드>
<결과>
[출력 결과]



</결과>

Button 의 Content 속성을 사용하여 XAML 에서 문자열 "Hello"를 컨텐츠로 지정했다. XAML 상에서는 Content 속성을 사용하지 않고 아래 예제처럼 요소의 데이터로 가질 수 있다. 같은 코드이며 같은 결과를 볼 수 있다.

<코드>
[예제 01-10] Button 요소의 데이터로 컨텐츠를 표현

```
<Window x:Class="ex01_09.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">
    <Button>
        Hello
    </Button>
</Window>
</코드>
```

혹은 XAML 문법 중 속성 요소 문법을 사용하여 아래처럼 구현할 수 있다.

```
<코드>
[예제 01-11] 속성 요소 문법으로 Button의 컨텐츠 표현
<Window x:Class="ex01_09.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">
    <Button>
        <Button.Content>
            Hello
        </Button.Content>
    </Button>
</Window>
</코드>
<결과>
[출력 결과]
</결과>
```

사실 지금은 이렇게 쓰는 것이 복잡하므로 쓸 필요가 없지만 다른 속성에 또 다른 요소를 표현하거나 여러 요소를 표현하고자 할 때 사용할 수 있는 문법으로 XAML 에서 구조를 단순화하기 위해 제공되는 문법이다. 또 하나의 사실은 Window 요소(요소(element)는 XML 용어다)도 ContentControl 로부터 파생된 컨텐츠 컨트롤이므로 Button 을 컨텐츠로 가지고 있다.

위 코드를 속성 요소 문법을 사용하면 아래처럼 쓸 수 있다.

```
<코드>
[예제 01-12] Window의 컨텐츠는 Button
<Window x:Class="ex01_09.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">
    <Window.Content>
        <Button>
            <Button.Content>
                Hello
            </Button.Content>
        </Button>
    </Window.Content>
</Window>
</코드>
```

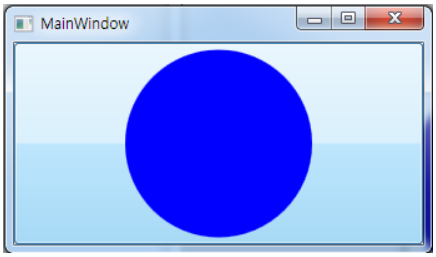
결과는 같고 단지 속성 요소 문법인 Window.Content 요소를 사용한 것 뿐이다.

다음은 버튼 컨트롤이 Ellipse 요소를 컨텐츠로 갖는 예제다.

<코드>
[예제 01-13] Ellipse 요소를 컨텐츠로 갖는 버튼

```
<Window x:Class="ex01_13.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">
    <Button>
        <Ellipse Fill="Blue" Width="150" Height="150" />
    </Button>
</Window>
```

</코드>
<결과>
[출력 결과]



</결과>

위 예제는 다음과 같이 속성 요소 문법으로 표현할 수 있으며 문자열처럼(Content=”Hello”) 요소의 특성(XML 문법 용어이다)으로 단순하게 표현할 수는 없다.

<코드>
[예제 01-14] Ellipse 요소를 컨텐츠로 갖는 버튼 2

```
<Button>
        <Button.Content>
            <Ellipse Fill="Blue" Width="150" Height="150" />
        </Button.Content>
</Button>
```

</코드>

다음 예제처럼 버튼의 컨텐츠를 C# 코드로 작성할 수 있지만 상당히 복잡하다는 것을 알 수 있다. 보통 이런 룩앤필은 XAML 로 구현하는 것이 간단하고 변경에 유연하다.

<코드>
[예제 01-15] Ellipse 요소를 C# 코드로 작성

```
namespace ex01_15
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();


            Ellipse ellipse = new Ellipse();
```

```
            ellipse.Fill = new SolidColorBrush(Colors.Blue);
            ellipse.Width = 150;
            ellipse.Height = 150;

            Button btn = new Button();
            btn.Content = ellipse;

            mainWindow.Content = btn;
        }
    }
}
</코드>
```

결과는 위와 같고 단지 C# 코드로 작성한 것뿐이다. 여기서 mainWindow 는 XAML 코드에서 작성한 Window 요소의 참조다. C# 코드에서는 mainWindow.Content = btn 이나 mainWindow.AddChild(btn) 모두 가능하다. 두 멤버 모두 ContentControl 이며 지금 기능은 같다. 일반적으로 대부분의 컨텐츠 컨트롤들은 AddChlid() 멤버 함수를 상속과정에서 멤버 접근 한정자를 사용하여 사용하지 못하도록 막으며 Window 와 같은 요소들만 AddChild() 멤버 함수를 사용할 수 있도록 한다. 그래서 대부분 Content 속성을 사용하게 된다.

다음 예제는 XAML 코드에서 C# 코드에서 사용할 Window 요소의 참조를 만들기 위해 Name 속성을 사용한 것이다.

<코드>
[예제 01-16] Window 요소의 Name 속성

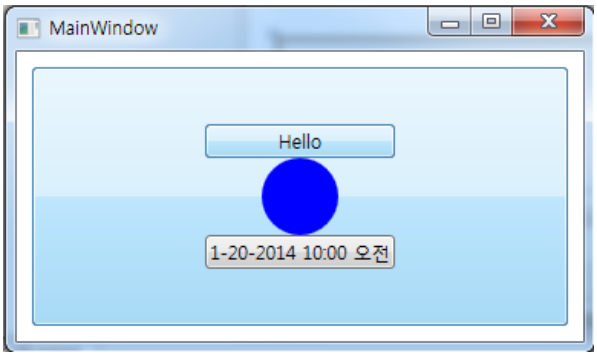
```
<Window x:Class="ex01_15.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525" Name="mainWindow">
</Window>
</코드>
```

다음은 몇 가지 요소를 컨텐츠로 가지는 버튼 예제다.

<코드>
[예제 01-17] 여러 요소를 컨텐츠로 가지는 버튼

```
<Window x:Class="ex01_17.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">
    <Button Margin="10">
        <StackPanel>
            <Button>
                Hello
            </Button>
            <Ellipse Fill="Blue" Width="50" Height="50" />
            <Button xmlns:sys="clr-namespace:System;assembly=mscorlib">
                <sys:DateTime>2014/1/20 10:00:00</sys:DateTime>
            </Button>
        </StackPanel>
    </Button>
```

```
</Window>
```
</코드>
<결과>
[출력 결과]



</결과>

Button 요소는 하나의 컨텐츠만을 가질 수 있으므로 여러 요소를 StackPanal 이라는 요소를 컨텐츠로 갖는다. StackPanel 은 세 요소(버튼 두 개와 DateTime)를 자식 요소로 갖는다. DateTime 은 mscorlib.dll 어셈블리에 있는 .Net 형식이므로 형식을 접근하기 위한 xmlns:sys="clr-namespace:System;assembly=mscorlib" 특성을 사용했다. 이처럼 WPF 는 WPF API 뿐만 아니라 모든 .Net API 를 활용할 수 있다. StackPanel 요소는 여러 자식 요소를 가질 수 있는 요소로 ContentControl 은 아니며 요소들을 자신만의 정렬 알고리즘으로 정렬하기 위한 레이아웃 요소다. StackPanel 은 레이아웃을 공부할 때 자세히 공부한다.

<장제목>

2. 레이아웃

</장제목>

WPF 의 레이아웃 시스템은 아주 유연하고 강력한 기능을 제공한다. WPF 는 패널(Panel)이라고 부르는 기본적인 하지만 강력한 레이아웃 요소를 제공한다. 이 패널들은 자신만의 기본적인 레이아웃 시스템(정렬 규칙)을 가지고 있으며 패널은 자식 요소를 가질 수 있고 또 자식 요소들은 컨텐츠나 또 다른 패널을 가질 수 있으므로 강력한 레이아웃을 구성할 수 있다.

다음 표는 핵심 패널을 정리한 것이다. Canvas 와 Grid 는 자식 요소들을 겹칠 수 있지만(Z-Order 가 존재: 어떤 요소가 위쪽에 오며 뒤쪽에 오는지) 다른 패널은 자식요소들이 겹쳐지지 않는다.

패널	특징	자식 요소 겹침
Canvas	가장 기본적인 패널로 자식 요소를 원하는 위치에 배치할 수 있다.	O
StackPanel	가로나 세로 방향으로 자식 요소를 일렬로 정렬한다.	X
WrapPanel	StackPanel 과 비슷하지만 자식 요소를 왼쪽에서 오른쪽으로 차례로 배치하며 크기를 벗어나면 다음 줄에 배치한다.	X
DockPanel	각 지정 방향에 자식 요소를 배치한다.	X
Grid	가장 강력한 패널로 표 형태로 자식 요소를 배치한다.	O

<절제목>

01. Canvas 패널

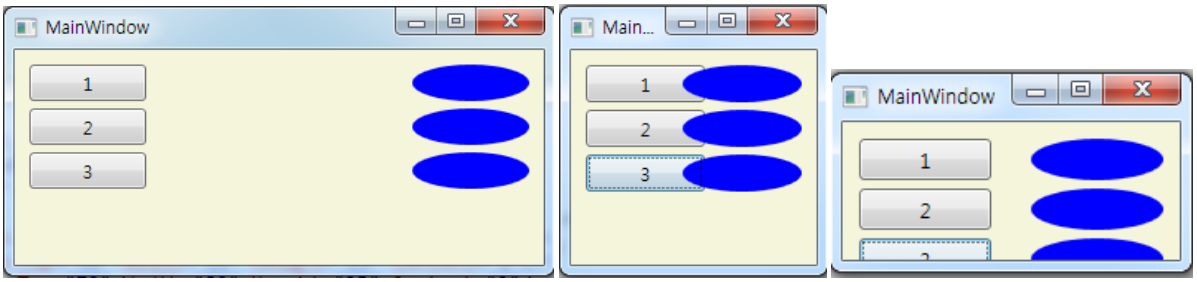
</절제목>

가장 기본적인 패널로 자식 요소를 원하는 위치(절대 위치)에 배치한다. 사용하기 가장 단순하다. Canvas 패널은 윈폼의 그리기 영역과 가장 유사한 패널로 일반적으로 그래픽 콘텍스를 배치(그리기)하고자 할 때 유용하게 사용할 수 있다. 핵심 속성은 Canvas.Left, Canvas.Top, Canvas.Right, Canvas.Bottom 이다.

<코드>
[예제 02-01] Canvas 패널

```
<Window x:Class="ex02_01.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">
    <Canvas Background="Beige">
        <Button Canvas.Left="10" Canvas.Top="10" Width="80" Height="25" Content="1"/>
        <Button Canvas.Left="10" Canvas.Top="40" Width="80" Height="25" Content="2"/>
        <Button Canvas.Left="10" Canvas.Top="70" Width="80" Height="25" Content="3"/>
        <Ellipse Canvas.Right="10" Canvas.Top="10" Width="80" Height="25" Fill="Blue"/>
        <Ellipse Canvas.Right="10" Canvas.Top="40" Width="80" Height="25" Fill="Blue"/>
        <Ellipse Canvas.Right="10" Canvas.Top="70" Width="80" Height="25" Fill="Blue"/>
    </Canvas>
</Window>
```

</코드>
<결과>
[출력 결과]



</결과>

결과는 각각 크기를 줄였을 때의 윈도우를 보인 것이다. 두 번째 그림에서 Button 은 왼쪽, 위를 기준으로 Ellipse 요소는 오른쪽, 위를 기준으로 배치되어 있으므로 각각을 기준으로 자식 요소들이 크기에 따라 배치되며 XAML 코드에서 Button 보다 Ellipse 를 아래에 정의하여 Ellipse 가 화면 위쪽에 보이는 것을 볼 수 있다. Z-Order 는 속성을 사용하여 변경 가능하다. 또 위쪽을 기준으로 배치되어 있으므로 자식 요소들이 윈도우가 작아지면 아래 자식 요소들이 잘려 보이는 모습을 볼 수 있다.

자식 요소는 Width 와 Height 을 결정하고 부모의 어떤 곳에 놓일 것인지를 Canvas.Left, Canvas.Top 속성을 이용하여 지정한다.(WPF 는 크기 단위로 장치 독립적인 픽셀만을 사용하여 1 pixel 은 1/96 인치로 고정되어 있다.)

잘 보면 알겠지만 자식 요소 Button 에서(<Button Canvas.Left="10" Canvas.Top="10" Width="80" Height="25" Content="1"/>)는 부모 요소의 속성인 Canvas.Left 와 Canvas.Top 을 사용한다는 것을 알 수 있다. 이 문법을 XAML 의 결합 속성 문법이라 부르며 WPF 의 의존 속성으로 가능한 것이다.

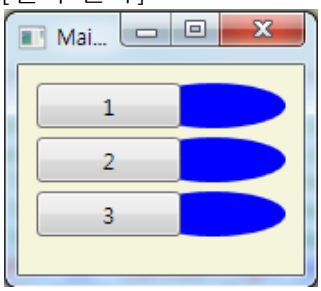
WPF 요소의 대부분의 속성은 의존 속성으로 정의되어 있으며 기존에 알고 있는 .Net 속성과는 다른 개념으로 정의된다. 간단하게 의존 속성은 여러 가지(데이터 바이딩, 스타일, 에니메이션 등) WPF 만의 기능을 구축하기 위해 새로 만들어낸 속성 개념이다. 다음에 또 공부하게 된다.

다음은 Panel.ZIndex 속성을 사용하여 Z-Order 를 변경한 예제다.

<코드>
[예제 02-02] Z-Order 변경하기

```
<Window x:Class="ex02_01.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">
    <Canvas Background="Beige">
        <Button Panel.ZIndex="2" Canvas.Left="10" Canvas.Top="10" Width="80" Height="25" Content="1"/>
        <Button Panel.ZIndex="2" Canvas.Left="10" Canvas.Top="40" Width="80" Height="25" Content="2"/>
        <Button Panel.ZIndex="2" Canvas.Left="10" Canvas.Top="70" Width="80" Height="25" Content="3"/>
        <Ellipse Panel.ZIndex="1" Canvas.Right="10" Canvas.Top="10" Width="80" Height="25" Fill="Blue"/>
        <Ellipse Panel.ZIndex="1" Canvas.Right="10" Canvas.Top="40" Width="80" Height="25" Fill="Blue"/>
        <Ellipse Panel.ZIndex="1" Canvas.Right="10" Canvas.Top="70" Width="80" Height="25" Fill="Blue"/>
    </Canvas>
</Window>
```

</코드>
<결과>
[출력 결과]



</결과>

결과처럼 Button 이 Ellipse 보다 위쪽에 보인다. Button 이 Ellipse 보다 Panel.ZIndex 가 높기 때문이다.

다음은 Ellipse 자식 요소를 아래쪽을 기준으로 Canvas 에 배치했을 때의 예제다.

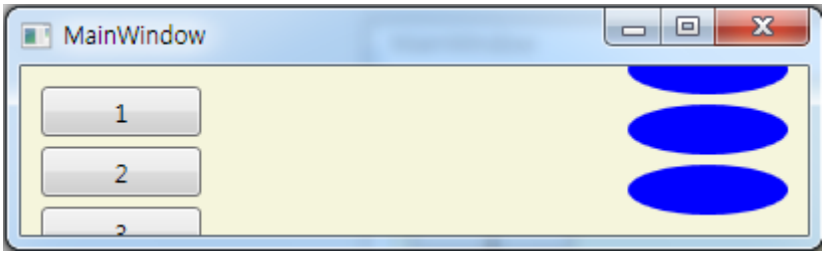
<코드>
[예제 02-03] Ellipse 요소를 Canvas.Bottom을 기준으로 배치

```
<Window x:Class="ex02_01.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">
    <Canvas Background="Beige">
        <Button Canvas.Left="10" Canvas.Top="10" Width="80" Height="25" Content="1"/>
        <Button Canvas.Left="10" Canvas.Top="40" Width="80" Height="25" Content="2"/>
        <Button Canvas.Left="10" Canvas.Top="70" Width="80" Height="25" Content="3"/>
        <Ellipse Canvas.Right="10" Canvas.Bottom="10" Width="80" Height="25" Fill="Blue"/>
        <Ellipse Canvas.Right="10" Canvas.Bottom="40" Width="80" Height="25" Fill="Blue"/>
        <Ellipse Canvas.Right="10" Canvas.Bottom="70" Width="80" Height="25" Fill="Blue"/>
    </Canvas>
</Window>
```

</코드>

<결과>

[출력 결과]



</결과>

Button 과 Ellipse 자식 요소를 각각 위와 아래쪽 Canvas 를 기준으로 배치했으므로 윈도우가 작아지면 Button 은 아래쪽 자식 요소가 잘려 보이며 Ellipse 는 위쪽 자식 요소가 잘려 보인다.

<절제목>

02. StackPanel

</절제목>

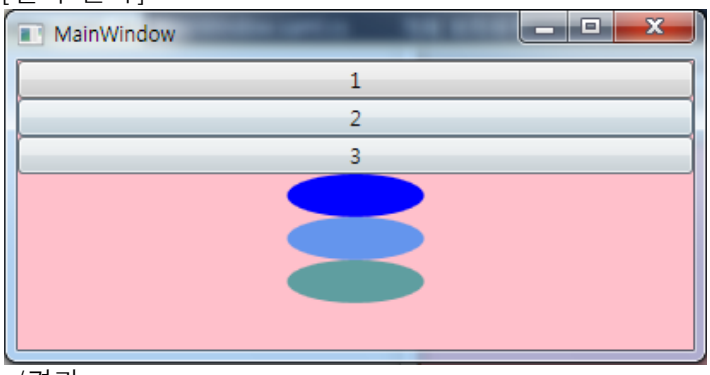
StackPanel 은 자식 요소를 행이나 열로 자동 배치하는 패널이다. 보통 다른 패널과 결합되어 많이 사용된다.

다음은 Button 세 개와 Ellipse 세 개를 StackPanel 에 배치한 예제다.

<코드>
[예제 02-04]

```
<Window x:Class="ex02_04.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">
    <StackPanel Background="Pink">
        <Button Content="1"/>
        <Button Content="2"/>
        <Button Content="3"/>
        <Ellipse Width="80" Height="25" Fill="Blue"/>
        <Ellipse Width="80" Height="25" Fill="CornflowerBlue"/>
        <Ellipse Width="80" Height="25" Fill="CadetBlue"/>
    </StackPanel>
</Window>
```

</코드>
<결과>
[출력 결과]



</결과>

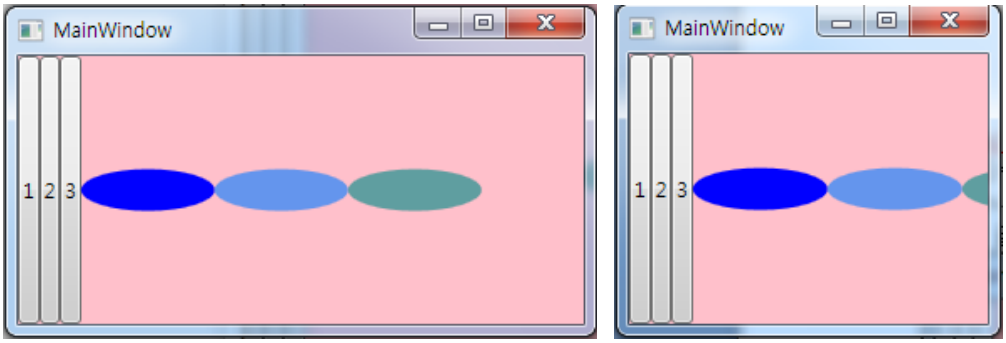
StackPanel 은 버튼과 Ellipse 를 수직으로 자동으로 배치한다. 여기서 Button 은 크기(Width, Height)을 설정하지 않았는데 Button 은 크기를 설정하지 않으면 Width 가 꽉 차게 배치되며 Height 은 Button 의 컨텐츠에 맞추어 배치된다. 대부분의 컨트롤은 크기를 설정하지 않으면 자신이 부모 요소와 자식 요소 컨텐츠에 따라 어떻게 배치될 것인지 기본 동작을 가진다. Ellipse 는 컨트롤이 아니므로 크기를 설정해야 하고 이때 그 크기에 맞게 StackPanel 에 배치된다. StackPanel 의 경계를 구분하기 위해 Background 에 핑크색을 설정했다.

Orientation 속성을 사용하여 수평 패널로 변경할 수 있으며 다음 예제는 수평 StackPanel 예제다.

<코드>
[예제 02-05] StackPanel의 수평으로 배치된 자식 요소

```
<StackPanel Background="Pink" Orientation="Horizontal">
        <Button Content="1"/>
        <Button Content="2"/>
        <Button Content="3"/>
        <Ellipse Width="80" Height="25" Fill="Blue"/>
        <Ellipse Width="80" Height="25" Fill="CornflowerBlue"/>
        <Ellipse Width="80" Height="25" Fill="CadetBlue"/>
</StackPanel>
```

</코드>
<결과>
[출력 결과]



</결과>

수직으로 배치할 때와 반대로 Button 의 Height 은 StackPanel 에 꽉 차게 배치되며 Width 는 Button 의 컨텐츠에 맞게 배치된다. Ellipse 는 같다.

두 번째 결과는 StackPanel 의 자식 요소보다 윈도우가 작았을 때 자식 요소가 잘려나가는 것을 볼 수 있다.

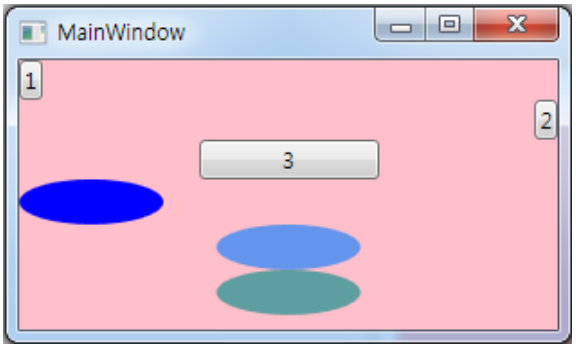
자식 요소에 HorizontalAlignment, VerticalAlignment 속성을 설정하면 부모 패널과의 남는 공간에 대한 배치를 다르게 할 수 있다.

다음 예제는 Button 에 HorizontalAlignment 속성을 사용한 예제다.

<코드>
[예제 02-06]

```
<StackPanel Background="Pink">
        <Button HorizontalAlignment="Left"    Content="1"/>
        <Button HorizontalAlignment="Right" Content="2"/>
        <Button Width="100" Content="3"/>
        <Ellipse HorizontalAlignment="Left" Width="80" Height="25" Fill="Blue"/>
        <Ellipse Width="80" Height="25" Fill="CornflowerBlue"/>
        <Ellipse Width="80" Height="25" Fill="CadetBlue"/>
</StackPanel>
```

</코드>
<결과>
[출력 결과]



</결과>

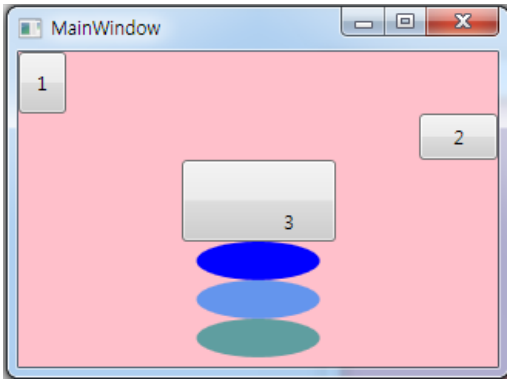
1 번 Button 은 왼쪽으로 배치되며 컨텐츠의 크기에 맞추어 Width 가 설정된다. 2 번 Button 은 오른쪽에 배치되며 또한 컨텐츠의 크기에 맞추어 Width 가 설정된다. 3 번 Button 은 중앙에 디폴트로 배치되며 Width 는 설정한 크기 100 이다. Ellipse 는 첫 번째를 제외한 모두 디폴트 중앙에 배치된다.

Padding 속성을 사용하면 요소와 컨텐츠 간의 여백(안쪽 여백)을 결정할 수 있다. 다음은 Padding 속성을 사용한 예제다.

<코드>
[예제 02-07] Padding 속성을 사용한 안쪽 여백

```
<StackPanel Background="Pink">
        <Button Padding="10" HorizontalAlignment="Left"    Content="1"/>
        <Button Padding="20, 5" HorizontalAlignment="Right" Content="2"/>
        <Button Padding="50,30,10,3" Width="100" Content="3"/>
        <Ellipse Width="80" Height="25" Fill="Blue"/>
        <Ellipse Width="80" Height="25" Fill="CornflowerBlue"/>
        <Ellipse Width="80" Height="25" Fill="CadetBlue"/>
</StackPanel>
```

</코드>
<결과>
[출력 결과]



</결과>

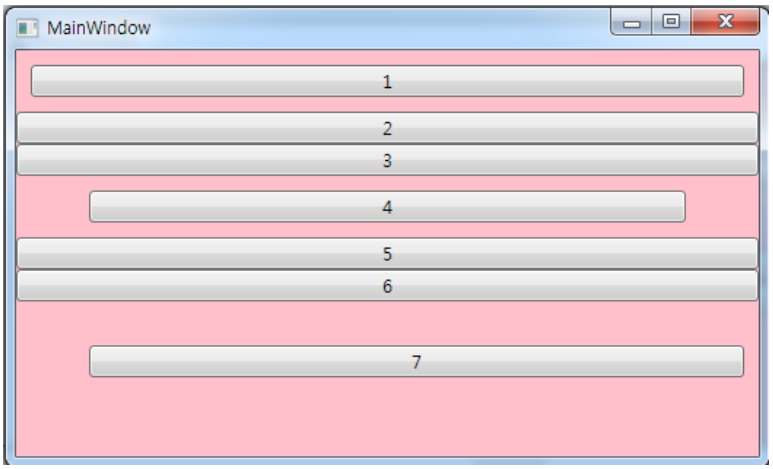
1 번 Button 은 Left, Top, Right, Bottom 의 모든 안쪽 여백을 10 으로 설정한다. 2 번 Button 은 Left, Right 를 20 으로 Top, Bottom 을 5 로 설정한다. 3 번 Button 은 Left, Top, Right, Bottom 을 각각 50, 30, 10, 3 으로 설정한다.

Padding 과 비슷하게 Margin 을 사용하여 요소들 간의 여백(바깥쪽 여백)을 설정할 수 있다. 다음 예제는 Margin 을 사용한 예제다.

<코드>
[예제 02-08]

```
<Button Margin="10" Content="1"/>
<Button Content="2"/>
<Button Content="3"/>
<Button Margin="50, 10" Content="4"/>
<Button Content="5"/>
<Button Content="6"/>
<Button Margin="50, 30, 10, 3" Content="7"/>
```

</코드>
<결과>
[출력 결과]



</결과>

1 번 Button 은 Left, Top, Right, Bottom 의 모든 바깥쪽 여백을 10 으로 설정한다. 4 번 Button 은 Left, Right 를 50 으로 Top, Bottom 을 10 로 설정한다. 7 번 Button 은 Left, Top, Right, Bottom 을 각각 50, 30, 10, 3 으로 설정한다.

<절제목>

03. WrapPanel

</절제목>

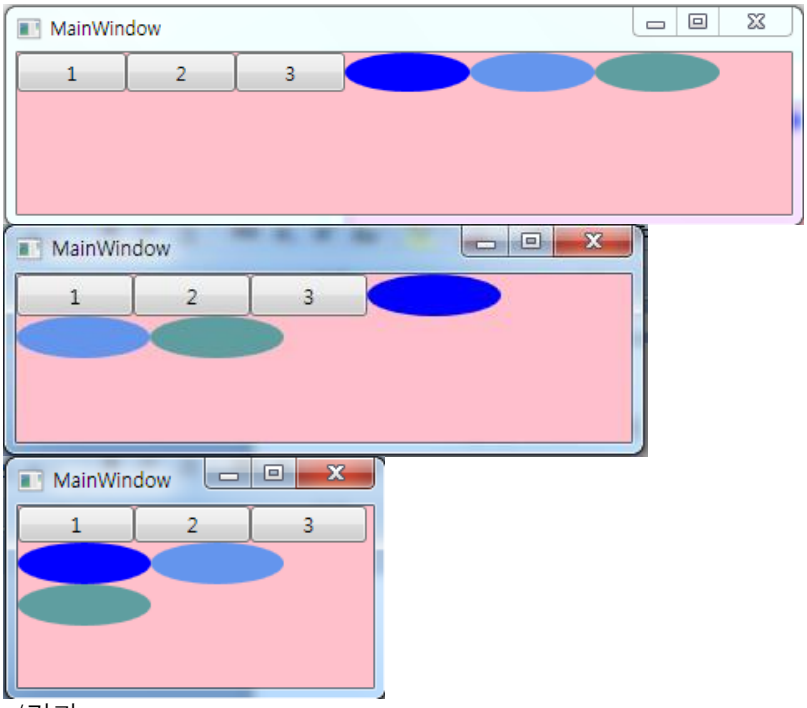
WrapPanel 의 기본 동작은 StackPanel 과 같고 다른 점은 자식 요소가 패널의 크기를 벗어나면 다음 줄에 배치한다.

다음은 Button 세 개와 Ellipse 세 개를 WrapPanel 에 배치한 예제다.

<코드>
[예제 02-09] WrapPanel의 자식 요소

```
<WrapPanel Background="Pink">
        <Button Width="70" Content="1"/>
        <Button Width="70" Content="2"/>
        <Button Width="70" Content="3"/>
        <Ellipse Width="80" Height="25" Fill="Blue"/>
        <Ellipse Width="80" Height="25" Fill="CornflowerBlue"/>
        <Ellipse Width="80" Height="25" Fill="CadetBlue"/>
</WrapPanel>
```

</코드>
<결과>
[출력 결과]



</결과>

첫 번째 결과는 StackPanel 과 같고 두 번째 결과는 윈도우의 크기를 줄이면 한 줄에 모두 자식 요소를 배치할 수 없을 때 다음 줄에 내려 배치한다. 세 번째 결과도 윈도우의 크기를 더 줄였을 때의 그림이다.

Orientation 속성을 이용하면 세로로 자식 요소들을 배치하고 정렬한다.

다음은 세로로 자식 요소를 배치한 WrapPanel 예제다.

<코드>

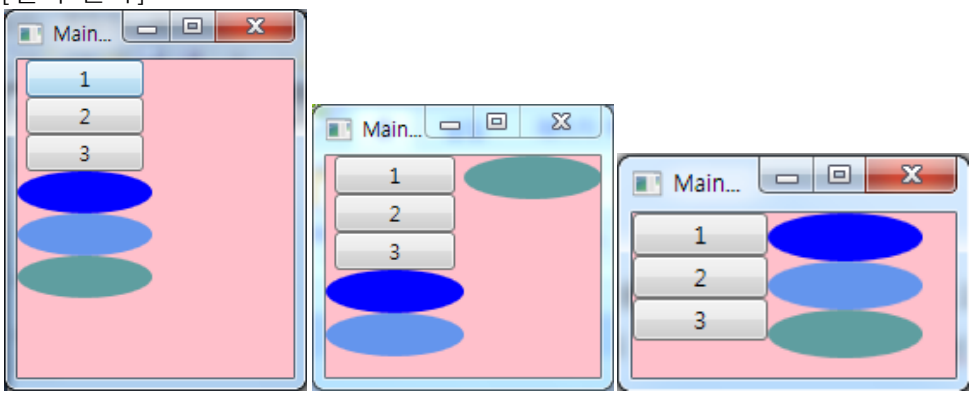
[예제 02-10] WrapPanel의 Orientation 속성

```
<WrapPanel Orientation="Vertical" Background="Pink">
        <Button Width="70" Content="1"/>
        <Button Width="70" Content="2"/>
        <Button Width="70" Content="3"/>
        <Ellipse Width="80" Height="25" Fill="Blue"/>
        <Ellipse Width="80" Height="25" Fill="CornflowerBlue"/>
        <Ellipse Width="80" Height="25" Fill="CadetBlue"/>
</WrapPanel>
```

</코드>

<결과>

[출력 결과]



</결과>

Orientation 속성을 사용하여 세로로 자식 요소들을 배치한 결과다. 윈도우의 크기를 줄이면 자식 요소들을 오른쪽으로 배치하게 된다.

<절제목>

04. DockPanel

</절제목>

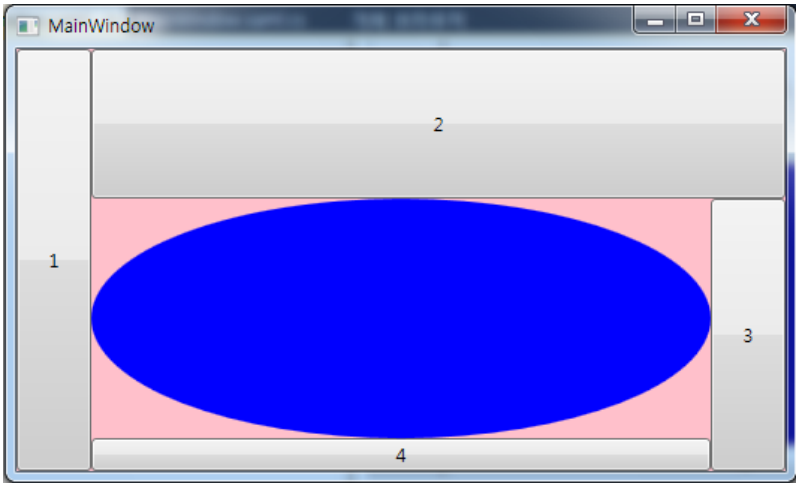
DockPanel 은 전체 레이아웃을 표현할 때 주로 사용하며 패널에서 지정한 방향으로 항목을 배치시킨다. DockPanel 은 자식 요소를 지정한 방향으로 배치시키고 마지막 자식 요소는 남은 공간을 채우도록 배치하는 것이 기본 동작이다. Canvas 와 같이 DockPanel 도 DockPanel.Dock 이란 결합 속성을 사용하며 이 결합 속성을 사용하여 자식 요소들이 지정한 방향에 배치된다.

다음은 DockPanel 에 자식 요소를 배치하는 예제다.

<코드>
[예제 02-11] DockPanel의 자식 요소

```
<DockPanel Background="Pink">
        <Button DockPanel.Dock="Left" Width="50" Content="1"/>
        <Button DockPanel.Dock="Top" Height="100" Content="2"/>
        <Button DockPanel.Dock="Right" Width="50" Content="3"/>
        <Button DockPanel.Dock="Bottom"   Content="4"/>
        <Ellipse Fill="Blue"/>
</DockPanel>
```

</코드>
<결과>
[출력 결과]



</결과>

1 번 자식 요소 Button 은 결합 속성 DockPanel.Dock=”Left” 을 사용하여 왼쪽에 가장 먼저 배치한다. 2 번 자식 요소 Button 은 Top 에 두 번째로 배치하며 3 번 Button 은 Right 에 4 번 Button 은 Bottom 에 각각 배치한다. 마지막으로 자식 요소 Ellipse 는 위치와 크기를 지정하지 않아도 남은 공간을 채운다. 왼쪽과 오른쪽에 배치되는 1 번과 3 번 Button 의 Height 은 기본 동작하도록 설정하지 않았으며 Height 은 기본 동작이 Button 의 컨텐츠에 맞춰지므로 Width 를

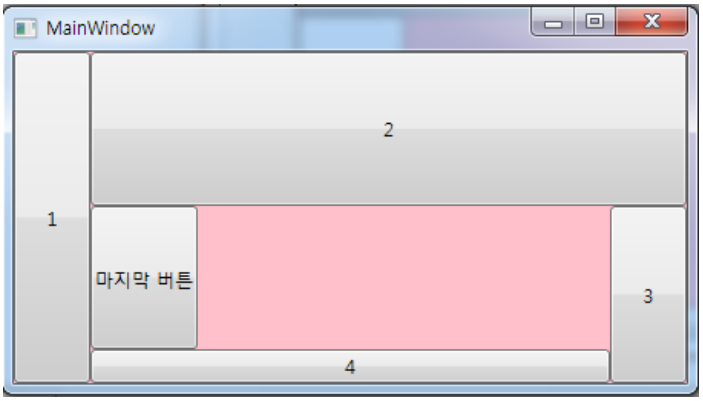
설정하였다. 2 번 Button 은 Width 는 기본 동작하도록 설정하지 않고 Height 은 100 을 설정했다. 4 번 Button 은 Height 을 설정하지 않았으므로 컨텐츠에 맞게 그려진다.

LastChildFill 속성을 False 로 설정하면 DockPanel 의 남은 공간을 채우지 않고 남기게 된다. 다음은 남은 공간을 채우지 않고 남기는 DocPanel 예제다.

<코드>
[예제 02-12] DockPanel의 LastChildFill 속성

```
<DockPanel LastChildFill="False" Background="Pink">
        <Button DockPanel.Dock="Left" Width="50" Content="1"/>
        <Button DockPanel.Dock="Top" Height="100" Content="2"/>
        <Button DockPanel.Dock="Right" Width="50" Content="3"/>
        <Button DockPanel.Dock="Bottom"   Content="4"/>
        <Button Content="마지막 버튼"/>
</DockPanel>
```

</코드>
<결과>
[출력 결과]



</결과>

LastChlidFill=”false”를 사용하여 남은 공간을 마지막 버튼이 채우지 않도록 했다. 이 Button 은 기본 동작이 왼쪽에 배치되어 Height 은 패널에 맞춰지고 Width 는 컨텐츠 크기를 갖는다. 만약 Ellipse 을 마지막 자식 요소로 선택했다면 Ellipse 는 크기(Width, Height)을 설정해야 화면에 보일 것이다.

<절제목>

05. Grid

</절제목>

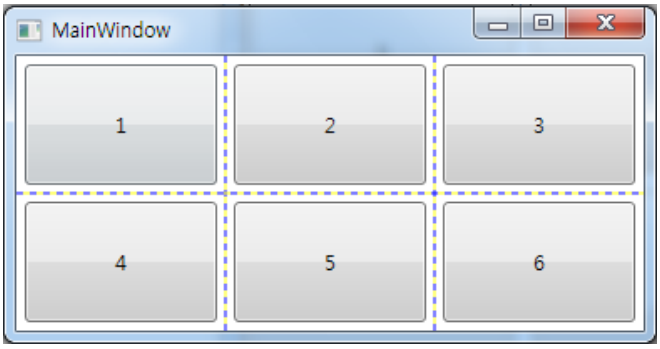
Grid 패널은 패널 중 가장 강력한 패널로 자식 요소를 표 형태로 배치하며 자식 요소들 간의 간격과 배율을 유지 시킬 수 있다. 행과 열로 이루어진 Grid 는 셀이라 부르는 행,열의 위치에 자식 요소를 배치하거나 여러 셀에 자식 요소를 배치할 수 있다.

다음은 2*3 형태의 Grid 패널로 Button 을 배치하는 예제다.

<코드>
[예제 02-13] Grid 패널

```
<Grid ShowGridLines="True">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>

        <Button Grid.Row="0" Grid.Column="0" Margin="5" Content="1" />
        <Button Grid.Row="0" Grid.Column="1" Margin="5" Content="2" />
        <Button Grid.Row="0" Grid.Column="2" Margin="5" Content="3" />
        <Button Grid.Row="2" Grid.Column="0" Margin="5" Content="4" />
        <Button Grid.Row="2" Grid.Column="1" Margin="5" Content="5" />
        <Button Grid.Row="2" Grid.Column="2" Margin="5" Content="6" />
</Grid>
```

</코드>
<결과>
[출력 결과]



</결과>

Grid.RowDefinitions 속성 요소는 RowDefinition 요소를 사용하여 행의 개수를 결정한다. Grid.ColumnDefinitions 속성 요소는 ColumnDefintion 요소를 사용하여 열의 개수를 결정한다.

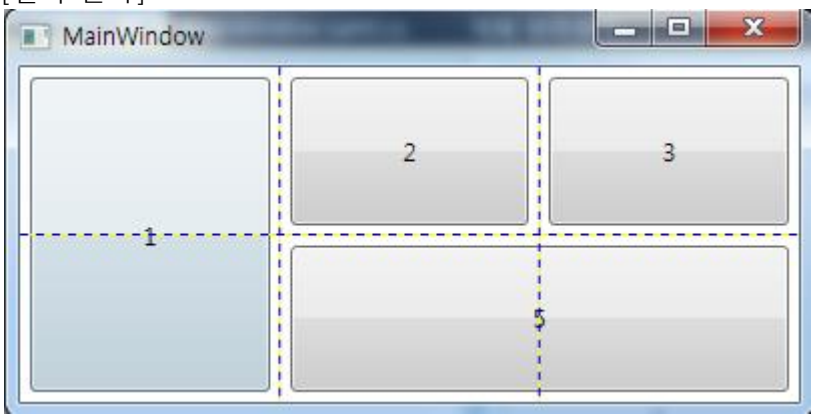
각 자식 요소는 Grid.Row, Grid.Column 결합 속성을 사용하여 배치할 위치를 지정한다. ShowGridLines=”true” 속성은 셀들의 경계를 확인하기 위해 점선을 출력한다.

Grid.RowSpan 과 Grid.ColumnSpan 을 사용하면 여러 셀에 하나의 자식 요소를 배치할 수 있다. 다음은 Grid.RowSpan 과 Grid.ColumnSpan 을 사용한 Grid 패널 예제다.

<코드>
[예제 02-14] 여러 셀을 하나로 합치기

```
<Grid ShowGridLines="True">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Button Grid.Row="0" Grid.Column="0" Grid.RowSpan="2" Margin="5" Content="1" />
        <Button Grid.Row="0" Grid.Column="1" Margin="5" Content="2" />
        <Button Grid.Row="0" Grid.Column="2" Margin="5" Content="3" />
        <Button Grid.Row="2" Grid.Column="1" Grid.ColumnSpan="2" Margin="5" Content="5" />
</Grid>
```

</코드>
<결과>
[출력 결과]



</결과>

1 번 Button 은 Grid.RowSpan 을 사용하여 두 행을 하나의 셀로 사용하고 있으며 5 번 Button 은 Grid.ColumnSpan 을 사용하여 두 열을 하나의 셀로 사용하고 있다.

Grid 패널은 자식 요소들의 크기를 다루기 위한 방법으로 세 가지(고정, 자동, 균등)의 옵션을 제공한다. 고정 크기는 Width 와 Height 을 사용하여 크기를 고정 시키고 자동은 자식 요소의 컨텐츠 크기에 맞게 자동으로 조정되며 균등은 각 요소들을 일정한 비율로 크기 조정한다.

다음 예제는 열의 크기를 Width 속성을 사용하여 고정 시킨 예제다.

<코드>
[예제 02-15] 열에 고정 크기 지정하기

```
<Grid ShowGridLines="True" Background="LightBlue">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100"/>
            <ColumnDefinition Width="150"/>
            <ColumnDefinition Width="200"/>
        </Grid.ColumnDefinitions>

         <Button Grid.Row="0" Grid.Column="0" Margin="5" Content="1번" />
        <Button Grid.Row="0" Grid.Column="1" Margin="5" Content="2번" />
        <Button Grid.Row="0" Grid.Column="2" Margin="5" Content="3번 컨텐츠가 긴 버튼" />
        <Button Grid.Row="2" Grid.Column="0" Margin="5" Content="4번 컨텐츠가 긴 버튼" />
        <Button Grid.Row="2" Grid.Column="1" Margin="5" Content="5번" />
        <Button Grid.Row="2" Grid.Column="2" Margin="5" Content="6번" />
    </Grid>
```

</코드>
<결과>
[출력 결과]

</결과>

열을 고정 크기(100, 150, 200)로 설정했으며 4 번 Button 은 고정 크기보다 컨텐츠가 길기 때문에 컨텐츠가 잘리게 된다. 윈도우가 커지면 자식 요소들이 고정 크기를 가지므로 Grid 패널은 잘리거나 남는 영역이 생긴다.

다음은 Grid 열을 자동(Auto) 크기로 설정한 예제다.

<코드>
[예제 02-16] 열에 자동 크기 지정하기

```
<Grid ShowGridLines="True" Background="LightBlue">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
```

```
        </Grid.ColumnDefinitions>

        <Button Grid.Row="0" Grid.Column="0" Margin="5" Content="1번 버튼" />
        <Button Grid.Row="0" Grid.Column="1" Margin="5" Content="2번" />
        <Button Grid.Row="0" Grid.Column="2" Margin="5" Content="3번 컨텐츠가 긴 버튼" />
        <Button Grid.Row="2" Grid.Column="0" Margin="5" Content="4번 컨텐츠가 긴 버튼" />
        <Button Grid.Row="2" Grid.Column="1" Margin="5" Content="5번" />
        <Button Grid.Row="2" Grid.Column="2" Margin="5" Content="6번" />
    </Grid>
```
</코드>
<결과>
[출력 결과]

</결과>

열의 옵션이 자동으로 설정되면 모든 컨텐츠의 내용이 보이게 되며 Grid 패널이 자식 요소들보다 커지면 빈 영역이 만들어 진다. 4 번 Button 을 포함하는 0 열은 4 번 Button 의 크기에 맞춰 열의 크기가 결정되고 3 번 Button 을 포함하는 2 열은 3 번 Button 의 크기에 맞춰 열의 크기가 결정된다. 2 번과 5 번 Button 은 컨텐츠의 크기가 같으므로 딱 그 크기 만큼으로 열의 크기가 결정된다.

다음은 Grid 패널의 열의 크기를 균등 크기를 사용한 예제다. 균등 조정은 Star(스타) 라고 부르는 *를 사용한다. 고정 크기 때와 비교해 보기 바란다.

<코드>
[예제 02-17] Grid 패널의 균등 조정
```
<Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*"/>
        <ColumnDefinition Width="1.5*"/>
        <ColumnDefinition Width="2*"/>
</Grid.ColumnDefinitions>
```
</코드>
<결과>
[출력 결과]

</결과>

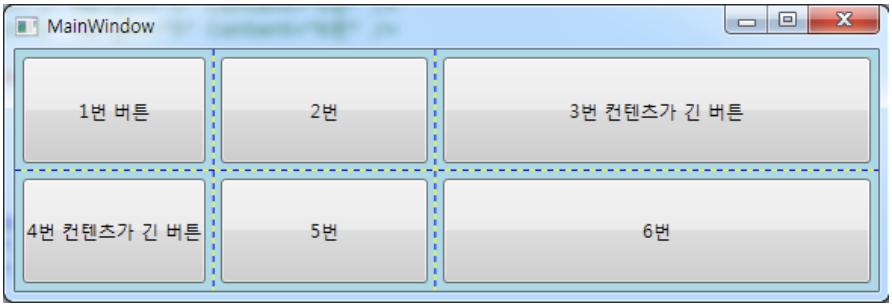
두 결과 모두 각각의 세 열이 항상 1:1.5:2 의 비율을 유지한다. 두 번째 결과는 4 번 컨텐츠가 잘리며 열들은 항상 비율을 유지한다. 또 Grid 패널은 남는 영역 없이 꽉 채워 그려진다. 크기를 변경하며 실행해 보기 바란다.

세 가지 크기 조정을 섞어 쓰는 것도 가능하며 다음 예제는 자동과 균등을 사용한 Grid 패널 예제다.

<코드>
[예제 02-18] Grid 패널의 자동과 균등 조정

```
<Grid ShowGridLines="True" Background="LightBlue">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="1*"/>
            <ColumnDefinition Width="2*"/>
        </Grid.ColumnDefinitions>
```

</코드>
<결과>
[출력 결과]



</결과>

0 열은 자동으로 4 번 Button 의 컨텐츠 크기에 맞춰 배치되며 1 열과 2 열은 1:2 비율로 크기가 맞춰진다.

GridSplitter 요소를 사용하면 행이나 열의 크기를 조절할 수 있다. GridSplitter 요소를 사용할 때 Width나 Height 속성을 사용해야 화면에 보인다.

다음은 GridSplitter 요소를 사용하여 동적으로 열의 크기를 변경하는 Grid 패널 예제다.

<코드>

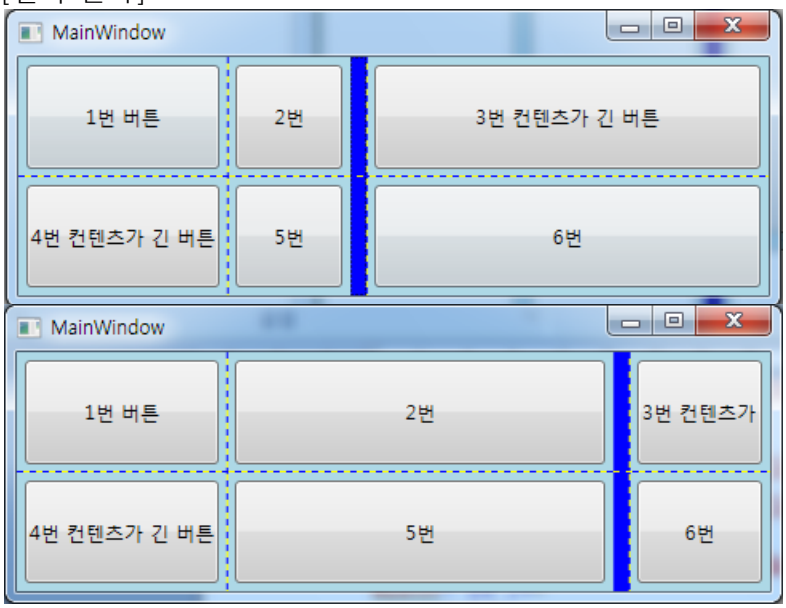
[예제 02-19] GridSplitter 요소

```
<Grid ShowGridLines="True" Background="LightBlue">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="1*"/>
            <ColumnDefinition Width="2*"/>
        </Grid.ColumnDefinitions>

        <Button Grid.Row="0" Grid.Column="0" Margin="5" Content="1번 버튼" />
        <Button Grid.Row="0" Grid.Column="1" Margin="5,5,15,5" Content="2번" />
        <Button Grid.Row="0" Grid.Column="2" Margin="5" Content="3번 컨텐츠가 긴 버튼" />
        <Button Grid.Row="2" Grid.Column="0" Margin="5" Content="4번 컨텐츠가 긴 버튼" />
        <Button Grid.Row="2" Grid.Column="1" Margin="5,5,15,5" Content="5번" />
        <Button Grid.Row="2" Grid.Column="2" Margin="5" Content="6번" />

        <GridSplitter Grid.RowSpan="2" Grid.Column="1" Background="Blue"    Width="10"/>
    </Grid>
```

</코드>

<결과>

[출력 결과]



</결과>

여기서 주의할 점은 GridSplitter 요소는 다른 자식 요소와 같은 행과 열에 놓이며 GridSplitter 요소가 움직일 때 행이나 열에 있는 요소들이 변경 가능하도록 설정되어야 스플라이터의 효과를 볼 수 있다. 위 예제는 GridSplitter 요소가 1 열에 위치하며 Grid.RowSpand=”2”로 하여 두 행 모두에 그려지도록 했으며 1 열과 2 열이 균등 조정으로 크기 변경이 가능하다. 또 2 번과 5 번 Button 의 Margin 의 Right 을 15 로 설정하여 GridSplitter 요소와 다른 Button 요소와의 간격을 일정하게 보이도록 설정하였다.

하나의 Grid 에서 열의 크기는 항상 일정하게 유지된다. 만약 두 개의 서로 다른 Grid 가 존재할 때 두 Grid 의 서로 독립적이므로 열의 키기가 서로 다르게 유지된다. 이때 서로 다른 Grid 의 열을 하나의 Grid 열처럼 다룰 수 있는데 ‘크기 공유 그룹’을 이용하면 가능하다.

다음은 서로 다른 두 Grid 가 열의 크기를 공유할 수 있도록 하는 예제다.

<코드>
[예제 02-01]

```
<Grid Grid.IsSharedSizeScope="True">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid Grid.Row="0" ShowGridLines="True" Background="Yellow">
            <Grid.RowDefinitions>
                <RowDefinition />
                <RowDefinition />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition SharedSizeGroup="Group1"/>
                <ColumnDefinition Width="200" SharedSizeGroup="Group2"/>
                <ColumnDefinition SharedSizeGroup="Group3"/>
            </Grid.ColumnDefinitions>

            <Button Grid.Row="0" Grid.Column="0" Margin="5" Content="1번 버튼" />
            <Button Grid.Row="0" Grid.Column="1" Margin="5" Content="2번" />
            <Button Grid.Row="0" Grid.Column="2" Margin="5" Content="3번 컨텐츠가 긴 버튼" />
            <Button Grid.Row="2" Grid.Column="0" Margin="5" Content="4번" />
            <Button Grid.Row="2" Grid.Column="1" Margin="5" Content="5번" />
            <Button Grid.Row="2" Grid.Column="2" Margin="5" Content="6번" />
        </Grid>
        <Grid Grid.Row="1" ShowGridLines="True" Background="LightBlue">
            <Grid.RowDefinitions>
                <RowDefinition />
                <RowDefinition />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition SharedSizeGroup="Group1"/>
                <ColumnDefinition SharedSizeGroup="Group2"/>
                <ColumnDefinition SharedSizeGroup="Group3"/>
            </Grid.ColumnDefinitions>
```

```
            <Button Grid.Row="0" Grid.Column="0" Margin="5" Content="7번 버튼" />
            <Button Grid.Row="0" Grid.Column="1" Margin="5" Content="8번" />
            <Button Grid.Row="0" Grid.Column="2" Margin="5" Content="9번" />
            <Button Grid.Row="2" Grid.Column="0" Margin="5" Content="10번 컨텐츠가 긴 버튼" />
            <Button Grid.Row="2" Grid.Column="1" Margin="5" Content="11번" />
            <Button Grid.Row="2" Grid.Column="2" Margin="5" Content="12번" />
        </Grid>
</Grid>
```
</코드>
<결과>
[출력 결과]

</결과>

색상이 다른 부분이 서로 다른 두 Grid 패널이다. 하지만 두 Grid 패널의 행과 열이(각각 2 행 3 열) 하나의 Grid 패널의 행과 열처럼(4 행 3 열) 동작하는 것을 볼 수 있다. SharedSizeGroup 속성을 열에 사용했으며 같은 이름을 갖는 그룹은 하나의 열처럼 동작한다. 또, 두 Grid 패널을 포함하는 부모 Grid 패널은 Grid.IsSharedSizeScope 가 true 로 설정되어 있어야 이 요소 내의 Grid 패널이 '크기 공유 그룹'을 설정한다.

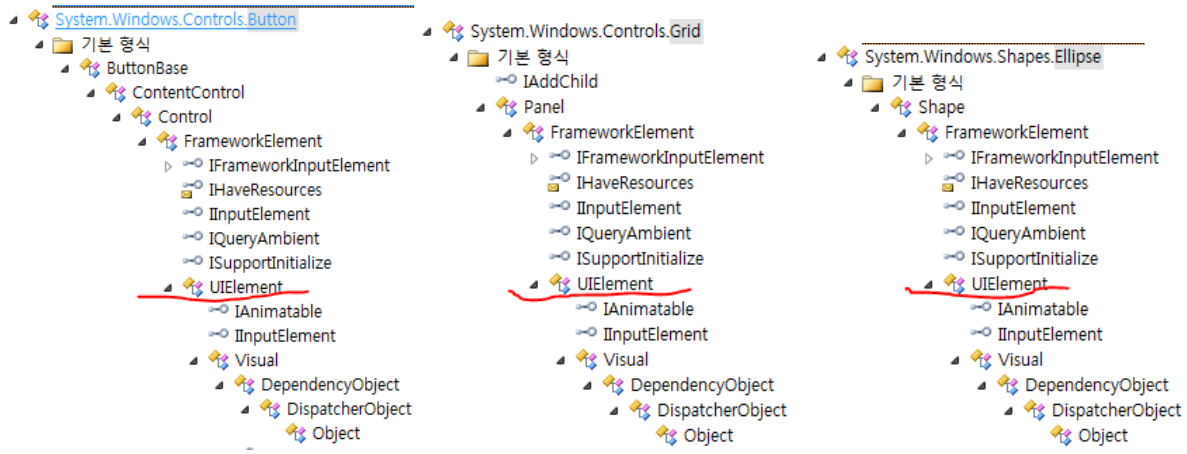
<장제목>

3. 사용자 인터페이스

</장제목>

WPF 의 사용자 인터페이스 요소는 사용자 입력을 받을 수 있도록 설계되었다. 정확히 말하면 모든 사용자 인터페이스 요소는 UIElement 클래스를 기반으로 만들어진다. UIElement 클래스는 키보드 마우스 등과 같은 사용자 인테페이스 기능과 뒤에 설명할 WPF 특징적인 이벤트 메커니즘인 '라우트된 이벤트' 시스템을 제공한다. 컨트롤뿐만 아니라 패널이나 Shape 요소까지도 UIElement 클래스를 기반으로 하는 요소이므로 사용자 인터페이스 요소다.

다음은 Button 과 Grid, Ellipse 요소의 클래스 상속 구조를 보인다.



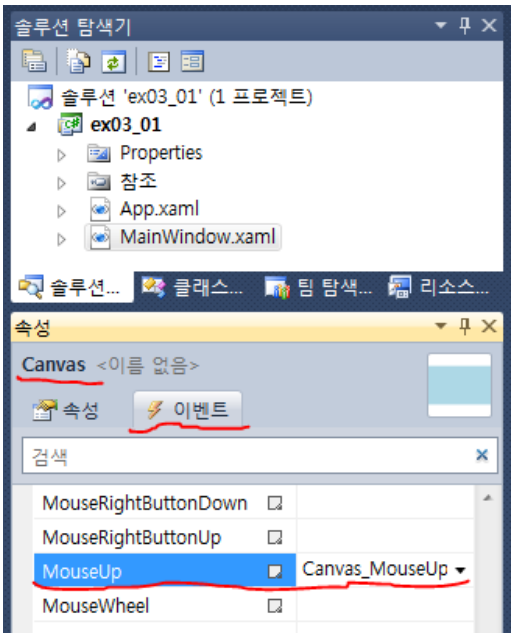
<절제목>

01. 마우스 이벤트

</절제목>

마우스 이벤트는 UIElement 라는 기본 클래스로부터 파생하며 이 클래스는 MouseDown, MouseUp, MouseMove 등의 여러 이벤트를 정의하고 있다. 이런 이벤트는 MouseEventArgs 대리자를 통해 마우스와 관련한 여러 정보를 확인할 수 있다. MouseEventArgs 는 입력관 관련된 기본 정보 클래스인 InputEventArgs 로부터 상속한다.

다음은 Canvas 에 MouseUp 이벤트를 VisualSudio 에서 생성하는 그림이다.

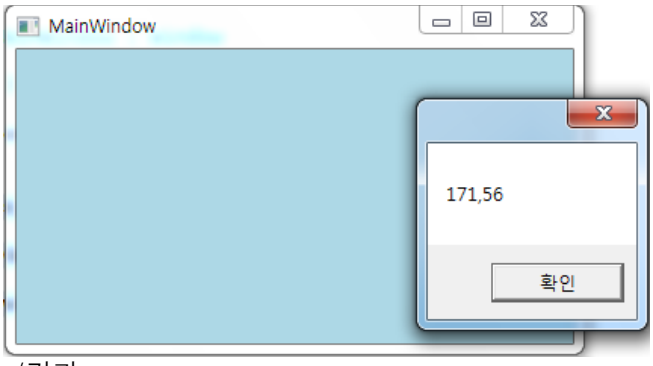


다음은 마우스 입력 테스트를 위한 간단한 Canvas 패널 예제다.

<코드>
[예제 03-01] 마우스 Up 이벤트

```
<Canvas Background="LightBlue" MouseUp="Canvas_MouseUp">

</Canvas>

private void Canvas_MouseUp(object sender, MouseButtonEventArgs e)
{
        Point pos = e.GetPosition(this);

        MessageBox.Show(pos.ToString());
```

}
</코드>
<결과>
[출력 결과]



</결과>

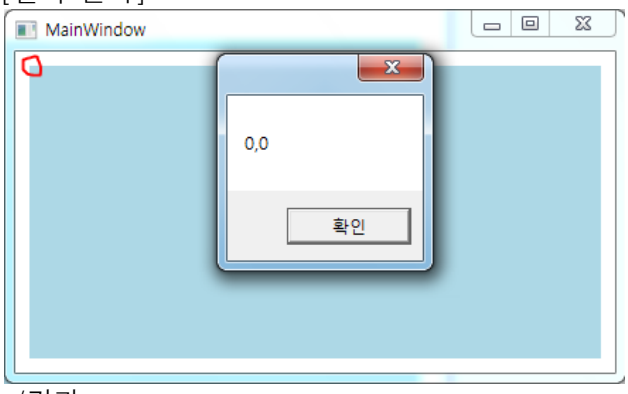
MouseUp 이벤트를 처리하기 위해 Canvas_MouseUp 핸들러를 등록하고 마우스 위치를 메시지 박스에 출력한다. 마우스 위치의 확인을 위해 MouseButtonEventArgs 의 e.GetPosition()을 사용하였다. 여기서 인수로 전달한 this 는 윈도우의 마우스 위치이며 Canvas 패널의 위치를 확인하기 위해서는 Canvas 의 참조를 인수로 전달해야 한다.

다음은 Canvas 를 e.GetPosition()의 인수로 전달했을 때의 마우스의 위치를 출력하는 예제다.

<코드>
[예제 03-02]

```
<Canvas Name="canvas" Margin="10" Background="LightBlue" MouseUp="Canvas_MouseUp">
</Canvas>

private void Canvas_MouseUp(object sender, MouseButtonEventArgs e)
{
                Point pos = e.GetPosition(canvas);

                MessageBox.Show(pos.ToString());
}
```

</코드>
<결과>
[출력 결과]



</결과>

하늘색이 Canvas 영역으로 Margin 을 설정하여 10 만큼의 여백을 주고 윈도우와 구분할 수 있도록 했다. Canvas 의 왼쪽, 위가 0,0 이 되고 윈도우 위치로는 10,10 이 된다. Margin 값을 지우고 출력해 보기 바란다.

IsMouseOver 와 IsMouseDirectlyOver 속성을 사용하여 현재 마우스가 요소 위에 위치하는지 확인할 수 있다.
다음은 윈도우와 Canvas 그리고 Ellipse 위에 마우스가 위치하는 지를 판단 예제다.

<코드>
[예제 03-03] 마우스 위치 확인을 위한 XAML 코드와 C# 코드

```
<Window x:Class="ex03_03.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"   MouseMove="MainWindow_MouseMove">
    <Canvas Name="canvas" Margin="30" Background="LightBlue">
        <Ellipse Canvas.Left="50" Canvas.Top="50" Width="100" Height="100" Fill="Yellow" />
    </Canvas>
</Window>
```

```
private void MainWindow_MouseMove(object sender, MouseEventArgs e)
{
    if (canvas.IsMouseOver)
    {
        Title = "Canvas 위에 마우스 있음";

        if (canvas.IsMouseDirectlyOver == false)
            Title += "+ Canvas 자식 요소에 마우스 있음 ";
    }
    else
        Title = "Canvas 위에 마우스 없음";
}
```

</코드>
<결과>
[출력 결과]

</결과>

우선 마우스 이동 이벤트를 처리하기 위해 윈도우에 MouseMove 이벤트를 등록했다. Canvas 패널의 참조 canvas.IsMouseOver 를 사용하여 하늘색 바탕의 Canvas 에 마우스가 위치하는 지 판단하고 윈도우의 Title 바에 문자열을 출력한다. 또 Canvas 를 기준으로 자식 요소인 Ellipse 요소 위의 마우스와 Canvas 자체 위의 마우스를 탐지하기 위해 canvas.IsMouseDirectlyOver 를 사용했다. IsMouseOver 속성은 자식 요소까지 포함하여 Canvas 위에 마우스가 존재한다면 true 이고 IsMouseDirectlyOver 속성은 자식 요소가 아닌 자신(Canvas) 위에만 마우스가 존재할 때 true 값이다.

여기서 특징적인 사항은 Ellipse 요소나 Canvas 요소의 MouseMove 이벤트를 윈도우(Window) 객체에서 처리하는 것을 볼 수 있는데 WPF 의 '라우트된 이벤트' 라는 새로운 이벤트 메커니즘으로 가능하며 이 개념으로 WPF 의 컨트롤들이 무한개?!의 컨텐츠를 가지면서도 컨트롤 이벤트를 효율적으로 처리할 수 있게 된다. '라우트된 이벤트' 개념은 뒤에서 공부한다.

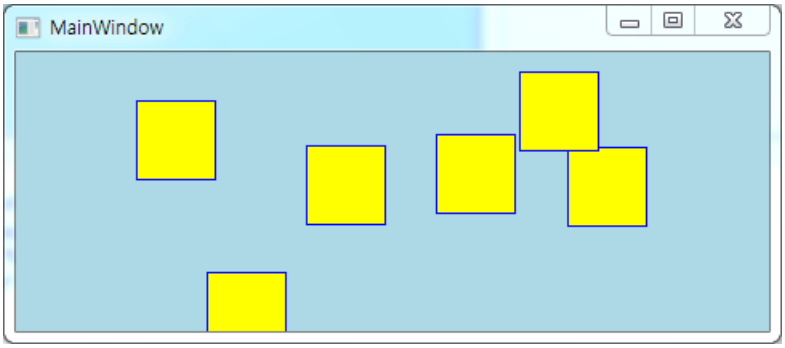
다음은 MouseLeftButtonUp 이벤트가 발생할 때 Canvas 패널에 사각형을 동적으로 배치하는(그리는) 예제다.

<코드>
[예제 03-04] Canavas에 사각형 그리기 XAML 코드와 C# 코드

```
<Canvas Name="canvas" Background="LightBlue" MouseLeftButtonUp="canvas_MouseLeftButtonUp">
</Canvas>
```

```
private void canvas_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
    Rectangle rt = new Rectangle();
    rt.Width = 50;
    rt.Height = 50;
    rt.Stroke = new SolidColorBrush(Colors.Blue);
    rt.Fill = new SolidColorBrush(Colors.Yellow);

    //사각형을 Canvas 어디에 배치할 것인지 지정한다.
    Canvas.SetLeft(rt, e.GetPosition(canvas).X);
    Canvas.SetTop(rt, e.GetPosition(canvas).Y);

    // 사각형을 위치시킨다.(그린다.)
    canvas.Children.Add(rt);
}
```

</코드>
<결과>
[출력 결과]



</결과>

XAML 코드에서 Ellipse 요소를 배치(그린)한 것처럼 C#코드에서 Rectangle 요소를 Canvas 에 배치하기 위해 Canvas.Left XAML 특성 대신 Canvas.SetLeft()를 Canvas.Top XAML 특성 대신 Canvas.SetTop() 메소드를 사용했으며 XAML 에서 Ellipse 요소를 자식 요소로 갖는 것처럼 Canvas.Children 컬렉션에 Rectangle 요소를 추가했다.

<절제목>

02. 키보드 이벤트

</절제목>

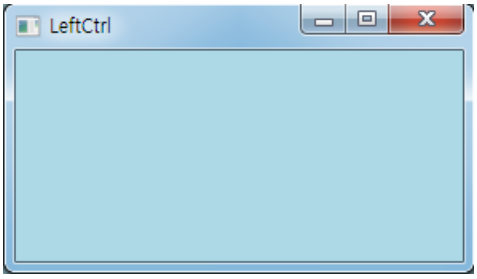
WPF 는 활성화된 요소의 키보드 동작을 처리하기 위해 키보드 관련 기본 기능을 UIElement 클래스를 통해 제공하다. KeyDown, KeyUp 이벤트 등이 정의되어 있으며 이 이벤트들의 핸들러로 KeyEventArgs 형식 정보가 전달된다.

다음은 윈도우가 키보드 입력을 받아 타이틀바에 출력 문자를 출력하는 예제다.

<코드>
[예제 03-05] 윈도우 타이틀바에 키보드 문자 출력

```
<Window x:Class="ex_3_05.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" KeyDown="Window_KeyDown">
    <Canvas Name="canvas" Background="LightBlue">
    </Canvas>
</Window>

private void Window_KeyDown(object sender, KeyEventArgs e)
{
    Title = e.Key.ToString();
}
```

</코드>
<결과>
[출력 결과]



</결과>

사용자 키 입력은 포커스를 가지는 단 하나의 요소만이 이벤트를 받는다. 이 예제는 프로그램이 시작되면 윈도우가 활성화되며 윈도우가 포커스를 가진 요소가 된다. 물론 키 이벤트도 '라우트된 이벤트' 이므로 일반 .Net 이벤트와 조금 다르게 동작한다.

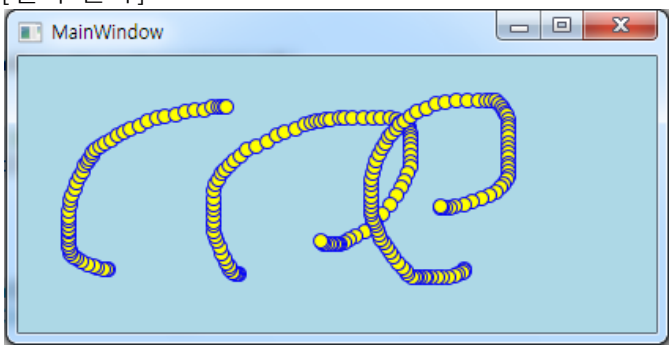
키 이벤트 외에도 Keyboard 클래스를 이용하면 키가 눌리는 것을 확인할 수 있다. Modifiers 속성이나 IsKeyDown() 메소드를 이용한다.

다음은 Keyboard 클래스의 메소드를 이용하여 Shift 키가 눌렸을 때 Ellipse 를 Canvas 에 배치(그리는)하는 예제다.

<코드>
[예제 03-06] Keyboard 클래스 사용한 XAML 코드와 C# 코드

```
<Canvas Name="canvas" Background="LightBlue" MouseMove="canvas_MouseMove">
</Canvas>

private void canvas_MouseMove(object sender, MouseEventArgs e)
{
    // 다음을 사용 가능하다
    //Keyboard.Modifiers & ModifierKeys.Shift) != 0
    if( Keyboard.IsKeyDown(Key.LeftShift))
    {
        Ellipse ell = new Ellipse();
        ell.Width = 10;
        ell.Height = 10;
        ell.Stroke = new SolidColorBrush(Colors.Blue);
        ell.Fill = new SolidColorBrush(Colors.Yellow);

        //Ellipse를 Canvas 어디에 배치할 것인지 지정한다.
        Canvas.SetLeft(ell, e.GetPosition(canvas).X);
        Canvas.SetTop(ell, e.GetPosition(canvas).Y);

        // Ellipse를 위치시킨다.(그린다.)
        canvas.Children.Add(ell);
    }
}
```

</코드>
<결과>
[출력 결과]



</결과>

Keyboard.IsKeyDown() 메소드 대신 Keyboard.Modifiers 를 사용해도 결과는 같다. MouseMove 이벤트에서 쉬프트가 눌렸을 때만 그리기를 수행한다.

<절제목>

03. 라우트된 이벤트

</절제목>

WPF 는 '라우트된 이벤트' 라는 새로운 개념의 이벤트 메커니즘을 제공한다. 의존 속성처럼 이벤트도 WPF 만의 기능(컨텐츠 모델)에 맞게 새로 정의된 이벤트로 크게 두 가지 버블링(bubbling) 이벤트와 터널링(tunneling) 이벤트가 있다. 터널링 이벤트는 말 그대로 부모 요소부터 자식 요소로 이벤트가 라우팅(전달)되며 연속하게 발생하고 버블링 이벤트는 마우스를 실제 위치한 자식 요소부터 부모 요소로 라우팅(전달)되며 발생한다.

WPF 의 대부분의 컨트롤은 컨텐츠 모델에 의해 다른 여러 요소들을 포함하므로 '라우트된 이벤트' 개념 없이 일반 .Net 이벤트를 사용한다면 컨트롤의 이벤트를 처리하기 위해 포함하는 컨텐츠의 모든 이벤트 처리 코드를 작성해야 한다. WPF 의 새로운 '라우트된 이벤트' 모델을 사용하면 관심있는 이벤트 하나만을 등록하고 사용할 수 있게 된다.

우리가 지금까지 사용한 이벤트는 사실 버블링 이벤트였으며 WPF 는 라우트된 이벤트를 .Net 의 일반 이벤트와 차이점 없이 사용할 수 있도록 문법을 제공한다. 예로 MouseDown 은 버블링 이벤트이며 상대적인 터널링 이벤트는 이벤트 접두어로 Preview 가 붙어 PreviewMouseDown 이다.

다음은 라우트된 이벤트(터널링 이벤트와 버블링 이벤트)를 확인해 보는 예제다.

<코드>
[예제 03-07] 터널링 이벤트와 버블링 이벤트(XAML 코드 + C# 코드)

```
<Window x:Class="ex03_07.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" PreviewMouseDown="Window_PreviewMouseDown"
MouseDown="Window_MouseDown">
    <Canvas Background="LightBlue" PreviewMouseDown="Canvas_PreviewMouseDown"
MouseDown="Canvas_MouseDown">
        <Label Canvas.Left="50" Canvas.Top="50" Padding="10" Background="Yellow"
PreviewMouseDown="Label_Yellow_PreviewMouseDown" MouseDown="Label_Yellow_MouseDown" >
            <StackPanel PreviewMouseDown="StackPanel_PreviewMouseDown"
MouseDown="StackPanel_MouseDown">
                <Ellipse Width="100" Height="50" Fill="Red"
PreviewMouseDown="Ellipse_Red_PreviewMouseDown" MouseDown="Ellipse_MouseDown" />
                <Label Background="LightGreen" HorizontalAlignment="Center" Content="버튼"
PreviewMouseDown="Label_Green_PreviewMouseDown" MouseDown="Label_Green_MouseDown" />
                <Ellipse Width="100" Height="50" Fill="Blue" />
            </StackPanel>
        </Label>
    </Canvas>
```

```
</Window>

public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
    public string Msg { get; set; }

    // 터널링 이벤트
    private void Window_PreviewMouseDown(object sender, MouseButtonEventArgs e)
    {
        //최초의 라우트된 이벤트 메시지
        Msg = "Window_터널링->";
    }
    private void Canvas_PreviewMouseDown(object sender, MouseButtonEventArgs e)
    {
        Msg += "Canvas_터널링->";
    }
    private void Label_Yellow_PreviewMouseDown(object sender, MouseButtonEventArgs e)
    {
        Msg += "Label_Yellow터널링->";
    }
    private void StackPanel_PreviewMouseDown(object sender, MouseButtonEventArgs e)
    {
        Msg += "StackPanel_터널링->";
    }
    private void Ellipse_Red_PreviewMouseDown(object sender, MouseButtonEventArgs e)
    {
        Msg += "Ellipse_터널링->";
    }
    private void Label_Green_PreviewMouseDown(object sender, MouseButtonEventArgs e)
    {
        Msg += "Label_Green터널링->";
    }

    // 버블링 이벤트
    private void Window_MouseDown(object sender, MouseButtonEventArgs e)
    {
        Msg += "Window_버블링";
        Title = Msg; // 지금까지 발생한 라우트된 이벤트를 출력한다.
    }
    private void Canvas_MouseDown(object sender, MouseButtonEventArgs e)
    {
        Msg += "Canvas_버블링->";
    }
    private void Label_Yellow_MouseDown(object sender, MouseButtonEventArgs e)
    {
        Msg += "Label_Yellow버블링->";
    }
    private void StackPanel_MouseDown(object sender, MouseButtonEventArgs e)
    {
```

```
            Msg += "StackPanel_버블링->";
        }
        private void Ellipse_MouseDown(object sender, MouseButtonEventArgs e)
        {
            Msg += "Ellipse_버블링->";
        }
        private void Label_Green_MouseDown(object sender, MouseButtonEventArgs e)
        {
            Msg += "Label_Green버블링->";
        }
}
```

</코드>
<결과>
[출력 결과]

</결과>

빨강 Ellipse 요소를 클릭했을 때 나타나는 결과로 최상위 부모 요소인 Window 요소의 터널링 이벤트부터 자식 요소로 한 단계씩 전달되며 다시 자식 요소부터 최상위 부모 요소인 Window 요소로 버블링 이벤트가 발생한다. '라우트된 이벤트' 개념이 존재하지 않다면 위의 노랑색 Label 이 클릭되었을 때 어떤 동작을 수행하고 싶다면 그 내부의 지식 컨텐츠인 StackPanel 이나 Ellipse 요소나 버튼 Label 요소에 모두 이벤트 핸들러를 등록하고 처리해야 한다. 라우트된 이벤트는 내부의 어떤 자식 요소가 클릭되어도 노랑색 Label 에 이벤트가 발생하므로 하나의 이벤트를 등록하고 처리할 수 있다.

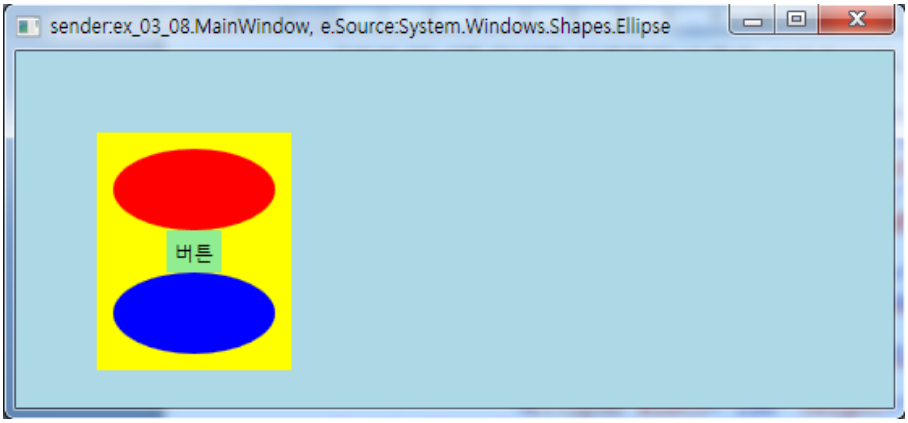
'라우트된 이벤트' 를 사용하면서 주의할 점은 핸들러에 전달되는 sender 는 이벤트를 등록한 객체의 참조이며 이벤트를 발생시킨 객체의 참조는 e.Source 로 얻을 수 있다.
다음은 핸들러로 전달된 이벤트 주체를 출력하는 예제다.

<코드>
[예제 03-08] 라우트된 이벤트의 sender와 e.Source 참조(XAML 코드와 C# 코드)

```
<Window x:Class="ex_03_08.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" MouseDown="Window_MouseDown">
    <Canvas Background="LightBlue" >
        <Label Canvas.Left="50" Canvas.Top="50" Padding="10" Background="Yellow" >
            <StackPanel>
                <Ellipse Width="100" Height="50" Fill="Red"/>
```

```
                    <Label Background="LightGreen" HorizontalAlignment="Center" Content="버튼"/>
                    <Ellipse Width="100" Height="50" Fill="Blue" />
                </StackPanel>
            </Label>
        </Canvas>
</Window>

private void Window_MouseDown(object sender, MouseButtonEventArgs e)
{
        string msg = "sender:"+sender.GetType().ToString()+", e.Source:" + e.Source.GetType().ToString();
                Title = msg;
}
```

</코드>
<결과>
[출력 결과]



</결과>

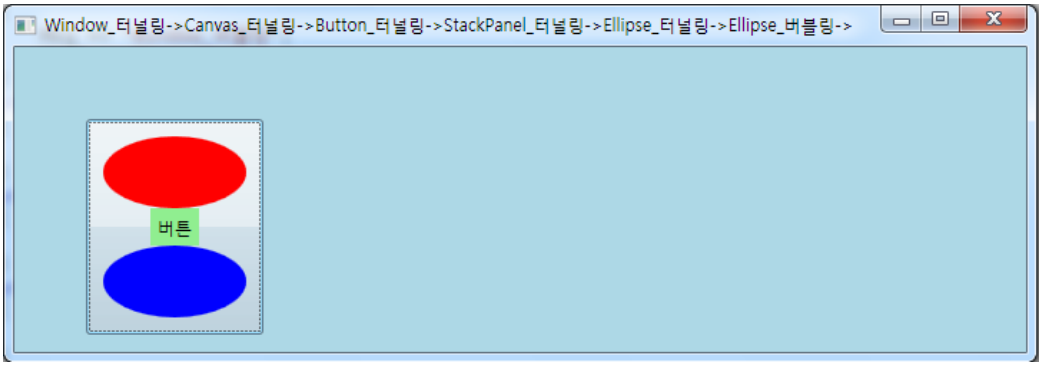
만약 노랑색의 Label 요소를 Button 으로 바꾸면 이벤트가 Button 요소부터 버블링되지 않는다. 그 이유는 Button 요소는 MouseDown 과 MouseUp 이벤트를 추상화하여 더 상위 개념의 Click 이벤트를 제공하며 Button 내부적으로 버블링 이벤트의 라우팅을 중단한다. Button 요소처럼 터널링이나 버블링 이벤트의 라우팅을 중단할 수 있는데 이벤트 핸들러의 e.Handled 속성을 true 설정하면 더 이상 라우트되지 않는다.

다음은 노랑색 Label 요소를 Button 요소로 바꾸고 이벤트의 라우팅을 확인한 예제다.

<코드>
[예제 03-09] 라우팅 이벤트 확인

```
<Canvas Background="LightBlue" PreviewMouseDown="Canvas_PreviewMouseDown"
MouseDown="Canvas_MouseDown">
        <Button Canvas.Left="50" Canvas.Top="50" Padding="10"
PreviewMouseDown="Button_PreviewMouseDown" MouseDown="Button_MouseDown" >
            <StackPanel PreviewMouseDown="StackPanel_PreviewMouseDown"
MouseDown="StackPanel_MouseDown">
                <Ellipse Width="100" Height="50" Fill="Red"
PreviewMouseDown="Ellipse_Red_PreviewMouseDown" MouseDown="Ellipse_MouseDown" />
                <Label Background="LightGreen" HorizontalAlignment="Center" Content="버튼"
PreviewMouseDown="Label_Green_PreviewMouseDown" MouseDown="Label_Green_MouseDown" />
```

```
                <Ellipse Width="100" Height="50" Fill="Blue" />
            </StackPanel>
        </Button>
</Canvas>
```
</코드>
<결과>
[출력 결과]



</결과>

빨강색 Ellipse 요소를 클릭했을 때 Preview 로 시작하는 터널링 이벤트는 모두 발생하지만 Button 요소의 버블링 이벤트부터는 발생되지 않는다. 다른 터널링 이벤트에서도 e.Handled 속성을 true 로 바꿔 이벤트 라우팅이 중단되는지 확인해 보기 바란다.

<장제목>

4. 컨트롤

</장제목>
WPF 에서 컨트롤은 사용자와 상용 작용하는 모든 사용자 인터페이스 요소를 말한다. WPF 의 컨트롤은 사용자와 인터페이스 한다는 의미에서 기존 컨트롤과 유사하지만 기존 컨트롤과 개념적으로 약간 다른 확장된 개념으로 정의된다.

컨트롤은 기본적인 형태를 갖추고 있지만 기존 컨트롤보다 쉽게 컨트롤의 외관을 변경할 수 있는 다양한 방법을 제공한다.

앞 장에서 공부한 컨텐츠 개념을 사용하여 컨텐츠 컨트롤이라면 모든 요소의 컨텐츠를 이용하여 변경이 가능하고 컨트롤 템플릿을 사용하여 전혀 다른 외관의 컨트롤로 변형이 가능하다. 사용자 정의 컨트롤을 사용하여 사용자가 직접 컨트롤을 작성할 수 있는 프레임 워크를 제공한다.

<절제목>

01. 기본 컨트롤

</절제목>

가장 기본적인 컨트롤은 버튼이다. 사용자의 클릭 입력을 애플리케이션에 전달하고 각각의 버튼에 따라 반응하도록 제공한다.

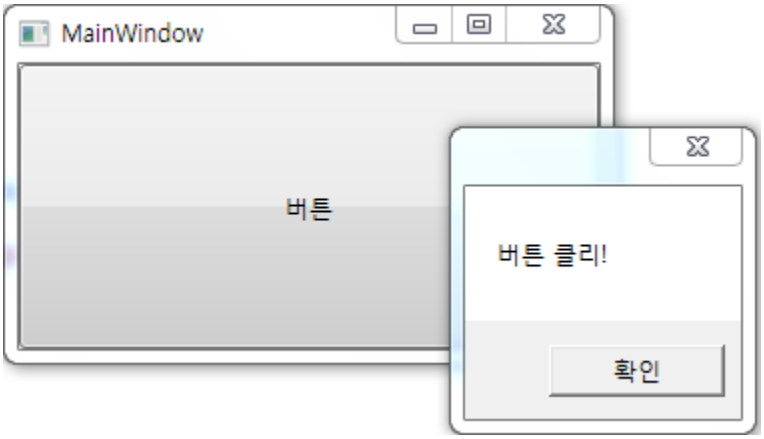
다음은 기본 적이 PUSH 버튼을 화면에 출력하고 클릭 이벤트 메시지를 처리하는 예제다.

<코드>
[예제 04-01] 기본 버튼

```
<Window x:Class="ex04_01.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>
        <Button Click="Button_Click">버튼</Button>
    </Grid>
</Window>
```

```
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show(" 버튼 클리! ");
        }
    }
```

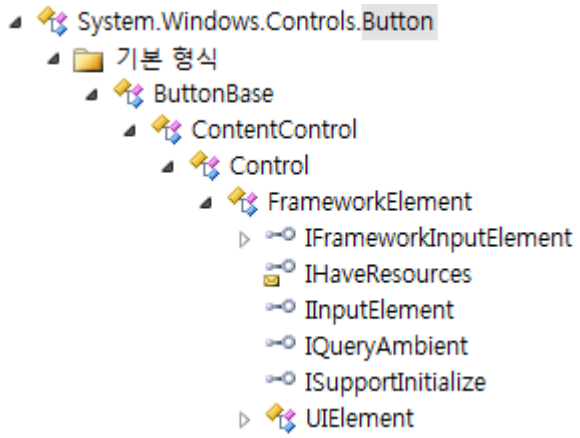
</코드>
<결과>
[출력 결과]



</결과>

결과에서 볼 수 있듯 버튼의 컨텐츠가 버튼의 캡션으로 사용된다. XAML 코드에서 Click 이벤트를 처리하기 위한 Button_Click 핸들러를 등록하고 C# 코드에서 메시지 박스를 뛰우는 간단한 코드를 작성한다.

버튼은 ButtonBase 를 기반 클래스로 파생되며 ButtonBase 클래스는 ContentControl 에서 파생되므로 모두 컨텐츠 모델을 지원한다.



다음은 Alt 키와 조합하여 엑세스 키를 만드는 예제다.

<코드>

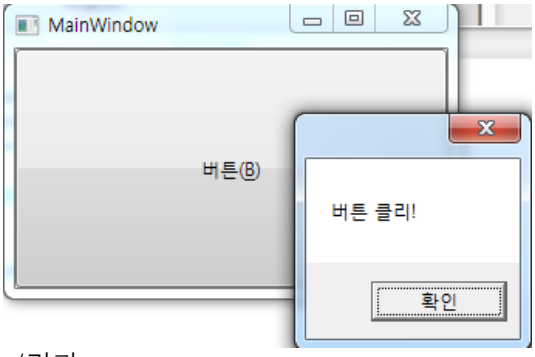
[예제 04-02] Alt-B 조합 만들기

```
<Grid>
        <Button Click="Button_Click">버튼(_B)</Button>
</Grid>
```

</코드>

<결과>

[출력 결과]



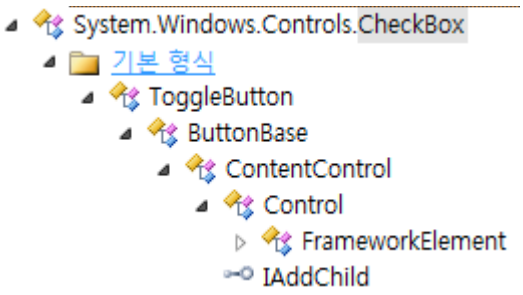
</결과>

버튼의 컨텐츠로 언더스코어를 사용하면(_B) 엑세스 키(기존 GUI 의 &문자와 같다)를 만들 수 있다.

WPF 는 AccessText 요소를 사용하여 키보드 액세스를 지원한다. 사실 위 예제는 AccessText 요소를 생략하고 작성한 코드로 다음과 같다.

```
<Grid>
        <Button Click="Button_Click">
            <AccessText>버튼(_B)</AccessText>
        </Button>
</Grid>
```

라디오 버튼과 체크 버튼은 ToggleButton 클래스에서 파생되며 IsChecked 속성을 통해 버튼이 체그되었는지를 확인할 수 있다. 이 속은 bool? 타입이다.



다음은 체크 버튼의 이벤트를 처리하는 예제다.

<코드>

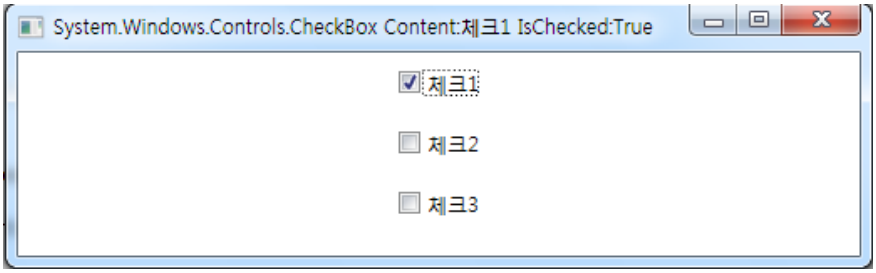
[예제 04-03] 체크 버튼

```
<Window x:Class="ex04_03.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow">
    <StackPanel>
        <CheckBox Content="체크1" Margin="10" HorizontalAlignment="Center" Checked="CheckBox_Checked"
/>
```

```
        <CheckBox Content="체크2" Margin="10" HorizontalAlignment="Center" Checked="CheckBox_Checked"
/>
        <CheckBox Content="체크3" Margin="10" HorizontalAlignment="Center" Checked="CheckBox_Checked"
/>
    </StackPanel>
</Window>
```

```
        private void CheckBox_Checked(object sender, RoutedEventArgs e)
        {
            Title = e.Source.ToString();
        }
```

</코드>
<결과>
[출력 결과]



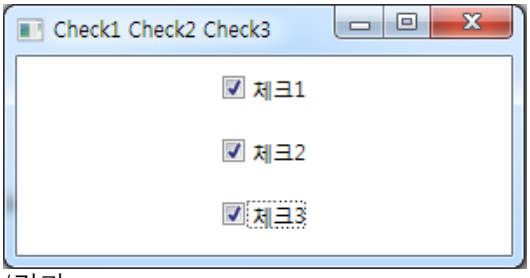
</결과>
체크 버튼에 Clicked 이벤트가 발생하면 타이틀 바에 체크 버튼 요소의 내용을 출력한다.

다음은 각각의 체크 버튼을 처리하기 위한 예제로 하나의 Checked 이벤트 핸들러를 사용한다.
<코드>
[예제 04-04] 각각의 체크 버튼 처리

```
<StackPanel>
        <CheckBox Name="check1" Content="체크1" Margin="10" HorizontalAlignment="Center"
Checked="CheckBox_Checked" />
        <CheckBox Name="check2" Content="체크2" Margin="10" HorizontalAlignment="Center"
Checked="CheckBox_Checked" />
        <CheckBox Name="check3" Content="체크3" Margin="10" HorizontalAlignment="Center"
Checked="CheckBox_Checked" />
    </StackPanel>
```

```
        private void CheckBox_Checked(object sender, RoutedEventArgs e)
        {
            string msg = "";
            if ((bool)check1.IsChecked)
                msg += "Check1 ";
            if ((bool)check2.IsChecked)
                msg += "Check2 ";
            if ((bool)check3.IsChecked)
                msg += "Check3 ";
            Title = msg;
        }
```

</코드>
<결과>
[출력 결과]



</결과>

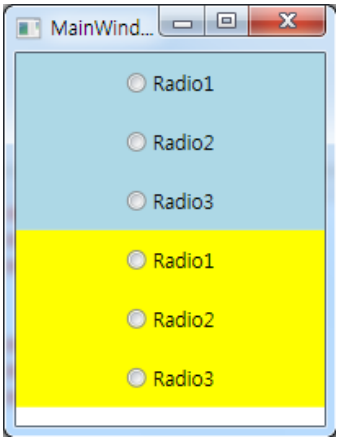
체크가 수행될 때마다 타이틀 바에 기록되며 체크가 취소될 때는 타이틀 바가 갱신되지 않는다. 이것은 Clicked 이벤트 핸들러에서 타이틀 바의 문자열을 변경하기 때문이다. 만약 체트가 취소될 때도 처리하려면 UnClicked 이벤트를 처리해야 한다. 또 C# 코드에서 IsChecked 를 bool 형식으로 변환한 이유는 IsChecked 가 null 을 포함한 bool? 형식이기 때문이다.

다음은 라디오 버튼의 그룹을 설정하는 예제로 부모 요소가 같은 라디오 버튼은 하나의 그룹으로 설정된다.

<코드>
[예제 04-05] 라디오 버튼

```
<StackPanel>
        <StackPanel Background="LightBlue">
            <RadioButton Content="Radio1" HorizontalAlignment="Center" Margin="10" />
            <RadioButton Content="Radio2" HorizontalAlignment="Center" Margin="10" />
            <RadioButton Content="Radio3" HorizontalAlignment="Center" Margin="10" />
        </StackPanel>
        <StackPanel Background="Yellow">
            <RadioButton Content="Radio1" HorizontalAlignment="Center" Margin="10" />
            <RadioButton Content="Radio2" HorizontalAlignment="Center" Margin="10" />
            <RadioButton Content="Radio3" HorizontalAlignment="Center" Margin="10" />
        </StackPanel>
    </StackPanel>
```

</코드>
<결과>
[출력 결과]



</결과>

라디오 버튼의 각각 다른 두 부모 요소인 StaticPanel 이 다르므로 두 그룹이 설정된다. 만약 하나의 부모 요소에서 다른 그룹을 만들고자 한다면 GroupName 속성을 사용할 수 있다.

다음은 부모 요소가 하나인 서로 다른 라디오 버튼의 그룹을 설정한 예제다.

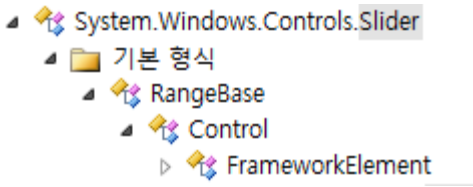
<코드>
[예제 04-06] GroupName 속성을 사용한 부모 요소 하나에 라이오 버튼 그룹

```
<StackPanel Background="LightBlue">
        <RadioButton GroupName="Group1" Content="Radio1" HorizontalAlignment="Center" Margin="10" />
        <RadioButton GroupName="Group1" Content="Radio2" HorizontalAlignment="Center" Margin="10" />
        <RadioButton GroupName="Group1" Content="Radio3" HorizontalAlignment="Center" Margin="10" />
        <RadioButton GroupName="Group2" Content="Radio1" HorizontalAlignment="Center" Margin="10" />
        <RadioButton GroupName="Group2" Content="Radio2" HorizontalAlignment="Center" Margin="10" />
        <RadioButton GroupName="Group2" Content="Radio3" HorizontalAlignment="Center" Margin="10" />
    </StackPanel>
```

</코드>
<결과>
[출력 결과]

</결과>
결과는 같고 단지 GroupName 속성을 사용하여 Group1과 Group2로 나눈 결과다.

일정한 범위 내에서 값을 선택해야 한다면 Slider 와 ScrollBar 를 사용할 수 있다. 두 컨트롤은 Orientation 속성을 사용하여 수평 모두와 수직 모드를 선택할 수 있으며 모두 RangeBase 클래스로부터 파생된다.



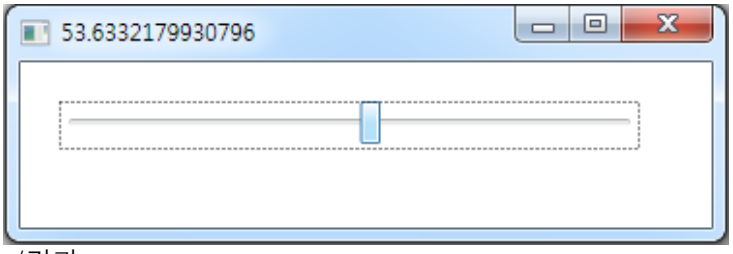
RangeBase 클래스의 주요 속성은 Minimum, Maximum, Value 속성으로 각각 컨트롤이 표할 최대 값의 범위, 최대 값의 범위 그리고 현재 선택된 값을 나타낸다. 또 화살표와 Page UP, Down 키를 눌렀을 때 조정 값의 크기를 SmallChang 속성과 LargeChange 속성으로 정의한다.

다음은 Slider 의 Value 값을 타이틀 바에 출력하는 예제다.

<코드>
[예제 04-07] Slider의 Value 확인

```
<StackPanel>
        <Slider Maximum="100" Minimum="0" Height="25" Width="300" HorizontalAlignment="Left"
Margin="20" ValueChanged="Slider_ValueChanged" />
    </StackPanel>

private void Slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
        {
            Title = ((System.Windows.Controls.Primitives.RangeBase)e.Source).Value.ToString();
        }
```

</코드>
<결과>
[출력 결과]



</결과>
Value 속성은 double 값이므로 double 값을 타이틀 바에 출력하며 Maximum 과 Minimum 을 각각 100에서 0으로 설정했다. Value 값이 변경될 때마다 ValueChanged 이벤트가 발생하므로 이벤트 핸들러를 등록하여 e.Source 를 RangeBase 형식으로 형변환하고 값을 타이틀 바에 출력한다.

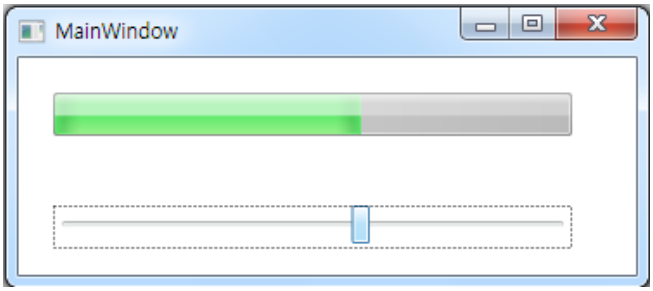
어떤 절차나 시간의 따른 작업 양을 표현하기 위해 ProgressBar 컨트롤을 사용한다. ProgressBar 컨트롤도 RageBase 클래스에서 파생되며 진행 상태를 Value 속성으로 확인할 수 있다.

다음은 ProgressBar 의 Value 값을 확인하는 예제다.

<코드>
[예제 04-08] ProgressBar의 Value 확인

```
<StackPanel>
        <ProgressBar Name="prog" Maximum="100" Minimum="0" Height="25" Width="300"
HorizontalAlignment="Left" Margin="20"/>
        <Slider Name="slider" Maximum="100" Minimum="0" Height="25" Width="300"
HorizontalAlignment="Left" Margin="20" ValueChanged="Slider_ValueChanged" />
    </StackPanel>

private void Slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
```

```
        {
            prog.Value = slider.Value;
        }
```
</코드>
<결과>
[출력 결과]



</결과>

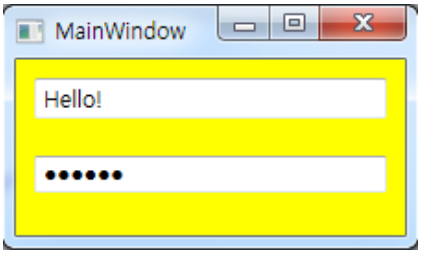
Slider 의 Value 값을 읽어 ProgressBar 의 Value 값으로 설정하며 각각의 Name 속성을 사용하여 컨트롤에 참조한다.

WPF 의 텍스트 컨트롤은 텍스트를 편집하고 보여주는 다양한 기능을 제공한다. TextBox 컨트롤은 텍스트를 간단하게 입력 받을 수 있는 컨트롤이며 PasswordBox 컨트롤은 비밀 번호를 입력 받을 수 있는 텍스트 컨트롤이다. 이반 텍스트 외에도 다양한 입력을 지원하는 RichTextBox 등을 제공한다.

다음 예제는 간단한 TextBox 컨트롤과 PasswordBox 컨트롤 예제다.

<코드>
[예제 04-09] TextBox와 PasswordBox 컨트롤

```
<StackPanel Background="Yellow">
        <TextBox Text="Hello!" Margin="10" VerticalAlignment="Center" />
        <PasswordBox Password="Hello!" Margin="10" VerticalAlignment="Center" />
    </StackPanel>
```
</코드>
<결과>
[출력 결과]



</결과>

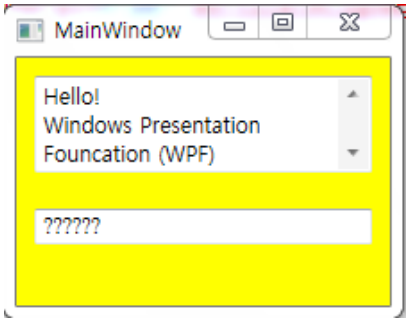
TextBox 컨트롤은 일반적인 텍스트를 화면에 보이며 PasswordBox 컨트롤은 화면에 직접 텍스트를 출력하지 않는다.

다음 예제는 TexBox 컨트롤과 Password 컨트롤의 속성을 변경한 예제다.

<코드>
[예제 04-10] TextBox 컨트롤의 Return 키 사용하기, Password 문자 변경하기

```
<StackPanel Background="Yellow">
        <TextBox VerticalScrollBarVisibility="Visible"
                 AcceptsReturn="True" Text="Hello!" Margin="10" VerticalAlignment="Center" />
        <PasswordBox PasswordChar="?"
                 Password="Hello!" Margin="10" VerticalAlignment="Center" />
    </StackPanel>
```

</코드>
<결과>
[출력 결과]



</결과>

TextBox 컨트롤은 AcceptsReturn 속성을 사용하여 Return 키 사용이 가능하며 PasswordBox 의 PasswordChar 속성은 PasswordBox 의 출력 문자 패턴을 변경할 수 있다.

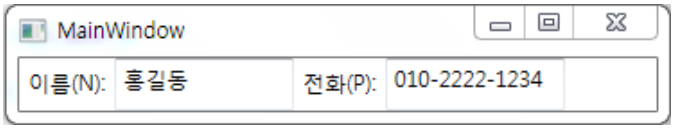
간단한 캡션을 컨트롤에 달고자 한다면 Label 컨트롤을 사용할 수 있다. TextBox 컨트롤처럼 캡션이 없는 컨트롤에 유용하게 사용되며 포커스 이동에 대한 동작까지도 담당할 수 있다.

다음 예제는 Label 컨트롤에서 포커스를 TextBox 컨트롤로 지정하는 예제다.

<코드>
[예제 04-11] Label 컨트롤

```
<StackPanel Orientation="Horizontal">
        <Label Target="{Binding ElementName=name}">이름(_N):</Label>
        <TextBox Name="name" Width="100" />
        <Label Target="{Binding ElementName=phone}">전화(_P):</Label>
        <TextBox Name="phone" Width="100" />
</StackPanel>
```

</코드>
<결과>
[출력 결과]



</결과>

Label 컨트롤의 Tartget 속성을 사용하여 포커스를 대상 컨트롤을 TextBox 로 설정한다. 여기서 대성 컨트롤은 바인딩 엔진을 사용하여 설정하며 바인딩에 대해서는 다음 장에서 공부한다.

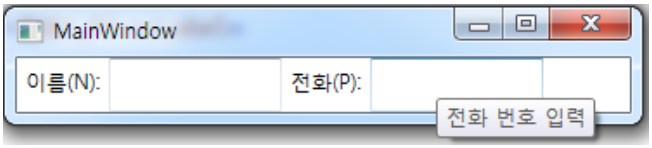
컨트롤의 간단한 설명이나 도움말, 알림 등을 사용자에게 전달하려면 ToolTip 컨트롤을 사용할 수 있다. ToolTip 컨트롤은 다른 요소와 연결되어 동작한다.

다음은 이름 TextBox 컨트롤과 전화 TextBox 컨트롤과 연결된 ToolTip 컨트롤 예제다.

<코드>
[예제 04-12] ToolTip 컨트롤

```
<StackPanel Orientation="Horizontal">
        <Label Target="{Binding ElementName=name}">이름(_N):</Label>
        <TextBox Name="name" Width="100" >
            <TextBox.ToolTip>
                <ToolTip Content="이름 입력" />
            </TextBox.ToolTip>
        </TextBox>
        <Label Target="{Binding ElementName=phone}">전화(_P):</Label>
        <TextBox Name="phone" Width="100" ToolTip="전화 번호 입력" >
        </TextBox>
</StackPanel>
```

</코드>
<결과>
[출력 결과]



</결과>
마우스 포인터가 각각의 컨트롤에 위치하면 ToolTip 메시지가 화면에 출력된다. 이름의 ToolTop 컨트롤은 속성 요소 문법을 사용하여 ToolTip 메시지를 출력했으며 전화의 ToolTip 컨트롤은 직접 ToolTip 속성을 문자열로 지정했다.

문자열이 아닌 다양한 형태의 ToolTip 메시지를 컨트롤과 연결하고자 한다면 속성 요소 문법을 사용하여 메시지를 출력할 수 있다. 다음은 속성 요소 문법을 사용한 ToolTip 컨트롤 예제다.

<코드>
[예제 04-13] ToolTip 컨트롤의 컨텐츠

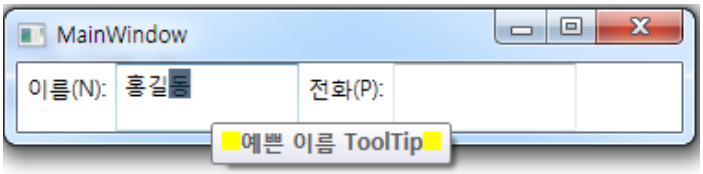
```
<StackPanel Orientation="Horizontal">
        <Label Target="{Binding ElementName=name}">이름(_N):</Label>
        <TextBox Name="name" Width="100" >
            <TextBox.ToolTip>
                <StackPanel Orientation="Horizontal">
                    <Rectangle Width="10" Height="10" Fill="Yellow" />
```

```
                    <TextBlock > <Bold> 예쁜 이름 ToolTip</Bold> </TextBlock>
                    <Rectangle Width="10" Height="10" Fill="Yellow" />
                </StackPanel>
            </TextBox.ToolTip>
        </TextBox>
        <Label Target="{Binding ElementName=phone}">전화(_P):</Label>
        <TextBox Name="phone" Width="100" ToolTip="전화 번호 입력" >
        </TextBox>
</StackPanel>
```

</코드>

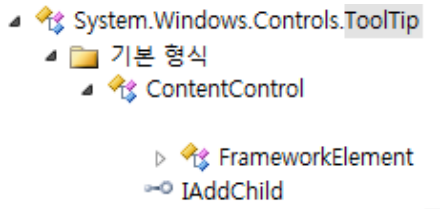
<결과>

[출력 결과]

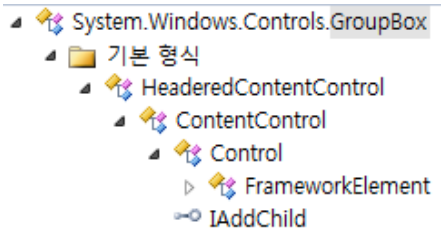


</결과>

TextBox 컨트롤과 연결된 TooTip 컨트롤로 몇 가지의 컨텐츠를 사용하여 예쁘게 꾸민 결과를 출력한다. ToolTip 도 컨텐츠 컨트롤이므로 얼마든지 다양한 컨텐츠로 표현할 수 있다. 다음은 ToolTip 컨트롤의 상속 구조를 보인다.



컨트롤을 하나의 그룹으로 다루고자 한다면 GroupBox 나 Expander 를 사용할 수 있다. 두 컨트롤은 ContentControl 을 상속받는 HeaderedContentControl 컨트롤이며 Header 와 Content 속성에 각각 컨텐츠를 가질 수 있다.



다음은 GroupBox 컨트롤과 Expander 컨트롤 예제다.

<코드>

[예제 04-14] GroupBox 컨트롤과 Expander 컨트롤

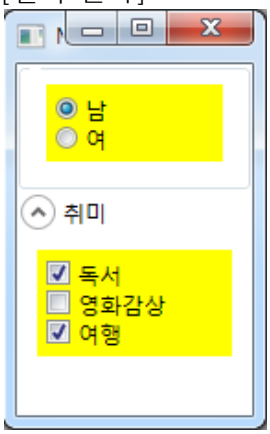
```
<StackPanel>
        <GroupBox Header="성별">
```

```
            <Border Margin="10" Background="Yellow" Padding="5">
                <StackPanel>
                    <RadioButton Content="남" />
                    <RadioButton Content="여" />
                </StackPanel>
            </Border>
        </GroupBox>
        <Expander Header="취미" IsExpanded="True">
            <Border Margin="10" Background="Yellow" Padding="5">
                <StackPanel>
                    <CheckBox Content="독서"/>
                    <CheckBox Content="영화감상" />
                    <CheckBox Content="여행" />
                </StackPanel>
            </Border>
        </Expander>
</StackPanel>
```

</코드>

<결과>

[출력 결과]



</결과>

두 컨트롤의 사용은 비슷하며 Expander 컨트롤은 IsExpanded 속성을 사용하여 컨트롤의 확대 축소를 조정할 수 있다.

HeaderedContentControl 클래스의 특징은 Header 와 Content 속성 모두에 컨텐츠를 가질 수 있다는 것이다.

다음은 Header 속성에 문자열이 아닌 컨텐츠를 표시한 예제다.

<코드>

[예제 04-15] Header 속성 사용하기

```
    <StackPanel>
        <GroupBox>
            <GroupBox.Header>
                <StackPanel Orientation="Horizontal">
                    <Ellipse Width="20" Height="20" Fill="Red" />
                    <TextBlock Padding="5"> <Bold>성별</Bold></TextBlock>
```

```
                    <Ellipse Width="20" Height="20" Fill="Blue" />
                </StackPanel>
            </GroupBox.Header>
            <Border Margin="10" Background="Yellow" Padding="5">
                <StackPanel>
                    <RadioButton Content="남" />
                    <RadioButton Content="여" />
                </StackPanel>
            </Border>
        </GroupBox>
        <Expander IsExpanded="True">
            <Expander.Header>
                <StackPanel Orientation="Horizontal">
                    <Ellipse Width="20" Height="20" Fill="Red" />
                    <TextBlock Padding="5"> <Bold>취미</Bold></TextBlock>
                    <Ellipse Width="20" Height="20" Fill="Blue" />
                </StackPanel>
            </Expander.Header>
            <Border Margin="10" Background="Yellow" Padding="5">
                <StackPanel>
                    <CheckBox Content="독서"/>
                    <CheckBox Content="영화감상" />
                    <CheckBox Content="여행" />
                </StackPanel>
            </Border>
        </Expander>
    </StackPanel>
```

</코드>

<결과>

[출력 결과]

</결과>

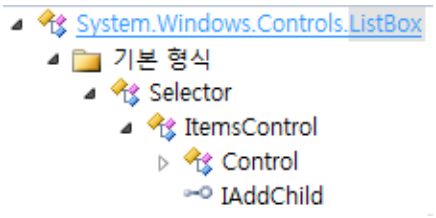
HeaderedContentControl 클래스의 Header 속성도 컨텐츠를 가질 수 있으므로 다양한 형태의 변경이 가능한다.

<절제목>

02. 리스트 형태의 컨트롤

</절제목>

리스트 형태의 컨트롤은 여러 개의 자식 아이템을 보여주기 위한 컨트롤로 WPF 는 ListBox, ComboBox, ListView, TreeView, TabControl 등 다양한 리스트 형태의 컨트롤을 제공한다. 이들 모두는 ItemsControl 로부터 파생된다. 다음은 ListBox 컨트롤의 상속 구조다.

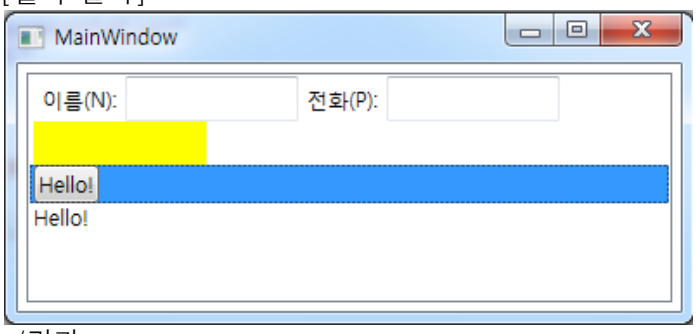


다음은 ListBox 컨트롤의 간단한 예제다.

<코드>
[예제 04-16] ListBox 컨트롤

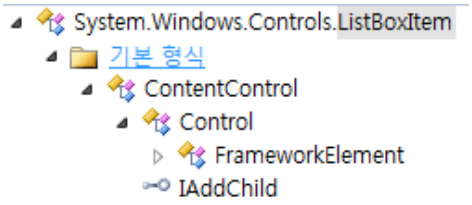
```
<ListBox Margin="5" >
    <StackPanel Orientation="Horizontal">
        <Label >이름(_N):</Label>
        <TextBox Name="name" Width="100" />
        <Label >전화(_P):</Label>
        <TextBox Name="phone" Width="100" />
    </StackPanel>
    <Rectangle Height="25" Width="100" Fill="Yellow"/>
    <Button>Hello!</Button>
    <TextBlock>Hello!</TextBlock>
</ListBox>
```

</코드>
<결과>
[출력 결과]

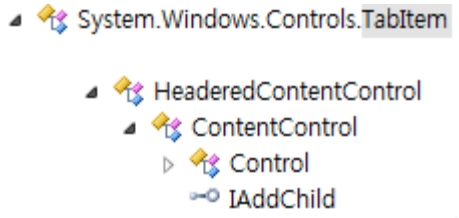


</결과>

ListBox 는 여러 개의 자식 아이템을 가질 수 있으며 이 자식 아이템은 Items 속성에 컨텐츠로 추가된다. Items 속성은 아이템 컨테이너라고 하는 컬렉션에 각각의 매핑된 아이템 요소들을 추가하는 형태다. 예로 ListBox 컨트롤의 Items 는 ListBoxItem 들을 추가하는 컬렉션이며 ComboBox 컨트롤의 Items 는 ComboBoxItem 들을 추가하는 컬렉션이다. XAML 문법에서는 자식 요소로 추가할 수 있으며 각각의 ListBoxItem 과 ComboBoxItem 들은 컨텐츠 컨트롤이며 다양한 형태의 컨텐츠를 가질 수 있다. 다음은 ListBoxItem 의 상속 구조를 보인다.



특히 TabItem 은 HeaderedContentControl 로 부터 파생하므로 Header 속성에 다양한 컨텐츠를 배치할 수 있다. 다음은 TabItem 요소의 상속 구조를 보인다.

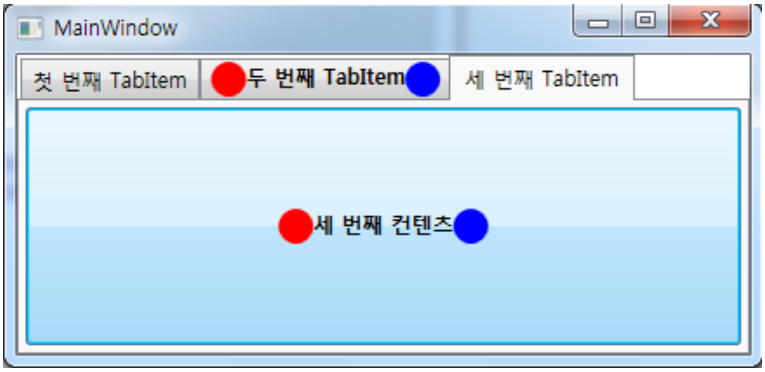


다음은 TabItem 의 Header 컨텐츠를 변경한 예제다.

<코드>
[예제 04-17] TabItem의 Header 컨텐츠

```
<TabControl>
        <TabItem Header="첫 번째 TabItem">
            <Button>첫 번째 컨텐츠</Button>
        </TabItem>
        <TabItem>
            <TabItem.Header>
                <StackPanel Orientation="Horizontal">
                    <Ellipse Width="20" Height="20" Fill="Red" />
                    <TextBlock > <Bold>두 번째 TabItem</Bold></TextBlock>
                    <Ellipse Width="20" Height="20" Fill="Blue" />
                </StackPanel>
            </TabItem.Header>
            <Button>두 번째 컨텐츠</Button>
        </TabItem>
        <TabItem Header="세 번째 TabItem">
            <Button>
                <StackPanel Orientation="Horizontal">
                    <Ellipse Width="20" Height="20" Fill="Red" />
```

```
                    <TextBlock > <Bold>세 번째 컨텐츠</Bold></TextBlock>
                    <Ellipse Width="20" Height="20" Fill="Blue" />
                </StackPanel>
            </Button>
        </TabItem>
    </TabControl>
```
</코드>
<결과>
[출력 결과]



</결과>

두 번째 TabItem 의 Header 에 컨텐츠로 StackPanel 을 사용하고 있으며 세 번째 TabItem 의 Content 에 버튼을 배치하고 StackPanel 로 컨텐츠를 변경하였다.

표형식의 데이터를 화면에 보여주고자 한다면 ListView 를 사용할 수 있다. ListView 는 View 속성에 열들을 정의하는 GridView 요소를 사용하여 열에 해당하는 GridViewColumn 을 정의할 수 있다. ListView 는 일반적으로 데이터 바인딩을 사용하여 ListView 의 아이템들을 표현하며 데이터 바인딩은 뒤 장에 공부한다.

다음은 간단한 ListView 예제다.

<코드>
[예제 04-18] ListView 컨트롤

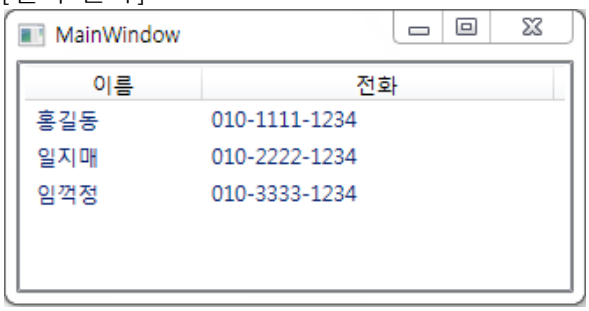
```
<Grid Name="grid">
        <ListView ItemsSource="{Binding}">
            <ListView.View>
                <GridView   ColumnHeaderToolTip="전화 번호 표시">
                    <GridViewColumn DisplayMemberBinding=
                            "{Binding Path=Name}"
                       Header="이름" Width="100" />

                    <GridViewColumn DisplayMemberBinding=
                            "{Binding Path=Phone}"
                       Header="전화" Width="200" />
                </GridView>
            </ListView.View>
        </ListView>
</Grid>
```

```
public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            grid.DataContext = new People();
        }
    }

    public class Person
    {
        public string Name { get; set; }
        public string Phone { get; set; }
    }

    public class People : ObservableCollection<Person>
    {
        public People()
        {
            Add(new Person() { Name = "홍길동", Phone = "010-1111-1234" });
            Add(new Person() { Name = "일지매", Phone = "010-2222-1234" });
            Add(new Person() { Name = "임꺽정", Phone = "010-3333-1234" });
        }
    }
```

</코드>

<결과>

[출력 결과]



</결과>

ListView 의 ItemsSource 는 데이터 바인딩을 사용하여 부모 요소의 DataContext 에 설정된 데이터 원본을 사용한다. DataContext 는 C#코드에서 설정하고 있으며 ListView 의 아이템은 GridView 의 자식 요소에 설정된 두 GridViewColumn 요소의 결합으로 표현된다.

각 GridViewColumn 은 이름과 전화 번호에 각각 매핑(바인딩)된다. 실제 데이터를 People 객체이며 People 객체의 Person 객체 각각이 GridView 를 통해 ListView 아이템으로 표현된다.

리스트 형태의 데이터 바인딩은 조금 복잡하지만 바인딩 엔진이 내부적으로 수행하는 동작을 이해하면 어렵지 않다. 데이터 바인딩은 뒷 장에서 공부한다.

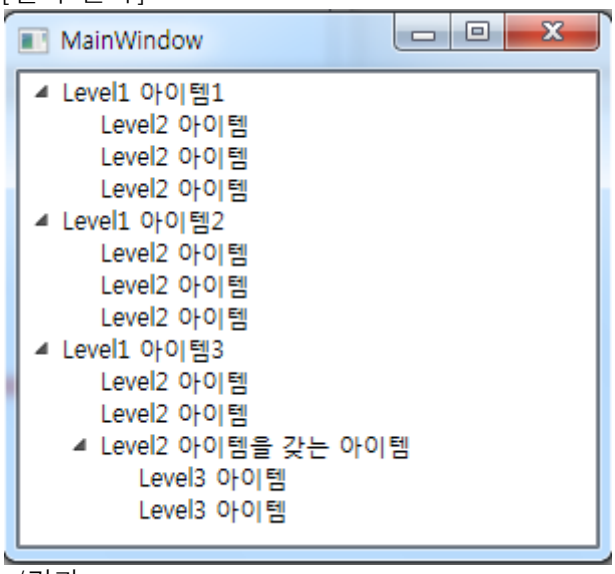
트리 자료구조처럼 계층적인 형태의 리스트를 나열하고자 한다면 TreeView 컨트롤을 사용할 수 있다. WPF 리스트 형태의 컨트롤은 대부분 데이터 바인딩과 함께 사용되며 마찬가지로 TreeView 도 데이터 바인딩과 함께 사용된다.

다음은 데이터 바인딩을 사용하지 않고 아이템을 계층적으로 구성한 TreeView 컨트롤 예제다.

<코드>
[예제 04-19] TreeViewItem

```
<TreeView>
        <TreeViewItem Header="Level1 아이템1" IsExpanded="True">
            <TreeViewItem Header="Level2 아이템" />
            <TreeViewItem Header="Level2 아이템" />
            <TreeViewItem Header="Level2 아이템" />
         </TreeViewItem>

        <TreeViewItem Header="Level1 아이템2" IsExpanded="True">
            <TreeViewItem Header="Level2 아이템" />
            <TreeViewItem Header="Level2 아이템" />
            <TreeViewItem Header="Level2 아이템" />
        </TreeViewItem>

        <TreeViewItem Header="Level1 아이템3" IsExpanded="True">
            <TreeViewItem Header="Level2 아이템" />
            <TreeViewItem Header="Level2 아이템" />
            <TreeViewItem Header="Level2 아이템을 갖는 아이템" IsExpanded="True">
                <TreeViewItem Header="Level3 아이템" />
                <TreeViewItem Header="Level3 아이템" />
            </TreeViewItem>
        </TreeViewItem>
</TreeView>
```

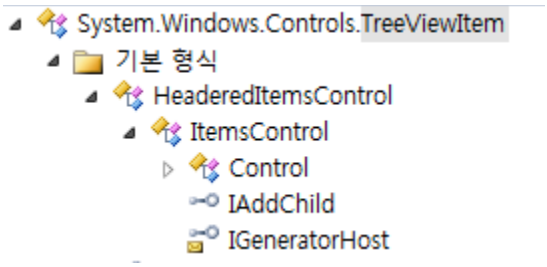
</코드>
<결과>
[출력 결과]



</결과>
TreeView 컨트롤은 TreeViewItem 들로 구성된다. TreeViewItem 은 HeaderedItemsControl 이며 Header 를 가지는 여러 아이템을 가질 수 있는 형태다. TreeViewItem 은 재귀적으로

TreeViewItem 을 포함할 수 있는 형태로 되어 있다. 간단한 구성이므로 예제만으로 어렵지 않게 이해할 수 있다. TreeView 는 데이터 바인딩과 함께 사용될 때 진가를 발휘 할 수 있다.

다음 그림은 TreeViewItem 의 상속 구조를 보인 것이다.



HeaderedItemsControl 은 ItemsControl 로부터 파생되어 ITemsControl 의 모든 기능을 사용할 수 있는 형태다.

<장제목>

5. 데이터 바인딩

</장제목>
WPF 는 데이터 바인딩이라는 기술을 통해서 프로그램 내부의 데이터를 사용자에게 전달하거나 사용자가 입력한 데이터를 프로그램 내부의 데이터로 변환하는 자동화 기술을 제공한다. 대부분의 애플리케이션은 데이터를 어떻게 사용자화 상호작용하는 가가 주요 내용이기 때문에 이 데이터 바인딩 기술은 프로그램 구조에서 상당히 중요한 부분을 차지한다.
WPF 의 데이터 바인딩을 이용하면 데이터의 변환, 정렬, 필터, 그룹화 등 데이터 사이의 관계를 정의하고 데이터 원본이라 부르는 데이터 자체와 UI 요소인 컨트롤 사이의 데이터를 보이거나 가져오는 일을 수행할 수 있다.

<절제목>

01. 일반적인 UI의 데이터 다루기

</절제목>

대부분의 애플리케이션은 UI 와 그 UI 와 연결된 데이터 사이의 통신이 필수적이며 이와 같은 작업을 수동으로 처리하기 위해 많은 코드를 필요로 한다. WPF 의 바인딩 시스템을 사용하면 수동으로 처리해야 하는 UI 와 데이터 사이의 통신의 통신을 쉽게 처리할 수 있을 뿐만 아니라 둘 사이의 형식 변환이나 정렬, 필터링, 그룹핑 등의 여러 작업을 자동으로 수행한다.

WPF 의 바인딩 엔진이 동작하는 원리를 이해하기 위해 이전에 사용했던 데이터와 UI 관계를 먼저 살펴보도록 하겠다.

다음은 간단한 이름과 전화 번호를 TextBox 컨트롤에 보이는 예제다.

<코드>
[예제 05-01] 데이터를 TextBox 컨트롤 UI에 보이기

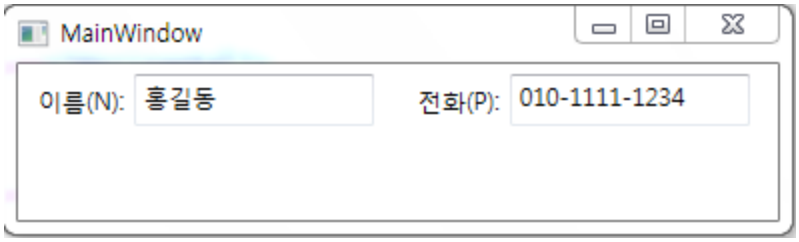
```
    <StackPanel>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Name="name" Width="120" />
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Name="phone" Width="120" />
            </StackPanel>
        </Grid>
    </StackPanel>

public partial class MainWindow : Window
    {
        private Person per = new Person()
                    { Name = "홍길동", Phone = "010-1111-1234" };
        public MainWindow()
        {
            InitializeComponent();
            UpdateNameToUI();
            UpdatePhoneToUI();
        }
        private void UpdateNameToUI()
```

```
        {
            name.Text = per.Name;
        }
        private void UpdatePhoneToUI()
        {
            phone.Text = per.Phone;
        }
    }
    public class Person
    {
        public string Name { get; set; }
        public string Phone { get; set; }
    }
```

</코드>

<결과>

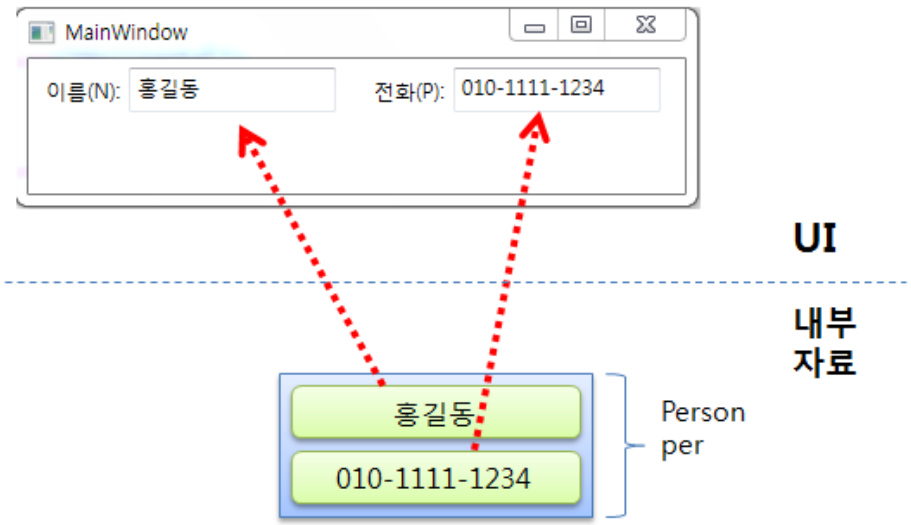
[출력 결과]



</결과>

Person 객체에 보관된 데이터 Name 과 Phone 의 데이터를 UI 의 TextBox 컨트롤에 보이고 있으며 만약 Person 객체의 데이터가 변경되면 그 때마다 UpdateNameToUI()와 UpdatePhoneToUI() 메소드를 호출하여 Person 객체를 갱신해야 한다.

다음은 위 예제에서 데이터를 UI 로 출력하는 동작을 표현한 것이다.

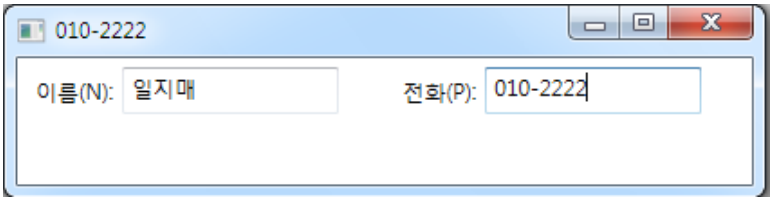


또, TextBox 컨트롤의 텍스트가 변경되면 내부의 데이터 객체(per)도 갱신되어야 한다. 대부분 UI 와 내부 데이터는 양방향 통신이 가능하도록 작성되기 때문이다.

<코드>
[예제 05-02] TextBox 컨트롤 UI의 내용을 데이터 객체에 갱신하기

```
<StackPanel>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Name="name" Width="120" TextChanged="name_TextChanged" />
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Name="phone" Width="120" TextChanged="phone_TextChanged" />
            </StackPanel>
        </Grid>
    </StackPanel>

        private void name_TextChanged(object sender, TextChangedEventArgs e)
        {
            per.Name = name.Text;
        }

        private void phone_TextChanged(object sender, TextChangedEventArgs e)
        {
            per.Phone = phone.Text;
            Title = per.Phone;
        }
```

</코드>
<결과>
[출력 결과]



</결과>

UI 가 변경되면 연결된 데이터 객체도 변경되어야 하므로 TextBox 컨트롤의 TextChanged 이벤트 핸들러를 등록하고 텍스트가 변경되면 데이터 객체(per)를 갱신한다. 상당히 많은 코드를 작성하는 것을 볼 수 있으며 UI 참조 변수가 변경되거나 데이터 객체를 사용하는 UI 요소가 늘어난다면 코드는 더 복잡해 진다.

다음은 Label 컨트롤을 만들고 각각의 이름과 전화번호를 출력하도록 구성한 예제다.

<코드>
[예제 05-03] 데이터 객체의 값을 공유하는 Label 컨트롤

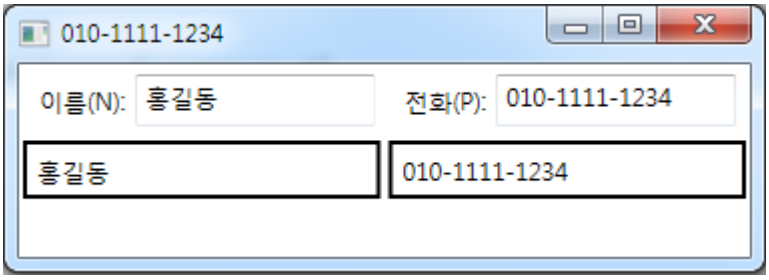
```
<StackPanel>
        <Grid >
            <Grid.ColumnDefinitions>
```

```
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Name="name" Width="120" TextChanged="name_TextChanged" />
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Name="phone" Width="120" TextChanged="phone_TextChanged" />
            </StackPanel>
        </Grid>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label    Height="25" Name="nameLabel"/>
            </Border>
            <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label Height="25" Name="phoneLabel"/>
            </Border>
        </Grid>
    </StackPanel>

        private void UpdateNameToUI()
        {
            name.Text = per.Name;
            nameLabel.Content = per.Name;
        }
        private void UpdatePhoneToUI()
        {
            phone.Text = per.Phone;
            phoneLabel.Content = per.Phone;
        }
```

</코드>

<결과>

[출력 결과]



</결과>

데이터 객체를 공유하는 UI 가 늘어나면서 내부 C# 코드가 늘어나고 복잡해 진다. 만약 UI 요소의 변경까지도 고려하여 설계한다면 코드는 더욱 복잡해 질 것이다.

WPF 의 바인딩 시스템은 리스트 형태의 아이템을 갖는 컨트롤에서 더욱 유용하게 사용된다. 다음은 바인딩 시스템을 사용하지 않고 ListBox 에 아이템을 출력하는 예제다.

<코드>
[예제 05-04] ListBox 컨트롤에 컬렉션 데이터 객체 출력

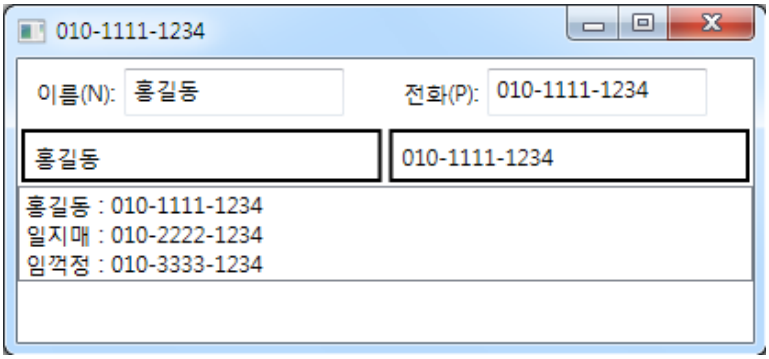
```
<StackPanel>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Name="name" Width="120" TextChanged="name_TextChanged" />
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Name="phone" Width="120" TextChanged="phone_TextChanged" />
            </StackPanel>
        </Grid>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label    Height="25" Name="nameLabel"/>
            </Border>
            <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label Height="25" Name="phoneLabel"/>
            </Border>
        </Grid>

        <ListBox Name="listbox"/>
    </StackPanel>

    public partial class MainWindow : Window
    {
        private People people = new People();
        private Person per;
        public MainWindow()
        {
            InitializeComponent();

            per = people[0];

            UpdateNameToUI();
            UpdatePhoneToUI();

            UpdateListBox();
        }
        private void UpdateNameToUI()
        {
            if (per == null)
                return;
            name.Text = per.Name;
            nameLabel.Content = per.Name;
        }
        private void UpdatePhoneToUI()
```

```
        {
            if (per == null)
                return;
            phone.Text = per.Phone;
            phoneLabel.Content = per.Phone;
        }
        private void UpdateListBox()
        {
            foreach (Person per in people)
                listbox.Items.Add(per.ToString());
        }

        private void name_TextChanged(object sender, TextChangedEventArgs e)
        {
            per.Name = name.Text;
        }

        private void phone_TextChanged(object sender, TextChangedEventArgs e)
        {
            per.Phone = phone.Text;
            Title = per.Phone;
        }
    }
    public class Person
    {
        public string Name { get; set; }
        public string Phone { get; set; }
        public override string ToString()
        {
            return Name + " : " + Phone;
        }
    }

    public class People : List<Person>
    {
        public People()
        {
            Add(new Person() { Name = "홍길동", Phone = "010-1111-1234" });
            Add(new Person() { Name = "일지매", Phone = "010-2222-1234" });
            Add(new Person() { Name = "임꺽정", Phone = "010-3333-1234" });
        }
    }
```

</코드>

<결과>

[출력 결과]



</결과>

ListBox 컨트롤과 매킹되는 데이터 객체를 표현하기 위해 List<Person> 컬렉션을 상속받은 People 컬렉션을 정의했으며 출력 결과에서 볼 수 있듯 ListBox 컨트롤의 People 컬렉션 객체의 Person 항목(아이템)을 하나하나 출력한다. 출력은 Person 객체의 ToString() 메소드가 반환하는 문자열 형태로 출력한다. 이름과 전화번호 TextBox 컨트롤에는 People 컬렉션 객체의 첫 번째 원소를 출력한다.

ListBox 컨트롤의 선택된 아이템을 TextBox 컨트롤과 Label 컨트롤에 갱신하기 위해서는 ListBox 컨트롤의 아이템 선택 변경 이벤트가 발생되었을 때 각각의 컨트롤들을 갱신하는 일도 만만치 않다.

다음 예제는 ListBox 컨트롤의 현재 아이템이 선택되면 다른 컨트롤들을 새롭게 갱신하는 예제다.

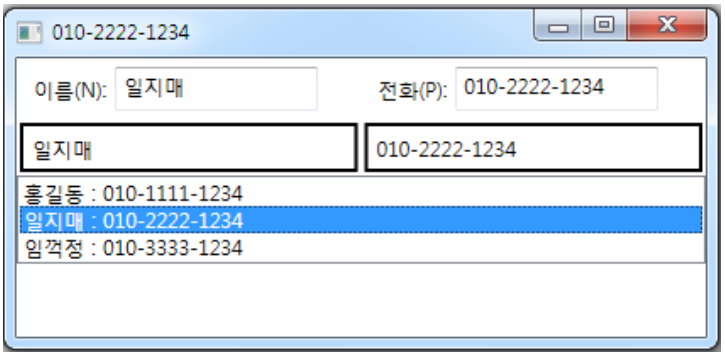
<코드>

[예제 05-05]

```
<StackPanel>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Name="name" Width="120" TextChanged="name_TextChanged" />
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Name="phone" Width="120" TextChanged="phone_TextChanged" />
            </StackPanel>
        </Grid>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label    Height="25" Name="nameLabel"/>
            </Border>
            <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label Height="25" Name="phoneLabel"/>
            </Border>
        </Grid>

        <ListBox Name="listbox" SelectionChanged="listbox_SelectionChanged" />
    </StackPanel>

        private void listbox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
```

```
            if (listbox.SelectedIndex >= 0)
            {
                per = people[listbox.SelectedIndex];
                UpdateNameToUI();
                UpdatePhoneToUI();
            }
        }
```

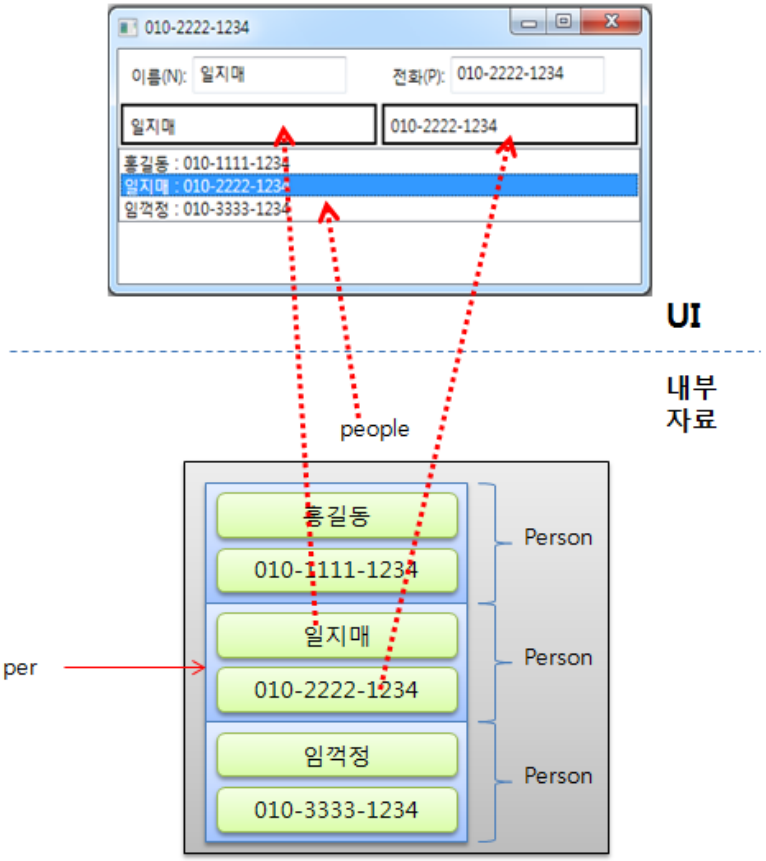
</코드>
<결과>
[출력 결과]



</결과>

ListBox 컨트롤의 아이템으로 일지매가 선택되면 각각의 이름과 전화번호에 해당하는 TextBox 컨트롤과 Label 컨트롤이 갱신되도록 SelectionChanged 이벤트를 처리하고 각각의 UpdateXXUI() 메소드를 호출한다. 이 또한 WPF 바인딩 시스템을 사용하면 코드 작업 없이 해결할 수 있다.

다음은 위 내용을 그림으로 표현한 것이다.



다음은 WPF 바인딩 시스템을 사용하지 않고 구현한 간단한 전화번호 추가, 삭제 프로그램이다. UI 와 내부 데이터 사이의 통신을 위한 코드의 양과 복잡도를 잘 살펴 보기 바란다. 몇몇 오류처리와 추가, 삭제, 변경 알고리즘도 가장 간단하게 작성했지만 더 좋은 오류처리 코드와 알고리즘을 작성하기 위해서는 더 많은 작업이 필요하게 된다.

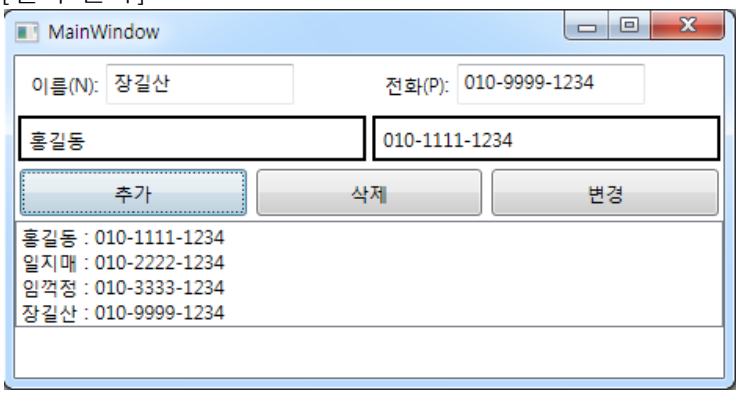
<코드>
[예제 05-06] 간단한 전화번호 추가, 삭제 프로그램

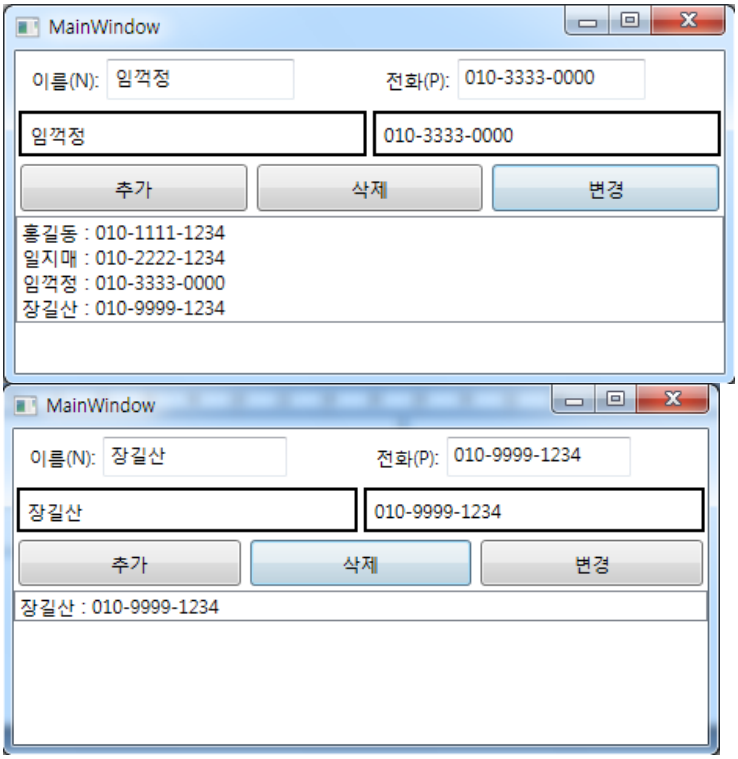
```
<StackPanel>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Name="name" Width="120" />
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Name="phone" Width="120" />
            </StackPanel>
        </Grid>
        <Grid >
```

```
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label    Height="25" Name="nameLabel"/>
            </Border>
            <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label Height="25" Name="phoneLabel"/>
            </Border>
        </Grid>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Button Name="addButton" Margin="3" Height="30" Grid.Column="0" Content="추가"
Click="addButton_Click" />
            <Button Name="removeButton" Margin="3" Height="30" Grid.Column="1" Content="삭제"
Click="removeButton_Click" />
            <Button Name="updateButton" Margin="3" Height="30" Grid.Column="2" Content="변경"
Click="updateButton_Click" />
        </Grid>

        <ListBox Name="listbox" SelectionChanged="listbox_SelectionChanged" />
    </StackPanel>

public partial class MainWindow : Window
    {
        private People people = new People();
        private Person per;
        public MainWindow()
        {
            InitializeComponent();

            per = people[0];

            UpdateNameToUI();
            UpdatePhoneToUI();

            UpdateListBox();
        }
        private void UpdateNameToUI()
        {
            if (per == null)
            {
                name.Text = "";
                nameLabel.Content = "";
            }
            else
            {
                name.Text = per.Name;
                nameLabel.Content = per.Name;
            }
        }
        private void UpdatePhoneToUI()
```

```
{
    if (per == null)
    {
        phone.Text = "";
        phoneLabel.Content = "";
    }
    else
    {
        phone.Text = per.Phone;
        phoneLabel.Content = per.Phone;
    }
}
private void UpdateListBox()
{
    listbox.Items.Clear();
    foreach (Person p in people)
        listbox.Items.Add(p.ToString());
}

private void listbox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (listbox.SelectedIndex >= 0)
    {
        per = people[listbox.SelectedIndex];
        UpdateNameToUI();
        UpdatePhoneToUI();
    }
}

private void addButton_Click(object sender, RoutedEventArgs e)
{
    if( name.Text == "" || phone.Text == "")
        return;

    people.Add(new Person() { Name = name.Text, Phone = phone.Text });
    // 리스트 박스의 아이템을 갱신한다.
    UpdateListBox();
}

private void removeButton_Click(object sender, RoutedEventArgs e)
{
    if (listbox.SelectedIndex >= 0)
    {
        people.RemoveAt(listbox.SelectedIndex);

        // 컬렉션에 원소가 없다면 리슽의 현재 아이템이 없도록(per =null) 한다.
        if (people.Count == 0)
            per = null;
        else
            per = people[0];

        // 모든 UI 컨트롤을 갱신한다.
        UpdateNameToUI();
        UpdatePhoneToUI();
        UpdateListBox();
    }
}
```

```
        private void updateButton_Click(object sender, RoutedEventArgs e)
        {
            if (name.Text == "" || phone.Text == "")
                return;

            per.Name = name.Text;
            per.Phone = phone.Text;

            UpdateNameToUI();
            UpdatePhoneToUI();
            UpdateListBox();
        }
    }
    public class Person
    {
        public string Name { get; set; }
        public string Phone { get; set; }
        public override string ToString()
        {
            return Name + " : " + Phone;
        }
    }

    public class People : List<Person>
    {
        public People()
        {
            Add(new Person() { Name = "홍길동", Phone = "010-1111-1234" });
            Add(new Person() { Name = "일지매", Phone = "010-2222-1234" });
            Add(new Person() { Name = "임꺽정", Phone = "010-3333-1234" });
        }
    }
</코드>
<결과>
[출력 결과]
```





</결과>

각각의 예제는 전화번호 추가, 삭제, 변경의 결과를 보인 것이다.

다음 코드는

```
public MainWindow()
{
    InitializeComponent();

    per = people[0];

    UpdateNameToUI();
    UpdatePhoneToUI();

    UpdateListBox();
}
```

ListBox 의 현재 아이템을 설정하기 위해 Person 형식의 per 객체를 사용했으며 people 컬렉션이 적어도 하나의 원소를 가진다는 가정하에 people[0]를 사용하고 있다. UpdateXX() 메소드는 각각의 컨트롤 UI 에 데이터를 출력한다.

다음 코드는

```
private void UpdateNameToUI()
{
    if (per == null)
    {
        name.Text = "";
        nameLabel.Content = "";
    }
    else
    {
        name.Text = per.Name;
```

```
            nameLabel.Content = per.Name;
        }
    }
    private void UpdatePhoneToUI()
    {
        if (per == null)
        ...
        else
        ...
    }
```

현재 아이템이 존재한다면 이름과 전화번호를 각각의 TextBox 컨트롤과 Label 컨트롤에 출력하는 코드로 현재 아이템이 존재하지 않는다면 문자열 없음을 출력한다. 현재 아이템이 존재하다가 '삭제' 버튼의 실행으로 현재 아이템이 존재하지 않을 수 있기 때문이다. 오류처리를 다른 형태로 한다면 코드의 복잡도가 커진다.

다음 코드는

```
    private void UpdateListBox()
    {
        listbox.Items.Clear();
        foreach (Person p in people)
            listbox.Items.Add(p.ToString());
    }
```

ListBox 컨트롤을 갱신하는 메소드로 리스트 아이템을 모두 비우고 people 의 모든 원소를 ListBox 컨트롤에 출력한다. 출력은 Person 객체의 오버라이드한 ToString() 메소드를 사용하여 출력한다. 만약 더 다양한 출력이 필요하다면 코드가 복잡도가 커진다.

다음 코드는

```
    private void listbox_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        if (listbox.SelectedIndex >= 0)
        {
            per = people[listbox.SelectedIndex];
            UpdateNameToUI();
            UpdatePhoneToUI();
        }
    }
```

ListBox 컨트롤의 셀이 변경될 때 발생하는 이벤트 핸들러로 선택된 아이템의 인덱스 번호에 해당하는 Person 객체를 people 에서 가져와 현재 아이템 per 객체로 설정하고 TextBox 컨트롤과 Label 컨트롤을 갱신한다. 바인딩 시스템 없이 직접 UI 를 갱신하다 보니 현재 아이템이 변경될 때 UI 를 갱신하는 코드가 '코드 중복' 되는 것을 확인할 수 있다. 이는 UI 컨트롤이 더 늘어나면 기하급수적으로 코드의 무질서 정도가 커지는 현상을 발생시킨다.

다음 코드는

```
    private void addButton_Click(object sender, RoutedEventArgs e)
```

```
{
    if( name.Text == "" || phone.Text == "")
        return;

    people.Add(new Person() { Name = name.Text, Phone = phone.Text });
    // 리스트 박스의 아이템을 갱신한다.
    UpdateListBox();
}
```

추가 버튼이 눌리면 TexBox 컨트롤의 텍스트를 확이하고 people 컬렉션 객체에 추가한 후 ListBox 컨트롤을 갱신한다.

다음 코드는

```
private void removeButton_Click(object sender, RoutedEventArgs e)
{
    if (listbox.SelectedIndex >= 0)
    {
        people.RemoveAt(listbox.SelectedIndex);

        // 컬렉션에 원소가 없다면 리슽의 현재 아이템이 없도록(per =null) 한다.
        if (people.Count == 0)
            per = null;
        else
            per = people[0];

        // 모든 UI 컨트롤을 갱신한다.
        UpdateNameToUI();
        UpdatePhoneToUI();
        UpdateListBox();
    }
}
```

삭제 버튼이 눌리면 현재 선택된 아이템의 인덱스 번호를 확인하여 people 컬렉션 원소를 삭제한다. people 컬렉션이 원소를 가지지 않으면 현재 아이템 객체 per 를 null 로 설정하고 아니면 0인덱스 번호를 현재 아이템 객체로(per) 설정한다. 여기서 좀더 나은 삭제 알고리즘을 구현하려면 선택된 ListBox 컨트롤의 선택된 셀에 해당하는 people 컬렉션 아이템을 제거한 후 컬렉션의 다음 원소가 현재 아이템 객체가 되어야 하고 다음 원소가 없다면 이전 원소를 현재 아이템 객체로 설정하는 등의 알고리즘을 구현해야 한다. 코드가 조금 복잡해 지므로 위와 같이 삭제 알고리즘을 구현하였다. UpdateXX() 메소드는 각각의 UI 컨트롤를 갱신한다. 위 코드에서도 UI 컨트롤이 추가되면 C#코드도 변경되어야 하므로 확장성이나 유지 보수가 상당히 어렵다는 것을 알 수 있다.

다음 코드는

```
private void updateButton_Click(object sender, RoutedEventArgs e)
{
    if (name.Text == "" || phone.Text == "")
        return;
```

```
            per.Name = name.Text;
            per.Phone = phone.Text;

            UpdateNameToUI();
            UpdatePhoneToUI();
            UpdateListBox();
        }
```

현재 아이템 객체(per)를 갱신하고 UI 컨트롤에 출력한다.

마지막으로 다음 코드는

```
    public class Person
    {
        public string Name { get; set; }
        public string Phone { get; set; }
        public override string ToString()
        {
            return Name + " : " + Phone;
        }
    }

    public class People : List<Person>
    {
        public People()
        {
            Add(new Person() { Name = "홍길동", Phone = "010-1111-1234" });
            Add(new Person() { Name = "일지매", Phone = "010-2222-1234" });
            Add(new Person() { Name = "임꺽정", Phone = "010-3333-1234" });
        }
    }
```

이름과 전화번호를 추상화한 Person 클래스 정의와 이 Person 객체를 관리하는 List<Person> 형식의 People 컬렉션 클래스의 정의다. Person 은 자신의 내용을 출력하기 위해 ToString() 메소드를 어버라이드한다. 애플리케이션이 문자열보다 더 다양한 형태의 출력을 요구한다면 코드 수정이 불가피하게 된다. People 은 생성자에서 기본적인 세 Person 객체를 추가하여 애플리케이션 시작 시 항상 현재 아이템('홍길동')을 갖도록 한다.

<절제목>

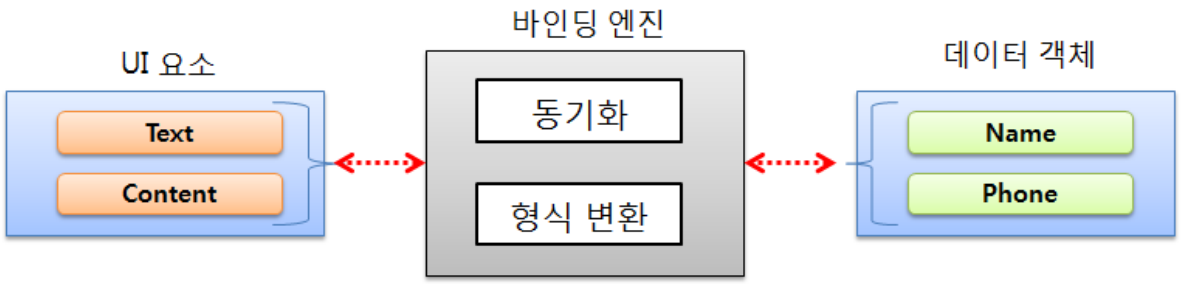
02. 데이터 바인딩을 이용한 UI 다루기

</절제목>

지금까지 진행했던 전화번호 추가, 삭제 프로그램을 WPF 바인딩 시스템을 사용하여 변경해 보도록 한다. WPF 의 바인딩 시스템을 사용하면 지금까지 우리가 수동으로 진행했던 UI 와 데이터 객체의 통신(동기화)를 자동으로 구축할 수 있다.

바인딩 시스템의 핵심 기능은 두 가지로 UI 의 변경을 데이터 객체로 데이터 객체의 변경을 UI 로 자동 통신하는 동기화 기능과 서로 다른 형식의 동기화를 위한 적절한 형식 변환 기능이다.

바인딩 시스템은 UI 의 의존속성과 데이터 객체의 속성이 연결되는 형태로 동작한다. 다음은 바인딩 시스템의 기능을 그림으로 표현한 것이다.



다음은 바인딩을 이용하여 이름과 전화번호 TextBox 컨트롤 UI 에 Person 객체를 출력하는 예제다.

<코드>

[예제 05-07]

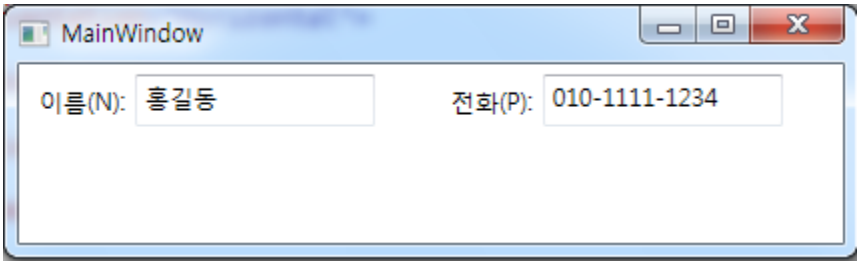
```
<StackPanel Name="panel">
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Text="{Binding Path=Name}" Width="120"/>
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Text="{Binding Path=Phone}" Width="120"/>
            </StackPanel>
```

```
        </Grid>
    </StackPanel>

    public partial class MainWindow : Window
    {
        private Person per = new Person() { Name = "홍길동",
                    Phone = "010-1111-1234" };
        public MainWindow()
        {
            InitializeComponent();
            panel.DataContext = per;
        }
    }
    public class Person
    {
        public string Name { get; set; }
        public string Phone { get; set; }
    }
```

</코드>

<결과>

[출력 결과]



</결과>

여기서 핵심은 XAML 코드에서 UI 요소의 의존 속성에 바인딩을 위한 코드를 작성했으며

```
<TextBox Text="{Binding Path=Name}" Width="120"/>
```

XAML 에서는 이것이 바인딩 코드의 전부다.

이제 UI 의 의존 속성 Text 와 바인딩 할(연결할) 데이터 객체(데이터 원본이라한다)를 설정하는 것 뿐이다. 이것은 C# 코드에서

```
panel.DataContext = per;
```

이와 같이 설정했다. 이후에 보면 알겠지만 이 데이터 원본 또한 XAMLE 에서 설정 가능하다.

이제 바인딩 문법부터 하나씩 살펴보자

```
<TextBox Text="{Binding Path=Name}" Width="120"/>
```

문법은

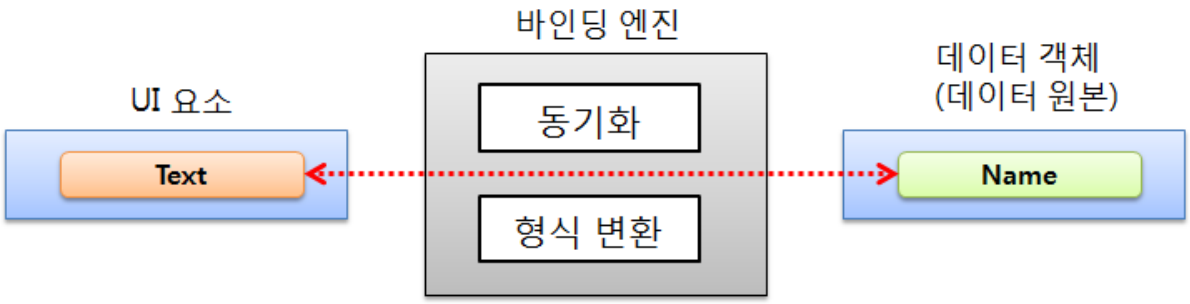
```
<TextBox >
  <TextBox.Text>
      <Binding Path=" Age" />
  </TextBox.Text>
</TextBox>
```

를 간단하게 작성한 XAML 문법으로 마크업 확장이라 한다.

UI 의 Text 의존속성과 데이터 원본 객체의 Name 속성을 바인딩 한다. 데이터 원본 객체의 속성은 의존 속성이나 일반 .Net 속성 등 모두 가능 하지만 UI 의 바인딩 속성은 꼭 의존 속성만 가능하다 바인딩 시스템이 의존 속성을 사용하여 동작하기 때문이다.

또 바인딩 원본은 구체적으로 기록되지 않으며 이것으로 UI 의 Text 의존 속성은 얼마든지 데이터 원본을 변경할 수 있는 유연성을 갖게 된다.

다음은 바인딩 코드를 그림으로 표현한 것이다.



또 바인딩 데이터 원본은 UI 요소에 설정할 수 있지만 바인딩 UI 요소에 데이터 원본이 없다면 부모 요소를 탐색하며 바인딩 데이터 원본을 찾게 된다.

이런 설계를 더 많이 사용하는데 이와 같이 구성하면 부모 요소 하나를 사용하는 모든 자식 요소들이 설정된 데이터 원본을 공유 하여 각각의 자식 요소 변경을 갱신할 필요 없이 자동으로 자식 요소들의 변경을 갱신할 수 있도록 한다.

여기에 사용되는 속성이 DataContext 속성이며 부모 요소의 트리를 검색하며 데이터 원본을 찾는다.

위 예제는 StackPanel(panel)의 DataContext 가 C# 코드에서

```
panel.DataContext = per;
```

이처럼 설정되어 데이터 원본으로 사용되며 자식 요소인 이름 TextBox 컨트롤과 전화번호 TextBox 컨트롤이 같은 데이터 원본인 per 객체가 된다.

다음 예제는 부모 요소 StackPanel 의 DataContext 속성의 데이터 원본을 공유하는 TextBox 컨트롤과 Label 컨트롤 UI 의 바인딩 예제다.

<코드>
[예제 05-08] 부모 요소의 DataContext 데이터 원본을 공유하는 자식 요소 바인딩

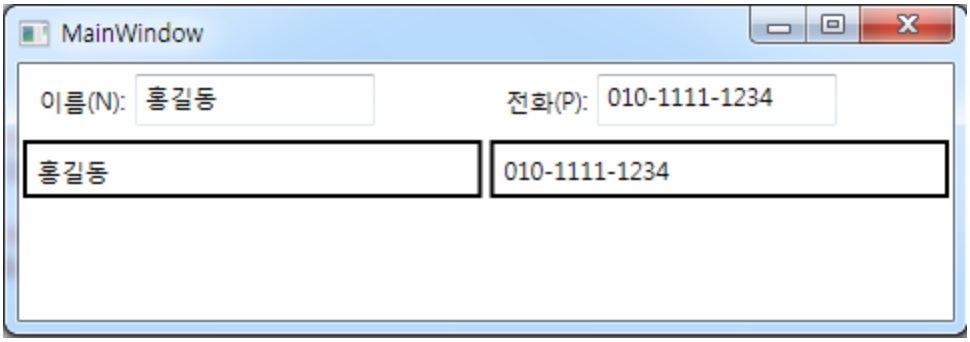
```
<StackPanel Name="panel">
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
```

```
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
            <Label >이름(_N):</Label>
            <TextBox Text="{Binding Path=Name}" Width="120"/>
        </StackPanel>
        <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
            <Label >전화(_P):</Label>
            <TextBox Text="{Binding Path=Phone}" Width="120"/>
        </StackPanel>
    </Grid>
    <Grid >
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
            <Label    Height="25" Content="{Binding Path=Name}"/>
    </Border>
    <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
            <Label Height="25" Content="{Binding Path=Phone}"/>
    </Border>
  </Grid>
  </StackPanel>
```

</코드>

<결과>

[출력 결과]



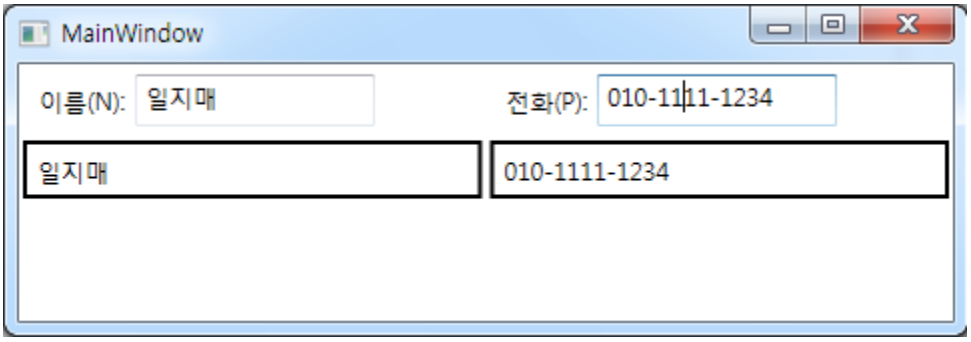
</결과>

단지 다음 코드를 추가한 것 외에는 다른 곳이 없다. 부모 요소 트리를 검색하여 데이터 원본을 설정하므로 Label 의 데이터 원본은 StackPanel 객체에 설정된 per 객체가 된다.

```
    <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
            <Label    Height="25" Content="{Binding Path=Name}"/>
    </Border>
    <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
            <Label Height="25" Content="{Binding Path=Phone}"/>
    </Border>
```

만약 이름이나 전화번호의 TextBox 컨트롤을 수정하여 다른 곳으로 포커스를 이동하면 데이터 원본이 변경되는 것을 확인할 수 있다. 기본적으로 바인딩 엔진에서 데이터 원본과 UI 속성의 동기화를 수행하며 기본 동작은 포커스가 다른 곳으로 변경될 때 동기화 되도록 설정되어 있기 때문이다. 당연히 바인딩 문법을 사용하여 변경 가능하다.

다음 그림은 '임꺽정' TextBox 컨트롤의 이름을 '일지매'로 변경한 그림이다.



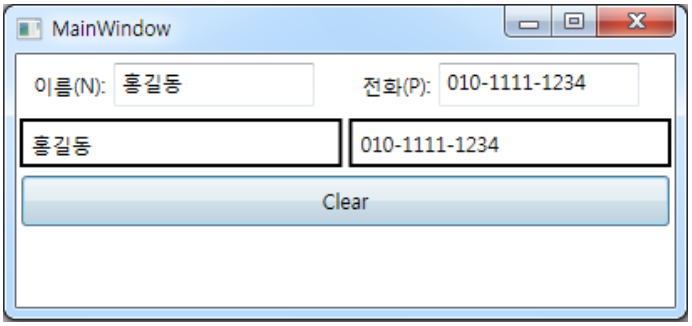
UI 가 변경되면 데이터 원본이 변경되도록 동기화된다는 것을 확인했으니 이번에는 반대로 데이터 원본이 변경되면 UI 가 변경되는지 바인딩을 통해 알아 보도록 하자.

다음은 Clear 버튼을 클릭하면 이름과 전화번호를 다시 작성할 수 있도록 모든 텍스트를 지우는 예제다.

<코드>
[예제 05-09] Clear 버튼을 이용한 TextBox 컨트롤 지우기

```
<Grid >
            <Button Name="eraseButton" Margin="3" Height="30" Grid.Column="2" Content="Clear"
Click="eraseButton_Click" />
        </Grid>

         private void eraseButton_Click(object sender, RoutedEventArgs e)
        {
            per.Name = "";
            per.Phone = "";
        }
```

</코드>
<결과>
[출력 결과]



</결과>

결과에서 볼 수 있듯 Clear 버튼을 클릭하면 데이터 원본 객체인 per 의 Name 과 Phone 의 모든 텍스트가 지워지지만 UI 에 변경이 적용되지 않는다. 이것은 per.Name 이나 per.Phone 속성은 일반 .Net 속성이므로 속성의 변경(데이터 원본의 변경) 사실을 UI 에 동기화하지 못한다.

```
        private void eraseButton_Click(object sender, RoutedEventArgs e)
```

```
        {
            per.Name = "";
            per.Phone = "";
        }
```

이 코드에서 per 의 속성을 변경하지만 이 속성과 바인딩되어 있는 UI 는 변경되지 않는다.

데이터 원본(원본 객체의 속성)의 변경을 UI 에 통보하기 위한 바인딩을 사용하려면 몇 가지 약속된 절차를 구현해야 한다.

다음은 데이터 원본의 변경을 UI 에 동기화하기 위한 예제다.

<코드>

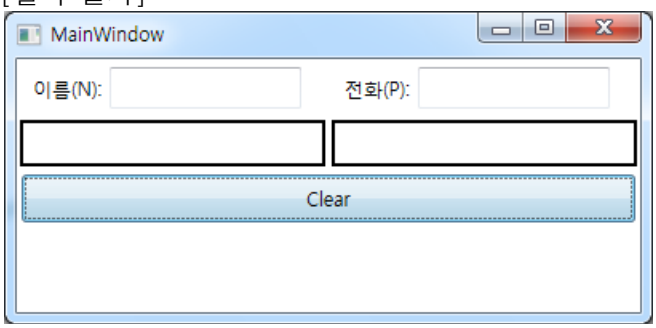
[예제 05-10] Clear 버튼으로 데이터 원본을 변경하고 UI에 동기화 하기

```
    public class Person : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        private string name;
        public string Name
        {
            get { return name; }
            set
            {
                name = value;
                if (PropertyChanged != null)
                    PropertyChanged(this, new PropertyChangedEventArgs("Name"));
            }
        }
        private string phone;
        public string Phone
        {
            get { return phone; }
            set
            {
                phone = value;
                if (PropertyChanged != null)
                    PropertyChanged(this, new PropertyChangedEventArgs("Phone"));
            }
        }
    }
```

</코드>

<결과>

[출력 결과]



</결과>
이제 Clear 를 클릭하면 원본 객체의 변경이 UI 에 동기화 된다.

이처럼 동작하기 위해서는 데이터 원본 객체의 속성(Name, Phone)은 속성이 set(변경)될 때 변경 사실을 INotifyPropertyChanged 인터페이스를 통해 알려야 한다. INotifyPropertyChanged 는 하나의 이벤트 public event PropertyChangedEventHandler PropertyChanged 를 멤버로 가지고 있는 인터페이스로 set 프로퍼티에서 호출하도록 약속되어 있다.

다음과 같이

```
public string Name
{
    get { return name; }
    set
    {
        name = value;
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs("Name"));
    }
}
```

PropertyChanged 는 첫 인수로 데이터 원본 객체로 두 번째 인수로 PropertyChangedEventArgs("Name")으로 이벤트를 발생시킨다. 이때 "Name" 은 데이터 원본에 참여하는 속성의 이름이다.

이제 다음 코드는

```
private void eraseButton_Click(object sender, RoutedEventArgs e)
{
    per.Name = "";
    per.Phone = "";
}
```

바인딩된 UI 를 변경한다.

데이터 아일랜드라고 부르는 각 요소에 표현할 수 있는 리소스를 사용하면 데이터 원본을 C# 코드에서 설정하지 않고 XAML 코드에서 설정하고 사용할 수 있다. (WPF 의 리소스는 기존 리소스 개념보다 조금 확장된 개념으로 명명될 수 있는 데이터를 일반적으로 리소스라고 하며 기존의 사용되었던 리소스도 모두 리소스라고 한다.)

다음은 XAML 상에 리소스를 정의하고 데이터 원본으로 사용한 예제다.

<코드>
[예제 05-11] XAML 리소스를 사용한 데이터 원본

```
<Window x:Class="ex05_11.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"
        xmlns:local="clr-namespace:ex05_11">
```

```
    <Window.Resources>
        <local:Person x:Key="person" Name="홍길동" Phone = "010-1111-1234" />
    </Window.Resources>
    <StackPanel DataContext="{StaticResource person}">
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Text="{Binding Path=Name}" Width="120"/>
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Text="{Binding Path=Phone}" Width="120"/>
            </StackPanel>
        </Grid>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label    Height="25" Content="{Binding Path=Name}"/>
            </Border>
            <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label Height="25" Content="{Binding Path=Phone}"/>
            </Border>
        </Grid>
        <Grid >
            <Button Name="eraseButton" Margin="3" Height="30" Grid.Column="2" Content="Clear"
Click="eraseButton_Click" />
        </Grid>

    </StackPanel>
</Window>

public partial class MainWindow : Window
    {
        Person per = null;
        public MainWindow()
        {
            InitializeComponent();
            per = (Person)FindResource("person");
        }

        private void eraseButton_Click(object sender, RoutedEventArgs e)
        {
            per.Name = "";
            per.Phone = "";
        }
    }
```

</코드>

결과는 같다.

C# 코드에 정의된 Person 클래스를 XAML 에서 사용하기 위해 C#코드상의 네임스페이스를 XAML 네임스페이스로 사용하기 위해

```
xmlns:local="clr-namespace:ex05_11"
```

를 사용하여 ex05_11 은 local 로 XAML 에서 사용하며

윈도우 요소에서 리소를 정의하기 위해 속성 요소문법 Window.Resources 를 사용하여 다음과 같이 Person 객체를 정의하고 생성하도록 하는 코드다.

```
<Window.Resources>
    <local:Person x:Key="person" Name="홍길동" Phone = "010-1111-1234" />
</Window.Resources>
```

리소스는 이제 x:Key 값 "person" 을 사용하여 XAML 이나 C#에서 사용할 수있다.

이제 바인딩 원본을 설정하기 위해 C# 코드를 사용하지 않고 다음과 같이

```
<StackPanel DataContext="{StaticResource person}">
```

XAML 코드에서 바로 설정하여 사용했으며

Clear 버튼이 클릭되었을 시 person 객체를 변경하기 위해

```
per = (Person)FindResource("person");
```

처럼 리소스의 키값으로 "person" 객체를 찾는 FindResource() 메소드를 이용하여 XAML 상의 객체의 참조를 찾고

```
per.Name = "";
per.Phone = "";
```

처럼 변경한다.

데이터 원본을 명시적으로 바인딩 문법을 사용하여 설정할 수 있으며 이때는 부모 요소의 DataContext 에 설정된 데이터 원본보다 우선한다.

다음은 TextBox 컨트롤은 부모 요소의 DataContext 에 설정된 데이터 원본을 사용하고 Label 컨트롤은 명시적인 데이터 원본을 사용하는 예제다.

<코드>

[예제 05-12] 명시적 데이터 원본

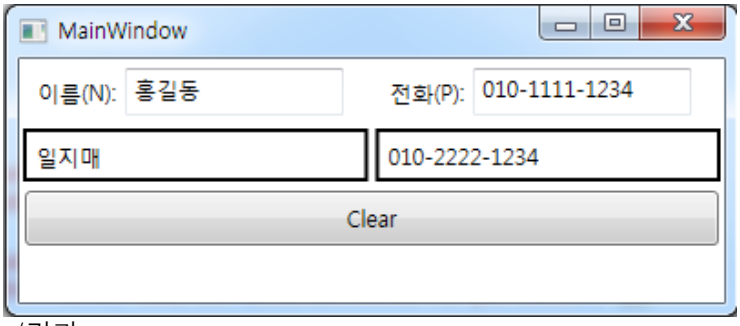
```
<Window x:Class="ex05_12.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525"
        xmlns:local="clr-namespace:ex05_12">
    <Window.Resources>
        <local:Person x:Key="person1" Name="홍길동" Phone = "010-1111-1234" />
        <local:Person x:Key="person2" Name="일지매" Phone = "010-2222-1234" />
    </Window.Resources>
    <StackPanel DataContext="{StaticResource person1}">
        <Grid >
```

```
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Text="{Binding Path=Name}" Width="120"/>
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Text="{Binding Path=Phone}" Width="120"/>
            </StackPanel>
        </Grid>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label    Height="25" Content="{Binding Path=Name, Source={StaticResource person2}}"/>
            </Border>
            <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label Height="25" Content="{Binding Path=Phone, Source={StaticResource person2}}"/>
            </Border>
        </Grid>
        <Grid >
            <Button Name="eraseButton" Margin="3" Height="30" Grid.Column="2" Content="Clear"
Click="eraseButton_Click" />
        </Grid>

    </StackPanel>
</Window>
```

</코드>

<결과>

[출력 결과]



</결과>

TextBox 는 리소스 person1을 사용하며 Label 은 리소스 person2를 사용한다. C# 코드의 변경은 없다.

다음과 같이 XAML 코드에서 리소스를 여러개 생성하고

```
    <Window.Resources>
        <local:Person x:Key="person1" Name="홍길동" Phone = "010-1111-1234" />
        <local:Person x:Key="person2" Name="일지매" Phone = "010-2222-1234" />
```

```
    </Window.Resources>
```

StackPanel 에 DataContext 는 다음과 같이 데이터 원본으로 person1을 설정하므로

```
<StackPanel DataContext="{StaticResource person1}">
```

TextBox 컨트롤은

```
        <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
            <Label >이름(_N):</Label>
            <TextBox Text="{Binding Path=Name}" Width="120"/>
        </StackPanel>
        <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
            <Label >전화(_P):</Label>
            <TextBox Text="{Binding Path=Phone}" Width="120"/>
        </StackPanel>
```

person1이 데이터 원본이 된다.

하지만 다음과 같이 Label 은 바인딩 문법을 사용하여 Source={StaticResource person2}를 설정했으므로 데이터 원본이 person2로 설정되어 바인딩 한다.

```
<Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
            <Label   Height="25" Content="{Binding Path=Name, Source={StaticResource person2}}"/>
        </Border>
        <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
            <Label Height="25" Content="{Binding Path=Phone, Source={StaticResource person2}}"/>
        </Border>
```

사실 Label 컨트롤은 데이터 원본으로 Person 객체(Name, Phone)를 직접 바인딩하지 않고 TextBox 의 객체를 원본으로 Text 속성과 바인딩할 수 있다.

다음은 UI 요소끼리의 바인딩한 예제다.

<코드>

[예제 05-13] UI 요소의 바인딩

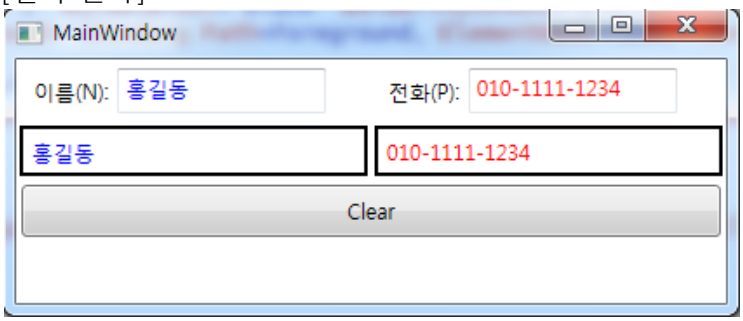
```
<StackPanel DataContext="{StaticResource person1}">
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Text="{Binding Path=Name}" Foreground="Blue" Width="120" x:Name="name"/>
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Text="{Binding Path=Phone}" Foreground="Red" Width="120" x:Name="phone"/>
            </StackPanel>
        </Grid>
        <Grid >
            <Grid.ColumnDefinitions>
```

```
                    <ColumnDefinition/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>
                <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
                    <Label    Height="25" Foreground="{Binding Path=Foreground, ElementName=name}"
Content="{Binding Path=Text, ElementName=name}"/>
                </Border>
                <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
                    <Label Height="25" Foreground="{Binding Path=Foreground, ElementName=phone}"
Content="{Binding Path=Text, ElementName=phone}"/>
                </Border>
            </Grid>
            <Grid >
                <Button Name="eraseButton" Margin="3" Height="30" Grid.Column="2" Content="Clear"
Click="eraseButton_Click" />
            </Grid>

        </StackPanel>
```

</코드>

<결과>

[출력 결과]



</결과>

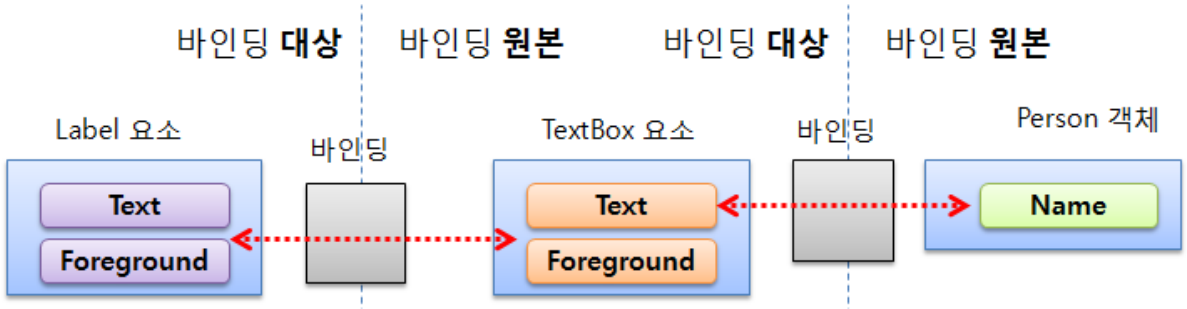
이름과 전화번호 각각의 Label 컨트롤은 TextBox 컨트롤의 Text 속성과 Foreground 속성을 바인딩한다.

다음 코드는

```
<Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
                    <Label    Height="25" Foreground="{Binding Path=Foreground, ElementName=name}"
Content="{Binding Path=Text, ElementName=name}"/>
                </Border>
                <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
                    <Label Height="25" Foreground="{Binding Path=Foreground, ElementName=phone}"
Content="{Binding Path=Text, ElementName=phone}"/>
                </Border>
```

바인딩 문법의 ElementName 을 사용하여 바인딩 원본 요소의 이름(name, phone)을 설정한다.

다음은 바인 순서를 그림으로 표현한 것이다.



WPF 바인딩 시스템의 핵심 기능 중 하나가 형식 변환이며 형식 변환기를 사용하여 서로 다란 속성끼리의 바인딩이 가능하도록 한다. 호환 가능한 형식 변환은 WPF 바인딩에서 자동으로 변환 형식 변환되지만 자동 형식 변환이 불가능한 형식은 직접 형식 변환기를 작성할 수 있다.

만약 전화번호 Person 객체에 성별을 구분하는 속성 Male 을 정의하고 남자면 true, 여자면 false 값을 갖도록 한다면 UI 에서 RadioButton 을 사용하여 남, 여를 구분할 수 있다. Person 객체의 속성 Male 을 보고 남, 여를 UI RadioButton 에 나타내므로 여기서 남자에 해당하는 RadioButton 이 true 라면 여자에 해당하는 RadioButton 은 false 가 되어야하며 반대로 여자에 해당하는 RadioButton 이 true 라면 남자에 해당하는 RadioButton 은 false 가 되어야한다. 여성을 표시하기 위한 UI 의 RadioButton 은 Person 속성의 Male 과 항상 역으로 매핑되어 나타내야 한다. 반대로 UI 의 여성에 해당하는 RadioButton 이 클릭되면 Person 속성의 Male 이 false 가 되어야한다.

다음 예제는 WPF 바인딩 시스템의 형식 변환기를 이용하는 예제다.

<코드>
[예제 05-14] 바인딩의 Type Converter

```
<Window x:Class="ex05_14.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"
        xmlns:local="clr-namespace:ex05_14">
    <Window.Resources>
        <local:Person x:Key="person" Name="이몽룡" Phone = "010-1111-1234" Male="True" />
        <!--<local:Person x:Key="person" Name="성춘향" Phone = "010-1111-1234" Male="false" />-->
        <local:MaleToFemaleConverter x:Key="maleConverter" />
    </Window.Resources>
    <StackPanel DataContext="{StaticResource person}">
        <Grid ClipToBounds="False">
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
```

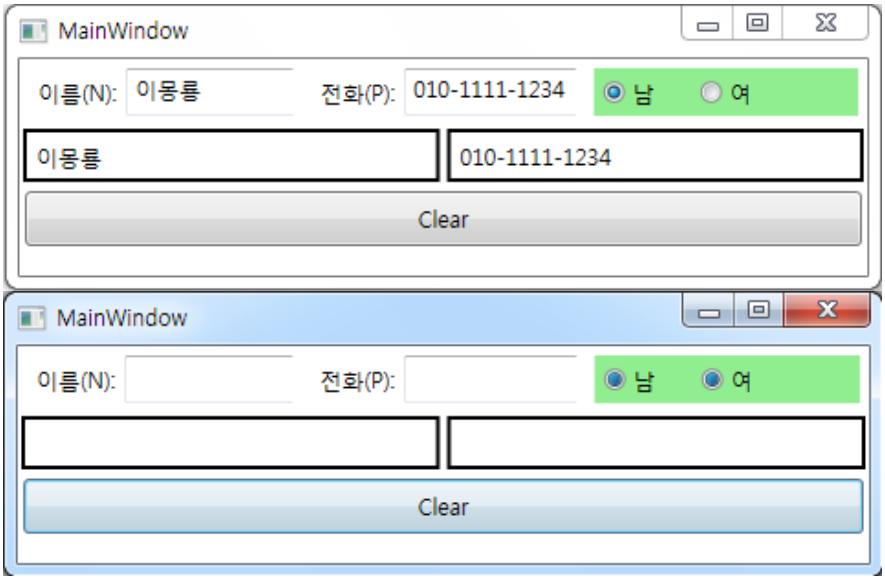
```
                <Label >이름(_N):</Label>
                <TextBox Text="{Binding Path=Name}" Width="120"/>
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Text="{Binding Path=Phone}" Width="120"/>
            </StackPanel>
            <StackPanel Grid.Column="2" Margin="5" Orientation="Horizontal" Background="LightGreen">
                <RadioButton IsChecked="{Binding Path=Male}" Content="남" Margin="5,5,20,5" />
                <RadioButton IsChecked="{Binding Path=Male, Converter={StaticResource maleConverter}}"
Content="여" Margin="5"/>
            </StackPanel>
        </Grid>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label    Height="25" Content="{Binding Path=Name}"/>
            </Border>
            <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label Height="25" Content="{Binding Path=Phone}"/>
            </Border>
        </Grid>
        <Grid >
            <Button Name="eraseButton" Margin="3" Height="30" Grid.Column="2" Content="Clear"
Click="eraseButton_Click" />
        </Grid>

    </StackPanel>
</Window>
```

```
    public partial class MainWindow : Window
    {
        Person per = null;
        public MainWindow()
        {
            InitializeComponent();
            per = (Person)FindResource("person");
        }

        private void eraseButton_Click(object sender, RoutedEventArgs e)
        {
            per.Name = "";
            per.Phone = "";
            per.Male = null;
        }
    }
    public class Person : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        private string name;
        public string Name
        {
            get { return name; }
```

```
            set
            {
                name = value;
                if (PropertyChanged != null)
                    PropertyChanged(this, new PropertyChangedEventArgs("Name"));
            }
        }
        private string phone;
        public string Phone
        {
            get { return phone; }
            set
            {
                phone = value;
                if (PropertyChanged != null)
                    PropertyChanged(this, new PropertyChangedEventArgs("Phone"));
            }
        }
        private bool? male;
        public bool? Male
        {
            get { return male; }
            set
            {
                male = value;
                if (PropertyChanged != null)
                    PropertyChanged(this, new PropertyChangedEventArgs("Male"));
            }
        }
}
[ValueConversion(/* 원본 형식 */ typeof(bool), /* 대상 형식 */ typeof(bool))]
public class MaleToFemaleConverter : IValueConverter
{
    // 데이터 속성을 UI 속성으로 변경할 때
    public object Convert(object value, Type targetType, object parameter,
                System.Globalization.CultureInfo culture)
    {
        if (targetType != typeof(bool?))
            return null;

        bool? male = (bool?) value;

        if (male == null)
            return null;
        else
            return !(bool?)value;
    }
    // UI 속성을 데이터 속성으로 변경할 때
    public object ConvertBack(object value, Type targetType, object
                parameter, System.Globalization.CultureInfo culture)
    {
        if (targetType != typeof(bool?))
            return null;

        bool? male = (bool?)value;

        if (male == null)
```

```
                return null;
            else
                return !(bool?)value;
        }
    }
```

</코드>
<결과>
[출력 결과]



</결과>

위 결과는 Male 속성이 남성일 때의 RadioButton UI 를 나타낸 것이며 아래 결과는 Clear 버튼을 클릭했을 때(Male 속성이 null)일 때의 결과다. 여성에 해당하는 RadioButton 은 Male 속성과 반대로 형식 변환을 통해 바인딩된다. Person 객체가 남, 여를 구분하지 않는다면 Male 속성은 null 로 어떤 Radio 버튼도 선택되지 않게 된다.

다음 코드는

```
<Window.Resources>
    <local:Person x:Key="person" Name="이몽룡" Phone = "010-1111-1234" Male="True" />
    <!--<local:Person x:Key="person" Name="성춘향" Phone = "010-1111-1234" Male="false" />-->
    <local:MaleToFemaleConverter x:Key="maleConverter" />
</Window.Resources>
```

Window 요소의 리소스로 person 키를 사용하는 Person 객체를 생성하고 Male 속성을 True 로 설정한다. 또 리소스에서 maleConverter 키를 사용하는 MaleToFemaleConverter 를 생성하도록 한다.

다음 코드는

```
<StackPanel Grid.Column="2" Margin="5" Orientation="Horizontal" Background="LightGreen">
    <RadioButton IsChecked="{Binding Path=Male}" Content="남" Margin="5,5,20,5" />
    <RadioButton IsChecked="{Binding Path=Male, Converter={StaticResource maleConverter}}"
```

```
Content="여" Margin="5"/>
            </StackPanel>
```

남, 여에 해당하는 두 RadioButton 을 정의하고 남자는 IsChecked 속성이 Person 객체의 Male 과 바인딩되고 여자는 IsChecked 속성이 Person 객체의 Male 과 리소스에 정의된 maleConverter 인 MaleToFemaleConverter 객체를 사용한 형식 변환으로 바인딩된다.

다음 코드는

```
private void eraseButton_Click(object sender, RoutedEventArgs e)
{
    per.Name = "";
    per.Phone = "";
    per.Male = null;
}
```

Clear 버튼이 클릭되면 Male 속성을 null 로 변경한다. UI 에서는 RadioButton 이 선택되지 않음으로 표시된다.

다음 코드는

```
public class Person : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    private string name;
    public string Name
    {
        get { return name; }
        set
        {
            name = value;
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs("Name"));
        }
    }
    private string phone;
    public string Phone
    {
        get { return phone; }
        set
        {
            phone = value;
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs("Phone"));
        }
    }
    private bool? male;
    public bool? Male
    {
        get { return male; }
        set
        {
            male = value;
            if (PropertyChanged != null)
```

```
                PropertyChanged(this, new PropertyChangedEventArgs("Male"));
            }
        }
    }
```

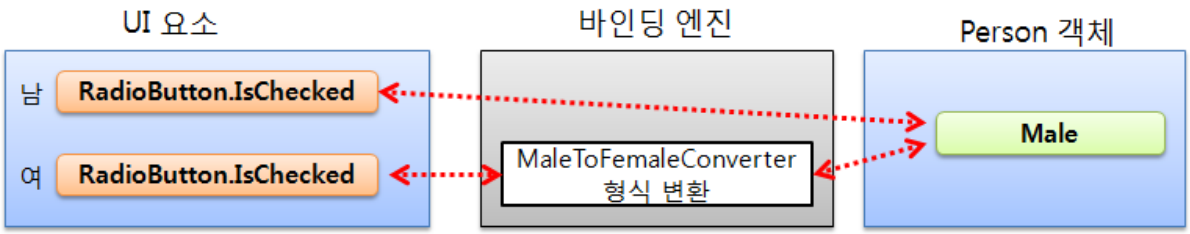
Person 클래스에 성별을 구분하기 위해 Male 속성을 정의하고 속성이 변경되면 PropertyChanged 이벤트를 발생시켜 속성의 변경을 UI 에 알린다.

다음 코드는

```
[ValueConversion(/* 원본 형식 */ typeof(bool), /* 대상 형식 */ typeof(bool))]
    public class MaleToFemaleConverter : IValueConverter
    {
        // 데이터 속성을 UI 속성으로 변경할 때
        public object Convert(object value, Type targetType, object parameter,
                System.Globalization.CultureInfo culture)
        {
            if (targetType != typeof(bool?))
                return null;

            bool? male = (bool?) value;

            if (male == null)
                return null;
            else
                return !(bool?)value;
        }
        // UI 속성을 데이터 속성으로 변경할 때
        public object ConvertBack(object value, Type targetType, object
                parameter, System.Globalization.CultureInfo culture)
        {
            if (targetType != typeof(bool?))
                return null;

            bool? male = (bool?)value;

            if (male == null)
                return null;
            else
                return !(bool?)value;
        }
    }
```

형식 변환의 핵심 코드로 여성에 해당하는 RadioButton 이 Person 의 Male 속성과 바인딩 하기 위해 IValueConverter 를 상속받은 클래스 MaleToFemaleConverter 를 정의하고 Converter() 메소드를 통해 데이터 속성(Male)을 UI 에 적용될 형식으로 변환하며 ConverterBack() 메소드를 통해 반대로 UI 에서 데이터 속성(Male)로 적용될 형식 변환을 정의한다.

다음은 코드를 그림으로 표현한 것이다.



WPF 의 바인딩은 동기화 기능과 형식 변환 기능 외에도 여러 가지 유용한 기능들을 제공하며 그 중 하나가 유효성 검사 규칙이다. 유효성 검사 규칙(validation rule)은 UI 에 입력되는 데이터가 데이터 원본에 갱신되기 전에 데이터 원본 형식에 맞는 데이터인지 유효성을 검사하는 객체다.

유효성 검사 코드는 ValidationRule 클래스를 상속받아 정의하며 Validate() 메소드를 오버라이드해 구현한다. 간단한 유효성 검사는 이미 내장되어 있는 유효성 검사 규칙인 ExceptionValidationRule 객체를 사용할 수 있다. ExceptionValidationRule 은 명시하지 않더라도 바인딩의 기본 유효성 검사 규칙으로 사용되며 직접 명시할 수도 있다.

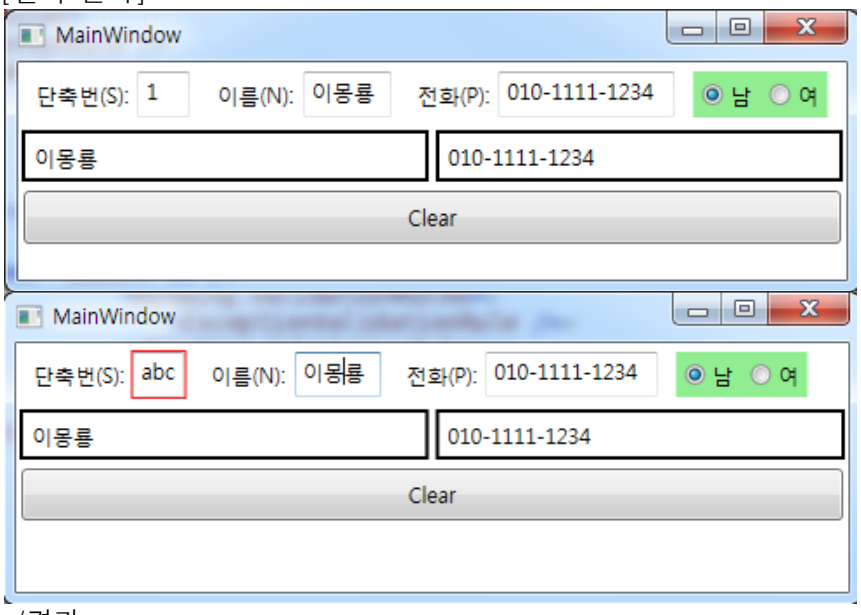
다음은 기본 유효성 검사 ExceptionValidationRule 의 사용을 확인하는 예제다.

<코드>
[예제 05-15] ExceptionValidattionRule 유효성 검사

```
    <Window.Resources>
        <local:Person x:Key="person" ShortNumber="1" Name="이몽룡" Phone = "010-1111-1234" Male="True"
/>
        <!--<local:Person x:Key="person" ShortNumber="2" Name="성춘향" Phone = "010-1111-1234"
Male="false" />-->
        <local:MaleToFemaleConverter x:Key="maleConverter" />
    </Window.Resources>
    <StackPanel DataContext="{StaticResource person}">
        <StackPanel Orientation="Horizontal">
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >단축번(_S):</Label>
                <!--<TextBox Text="{Binding Path=ShortNumber}" Width="30"/>-->
                <TextBox Width="30">
                    <TextBox.Text>
                        <Binding Path="ShortNumber">
                            <Binding.ValidationRules>
                                <ExceptionValidationRule />
                            </Binding.ValidationRules>
                        </Binding>
                    </TextBox.Text>
                </TextBox>
            </StackPanel>
        ...
    </StackPanel>
```

```
        private void eraseButton_Click(object sender, RoutedEventArgs e)
        {
            per.ShortNumber = null;
            per.Name = "";
            per.Phone = "";
            per.Male = null;
        }

    public class Person : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        private int? shortNumber;
        public int? ShortNumber
        {
            get { return shortNumber; }
            set
            {
                shortNumber = value;
                if (PropertyChanged != null)
                    PropertyChanged(this, new
                        PropertyChangedEventArgs("ShortNumber"));
            }
        }
        ...
    }
```

</코드>
<결과>
[출력 결과]



</결과>

위 결과는 데이터 원본에서 ShortNumber 결과를 바인딩하고 보여준 화면이며 아래 결과는 단축번호에 정수가 아닌 다른 값을 사용하면 빨강색으로 하이라이트되는 것을 보여주는 화면이다. ExceptionValidattionRule 유효성 검사 규칙은 바인딩될 때 기본으로 적용되는 유효성 검사 규칙이다.

다음 코드는

```
    <Window.Resources>
        <local:Person x:Key="person" ShortNumber="1" Name="이몽룡" Phone = "010-1111-1234" Male="True"
/>
        <!--<local:Person x:Key="person" ShortNumber="2" Name="성춘향" Phone = "010-1111-1234"
Male="false" />-->
        <local:MaleToFemaleConverter x:Key="maleConverter" />
    </Window.Resources>
```

Window 요소 리소스로 Person 객체를 생성하도록 정의하며 ShortNumber 를 '1' 로 설정한다.

다음 코드는

```
                <TextBox Width="30">
                    <TextBox.Text>
                        <Binding Path="ShortNumber">
                            <Binding.ValidationRules>
                                <ExceptionValidationRule />
                            </Binding.ValidationRules>
                        </Binding>
                    </TextBox.Text>
                </TextBox>
```

명시적으로 요효성 검사 규칙을 XAML 코드로 나타내기 위해 바인딩 마크업 확장문법을 사용하지 않고 Binding 요소를 직접 정의하여 바인딩 코드를 작성한 것이다. 속성의 형식에 따라 다음과 같이 코딩해도 기본 유효성 검사 규칙 ExceptionValidattionRule 이 적용되므로 꼭 위와 같이 입력하지 않아도 된다.

```
<TextBox Text="{Binding Path=ShortNumber}" Width="30"/>
```

다음 코드는

```
        private void eraseButton_Click(object sender, RoutedEventArgs e)
        {
            per.ShortNumber = null;
            per.Name = "";
            per.Phone = "";
            per.Male = null;
        }
```

Clear 버튼이 클릭되었을 때 ShortNumber 속성을 null 로 설정한다.

다음 코드는

```
        private int? shortNumber;
        public int? ShortNumber
        {
            get { return shortNumber; }
            set
            {
                shortNumber = value;
```

```
            if (PropertyChanged != null)
                PropertyChanged(this, new
                    PropertyChangedEventArgs("ShortNumber"));
        }
    }
```

단축 번호를 표현하기 위한 Person 클래스의 속성으로 ShortNumber 형식은 int?를 사용하여 null 설정이 가능하도록 하였다.

유효성 검사 규칙가 실패하여 하이라이트되는 것 외에도 잘못된 값에 대한 더 많은 정보를 얻고자 한다면 ValidationError 객체를 이용할 수 있다. 이 객체는 유효성 검사가 실패하면 생성되는 객체로 잘못된 값에 대한 자세한 정보를 표현한 객체다.

다음은 ValidationError 를 확인하여 메시지 박스로 출력하는 예제다.

<코드>

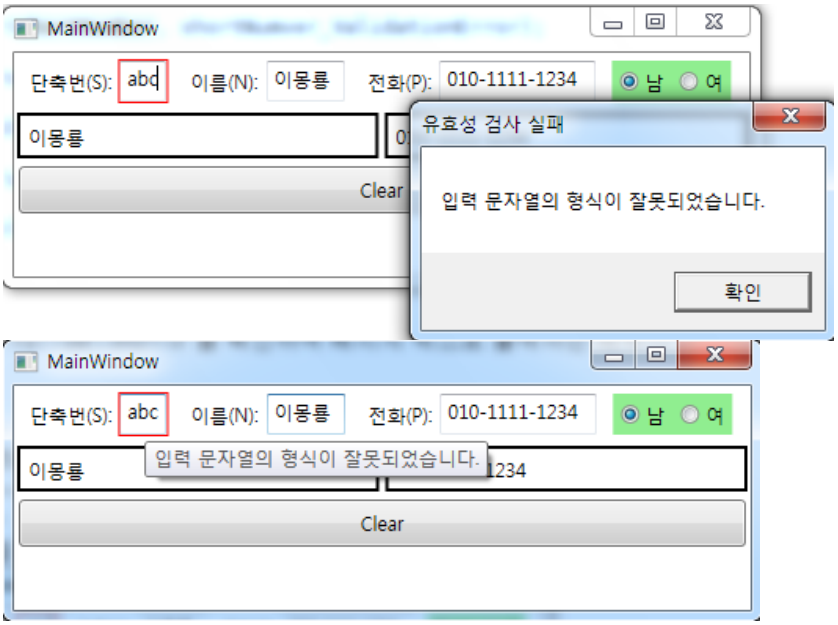
[예제 05-16] ValidatinError 확인하기

```
<TextBox Name="shortNumber"    Width="30">
    <TextBox.Text>
        <Binding Path="ShortNumber"
                 NotifyOnValidationError="True">
            <Binding.ValidationRules>
                <ExceptionValidationRule />
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>

public MainWindow()
{
    InitializeComponent();
    per = (Person)FindResource("person");

    Validation.AddErrorHandler(shortNumber, shortNumver_ValidationError);
}
void shortNumver_ValidationError(object sender,
        ValidationErrorEventArgs e)
{
    MessageBox.Show((string)e.Error.ErrorContent, "유효성 검사 실패");

    shortNumber.ToolTip = (string)e.Error.ErrorContent;
}
```

</코드>

<결과>

[출력 결과]



</결과>

결과는 ValidationError 를 메시지 박스와 ToolTip 으로 확인한 화면이다. 유효성 검사 에러 메시지를 확인학 위해서는 유효성 검사가 실패하면 발생되는 이벤트 핸들러를 다음과 같이 등록하고

```
Validation.AddErrorHandler(shortNumber, shortNumver_ValidationError);
```

핸들러에서

```
        void shortNumver_ValidationError(object sender,
                ValidationErrorEventArgs e)
        {
            MessageBox.Show((string)e.Error.ErrorContent, "유효성 검사 실패");

            shortNumber.ToolTip = (string)e.Error.ErrorContent;
        }
```

두 번째 인수의 e.Error 가 ValidationError 객체로 여러 가지 정보를 확인할 수 있으며 간단한 메시지는 ErrorContent 로 확인할 수 있다.

주의할 점은 유효성 검사가 실패하여 이벤트가 발생하도록 하려면 다음과 같이

```
<Binding Path="ShortNumber"   NotifyOnValidationError="True">
```

NotifyOnValidationError 속성을 True 로 설정해야 한다.

기본 유효성 검사 외에도 사용자가 직접 유효성 검사 로직을 작성할 수 있으며 사용자 유효성 검사 객체는 ValidationRule 클래스를 상속받아 Validate() 메소드를 오버라이드하여 구현한다.

다음은 사용자 정의 유효성 검사 예제 코드다.

<코드>

[예제 05-17] 사용자 정의 유효성 검사

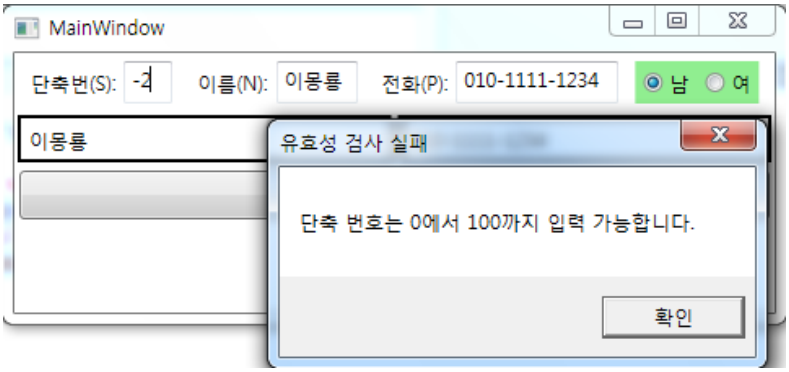
```
        <TextBox Name="shortNumber"   Width="30">
                <TextBox.Text>
```

```
                        <Binding Path="ShortNumber"
                                 NotifyOnValidationError="True">
                            <Binding.ValidationRules>
                                <ExceptionValidationRule />
                            </Binding.ValidationRules>
                        </Binding>
                    </TextBox.Text>
                </TextBox>

    public class ShortNumberValidationRule : ValidationRule
    {
        int min;
        public int Min
        {
            get { return min; }
            set { min = value; }
        }

        int max;
        public int Max
        {
            get { return max; }
            set { max = value; }
        }

        public override ValidationResult Validate(object value,
                System.Globalization.CultureInfo cultureInfo)
        {
            int number;
            if (!int.TryParse((string)value, out number))
            {
                return new ValidationResult(false, "정수를  입력하세요.");
            }

            if (min <= number && number <= max)
            {
                // new ValidationResult(true, null)  같다
                return ValidationResult.ValidResult;
            }
            else
            {
                string msg =
                 string.Format("단축  번호는  {0}에서  {1}까지  입력  가능합니다.", min, max);
                return new ValidationResult(false, msg);
            }
        }
    }
```

</코드>

<결과>

[출력 결과]



</결과>

결과는 사용자 정의 유효성 검사 객체 ShortNumberValidationRule 를 사용하여 출력한 결과다. 사용자 저의 유효성 검사는 다음과 같이 ValidationRule 클래스를 상속받아 Validate() 메소드를 오버라이드 한다.

```
public override ValidationResult Validate(object value,
            System.Globalization.CultureInfo cultureInfo)
{
    int number;
    if (!int.TryParse((string)value, out number))
    {
        return new ValidationResult(false, "정수를 입력하세요.");
    }

    if (min <= number && number <= max)
    {
        // new ValidationResult(true, null) 같다
        return ValidationResult.ValidResult;
    }
    else
    {
        string msg =
         string.Format("단축 번호는 {0}에서 {1}까지 입력 가능합니다.", min, max);
        return new ValidationResult(false, msg);
    }

}
```

value 변수가 유효성 검사에 사용되는 UI 입력된 데이터 값이며 value 를 검사하여 정수인지 확인하고 단축 번호의 최대, 최소 값의 범위에 내에 값인지 검사한다. Validate() 오버라이드 메소드는 ValidationResult(true, null) 객체와 ValidationResult(false, null) 객체를 반환하여 유효성 검사가 성공했는지 실패했는지를 구분한다. 두 번째 인수는 유효성 검사가 실패하면 표시될 메시지 문자열이다.

WPF 의 바인딩 시스템은 리스트 형태의 데이터를 UI 와 바인딩하기 위한 다양한 기능을 제공한다. 컬렉션 객체의 현재 아이템(원소)을 개념을 제공하며 컨렉션 객체의 정렬이나 필터링, 그룹화 등의 강력한 기능을 어렵지 않게 사용할 수 있도록 제공한다.

다음은 Person 객체를 관리하는 People 컬렉션 객체를 ListBox 컨트롤에 바인딩하는 예제다. People 컬렉션은 List<Person>을 상속받아 생성하며 기본 원소로 세 개의 Person 객체를 가진다.

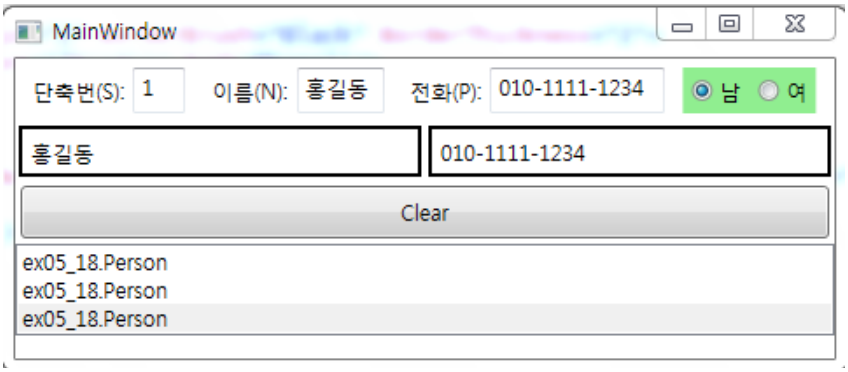
<코드>
[예제 05-18] ListBox 컨트롤 바인딩

```
<StackPanel Name="panel">
        <StackPanel Orientation="Horizontal">
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >단축번(_S):</Label>
                <TextBox Name="shortNumber"    Width="30">
                    <TextBox.Text>
                        <Binding Path="ShortNumber" NotifyOnValidationError="True">
                            <Binding.ValidationRules>
                                <ExceptionValidationRule />
                            </Binding.ValidationRules>
                        </Binding>
                    </TextBox.Text>
                </TextBox>
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
                <Label >이름(_N):</Label>
                <TextBox Text="{Binding Path=Name}" Width="50"/>
            </StackPanel>
            <StackPanel Grid.Column="2" Margin="5" Orientation="Horizontal">
                <Label >전화(_P):</Label>
                <TextBox Text="{Binding Path=Phone}" Width="100"/>
            </StackPanel>
            <StackPanel Grid.Column="3" Margin="5" Orientation="Horizontal" Background="LightGreen">
                <RadioButton IsChecked="{Binding Path=Male}" Content="남" Margin="5" />
                <RadioButton IsChecked="{Binding Path=Male, Converter={StaticResource maleConverter}}"
Content="여" Margin="5"/>
            </StackPanel>
        </StackPanel>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Border Grid.Column="0" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label    Height="25" Content="{Binding Path=Name}"/>
            </Border>
            <Border Grid.Column="1" Margin="2" BorderBrush="Black" BorderThickness="2">
                <Label Height="25" Content="{Binding Path=Phone}"/>
            </Border>
        </Grid>
        <Grid >
            <Button Name="eraseButton" Margin="3" Height="30" Grid.Column="2" Content="Clear"
Click="eraseButton_Click" />
        </Grid>
        <ListBox ItemsSource="{Binding}"/>
    </StackPanel>

    public partial class MainWindow : Window
    {
```

```
        People people = new People();
        public MainWindow()
        {
            InitializeComponent();
            Validation.AddErrorHandler(shortNumber, shortNumver_ValidationError);

            panel.DataContext = people;
        }
        void shortNumver_ValidationError(object sender,
                ValidationErrorEventArgs e)
        {
            MessageBox.Show((string)e.Error.ErrorContent, "유효성 검사 실패");

            shortNumber.ToolTip = (string)e.Error.ErrorContent;
        }
        private void eraseButton_Click(object sender, RoutedEventArgs e)
        {
        }
}
public class Person : INotifyPropertyChanged
{
        public event PropertyChangedEventHandler PropertyChanged;

        private int? shortNumber;
        public int? ShortNumber
        {
            get { return shortNumber; }
            set
            {
                shortNumber = value;
                if (PropertyChanged != null)
                    PropertyChanged(this,
                        new PropertyChangedEventArgs("ShortNumber"));
            }
        }
        private string name;
        public string Name
        {
            get { return name; }
            set
            {
                name = value;
                if (PropertyChanged != null)
                    PropertyChanged(this, new PropertyChangedEventArgs("Name"));
            }
        }
        private string phone;
        public string Phone
        {
            get { return phone; }
            set
            {
                phone = value;
                if (PropertyChanged != null)
                    PropertyChanged(this, new PropertyChangedEventArgs("Phone"));
            }
        }
        private bool? male;
```

```
        public bool? Male
        {
            get { return male; }
            set
            {
                male = value;
                if (PropertyChanged != null)
                    PropertyChanged(this, new PropertyChangedEventArgs("Male"));
            }
        }
    }
    public class People : List<Person>
    {
        public People()
        {
            Add(new Person() { ShortNumber = 1, Name = "홍길동",
                Phone = "010-1111-1234", Male = true });
            Add(new Person() { ShortNumber = 2, Name = "이몽룡",
                Phone = "010-2222-1234", Male = true });
            Add(new Person() { ShortNumber = 3, Name = "성춘향",
                Phone = "010-1111-1234", Male = false });
        }
    }
```

</코드>
<결과>
[출력 결과]



</결과>

출력 결과는 ListBox 컨트롤이 컬렉션 객체를 원본 객체로 사용했을 때의 결과로 컬렉션 원소 하나하나를 ListBox 컨트롤의 아이템으로 바인딩하며 각 아이템을 어떻게 보여야 할지 명사하지 않으면 컬렉션 원소들의 ToString() 메소들을 호출하여 문자열과 ListBox 컨트롤의 아이템으로 출력한다. 단축번호나 이름, 전화, 성별은 첫 번째 컬렉션의 원소와 바인딩 된 내용을 볼 수 있다. 이것은 현재 아이템이라는 개념으로 사용되며 현재 아이템을 변경하지 않는다면 첫 번째 컬렉션의 원소가 현재 아이템이된다. 물론 얼마 든지 현재 아이템을 변경할 수 있다.

다음 코드는

```
<ListBox ItemsSource="{Binding}"/>
```

ListBox 컨트롤의 ItemsSource 속성에 바인딩 문법을 사용했으며 데이터 원본을 명시하지 않았으므로 부모 요소의 DataContext 가 데이터 원본이다.

부모 요소는 다음과 같이

```
<StackPanel Name="panel">
```

이로 명명하고 C# 코드에서 다음과 같이 DataContext 를 설정한다.

```
panel.DataContext = people;
```

다음 코드는

```
public class People : List<Person>
{
    public People()
    {
        Add(new Person() { ShortNumber = 1, Name = "홍길동",
            Phone = "010-1111-1234", Male = true });
        Add(new Person() { ShortNumber = 2, Name = "이몽룡",
            Phone = "010-2222-1234", Male = true });
        Add(new Person() { ShortNumber = 3, Name = "성춘향",
            Phone = "010-1111-1234", Male = false });
    }
}
```

List<Person>을 상속받아 정의한 People 클래스로 생성자에서 기본으로 세 개의 Person 를 생성한다.

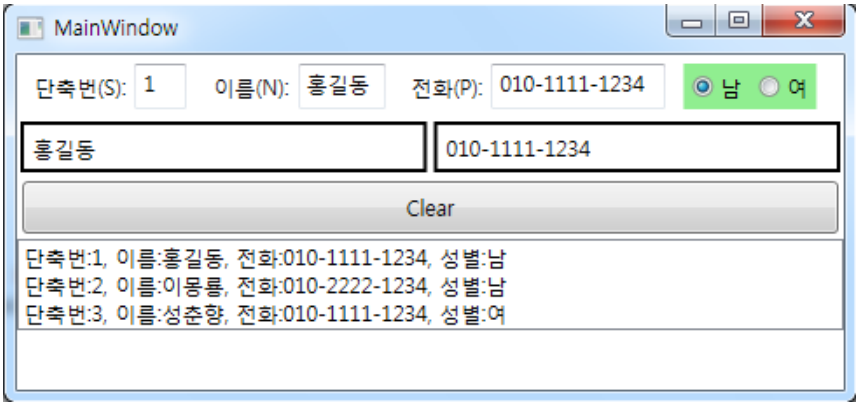
ListBox 컨트롤의 각 아이템은 데이터 원본으로 선택된 People 컬렉션 객체는 각 원소의 ToString() 메소드를 호출하여 아이템을 출력하므로 다음과 같이 ToString()을 오버라이드하여 출력할 수 있다.

<코드>

[예제 05-19] Person 객체의 ToString() 메소드 오버라이드

```
public class Person : INotifyPropertyChanged
{
    ...
    public override string ToString()
    {
        string sex ;
        if (Male == null)
            sex = "남/여";
        else if (Male == true)
            sex = "남";
        else
            sex = "여";

        return string.Format("단축번:{0}, 이름:{1}, 전화:{2}, 성별:{3}",
            ShortNumber, Name, Phone, sex);
```

```
        }
    }
```
</코드>
<결과>
[출력 결과]



</결과>

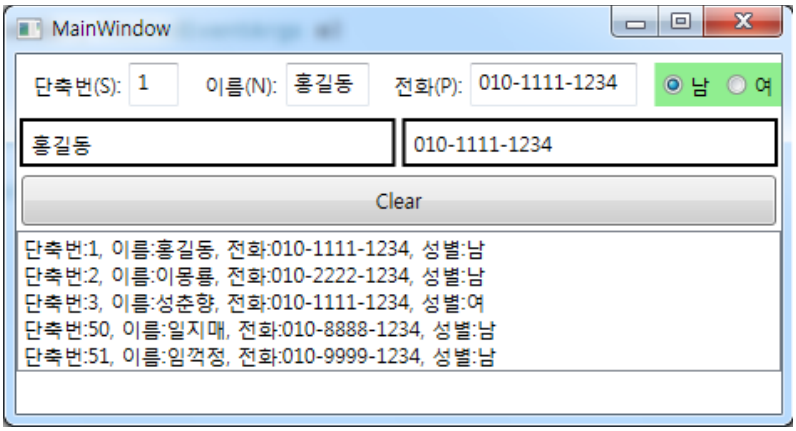
Person 의 ToString() 메소드를 오버라이드하면 ListBox 컨트롤의 아이템에 People 컬렉션 객체의 Person 객체 원소 하나하나의 ToString() 메소드가 반환하는 문자열이 출력된다.

컬렉션 객체도 XAML 코드에서 리소스로 정의할 수 있으며 컬렉션 원소의 생성도 정의할 수 있다. 다음은 Window 요소의 리소스에서 컬렉션을 생성 정의하고 바인딩한 예제다.

<코드>
[예제 05-20] 컬렉션 객체를 리소스에서 정의 생성하고 바인딩 하기

```
<Window.Resources>
        <local:People x:Key="people">
            <local:Person ShortNumber = "50" Name = "일지매"
                Phone = "010-8888-1234" Male = "True" />
            <local:Person ShortNumber = "51" Name = "임꺽정"
                Phone = "010-9999-1234" Male = "True" />
        </local:People>

        <local:MaleToFemaleConverter x:Key="maleConverter" />
    </Window.Resources>

    <StackPanel DataContext="{StaticResource people}">

    public class People : List<Person>
    {
        public People()
        {
            Add(new Person() { ShortNumber = 1, Name = "홍길동",
                Phone = "010-1111-1234", Male = true });
            Add(new Person() { ShortNumber = 2, Name = "이몽룡",
                Phone = "010-2222-1234", Male = true });
            Add(new Person() { ShortNumber = 3, Name = "성춘향",
                Phone = "010-1111-1234", Male = false });
        }
```

```
    }
</코드>
```
<결과>
[출력 결과]



</결과>

People 객체를 생성하면 생성자에서 기본으로 세 person 객체(홍길동, 이몽룡, 성춘향)를 원소로 가지므로 XAML 에서 원소를 생성(일지매, 임꺽정)하면 컬렉션에 추가된다.

다음 코드는

```
<Window.Resources>
        <local:People x:Key="people">
            <local:Person ShortNumber = "50" Name = "일지매"
                Phone = "010-8888-1234" Male = "True" />
            <local:Person ShortNumber = "51" Name = "임꺽정"
                Phone = "010-9999-1234" Male = "True" />
        </local:People>

        <local:MaleToFemaleConverter x:Key="maleConverter" />
    </Window.Resources>
```

XAML 코드에서 Window 요소의 리소스로 People 컬렉션 객체 생성을 정의하고 people 키 값을 사용하며 원소로 Person 객체 두 개를 추가한다.

리스트 형태의 UI 가 아닌 TextBox 나 Label, RadioBox 컨트롤 등은 한 번에 한 개의 객체 속성과 바인딩되므로 현재 아이템(current item)을 변경하면 단축번호, 이름, 전화, 성별를 바꿀 수 있으며 바인딩 시스템을 사용해 ListBox 컨트롤에서 선택된 내용을 현재 아이템으로 설정할 수 있다.

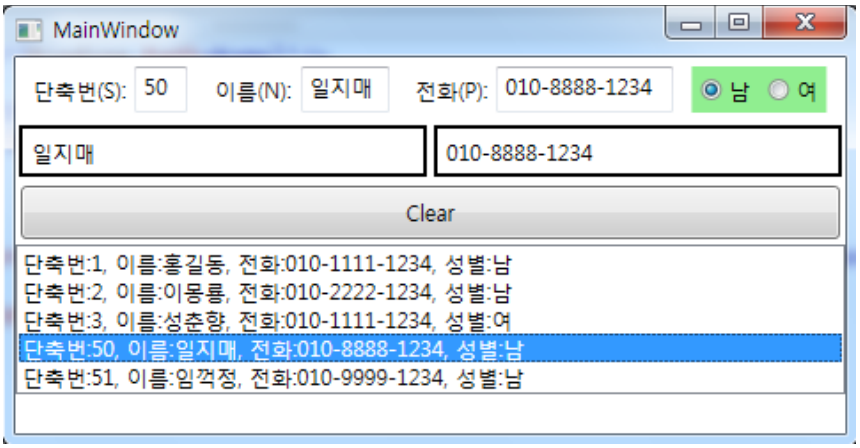
다음은 ListBox 컨트롤의 IsSynchronizedWithCurrentItem="True" 속성을 상용하여 현재 아이템이 단일 바인딩 UI 와 바인딩 되도록한 예제다.

<코드>
[예제 05-21] ListBox 컨트롤의 현재 아이템

```
<ListBox ItemsSource="{Binding}" IsSynchronizedWithCurrentItem="True"/>
</코드>
```

<결과>

[출력 결과]



</결과>

단지 ListBox 컨트롤의 IsSynchronizedWithCurrentItem 속성을 True 로 설정한 것 뿐이지만 ListBox 컨트롤의 아이템을 클릭하면 클릭된 아이템이 현재 아이템으로 설정되는 것을 볼 수 있다. WPF 바인딩 엔진의 강력한 기능이다.

우리가 ListBox 컨트롤의 아이템을 클릭했을 때 현재 아이템 개념을 사용할 수 있다고 하더라도 일반 컬렉션(List<>)은 현재 아이템(current item) 개념이 없으므로 바인딩 시스템에서 현재 아이템 개념을 사용할 수 있도록 하려면 우리는 컬렉션 뷰(collection view)라고 부르는 UI 컨트롤과 일반 컬렉션 사이의 인터페이스를 수행하는 뷰를 얻어 사용해야 한다. 예로 이전 버튼과 다음 버튼 두 개를 만들어 버튼이 눌릴때 마다 현재 아이템을 변경하고자 한다면 C# 코드에서 직접 현재 아이템 개념을 처리해야 하고 일반 컬렉션은 현재 아이템 개념이 없으므로 컬렉션 뷰를 얻어 사용해야 한다.

다음은 현재 아이템 개념을 사용하기 위해 일반 컬렉션(List<>)에 컬렉션 뷰를 사용한 예제다.

<코드>

[예제 05-22] 컬렉션 뷰 사용하기

```
<StackPanel Grid.Column="2" Margin="5" Orientation="Horizontal">
    <Label >전화(_P):</Label>
    <TextBox Text="{Binding Path=Phone}" Width="100"/>
</StackPanel>
<StackPanel Grid.Column="3" Margin="5,5,0,5"
     Orientation="Horizontal" Background="LightGreen">
    <RadioButton IsChecked="{Binding Path=Male}" Content="남"
                 Margin="5" />
</StackPanel>
<StackPanel Grid.Column="3" Margin="0,5,5,5"
     Orientation="Horizontal" Background="LightGreen">
```

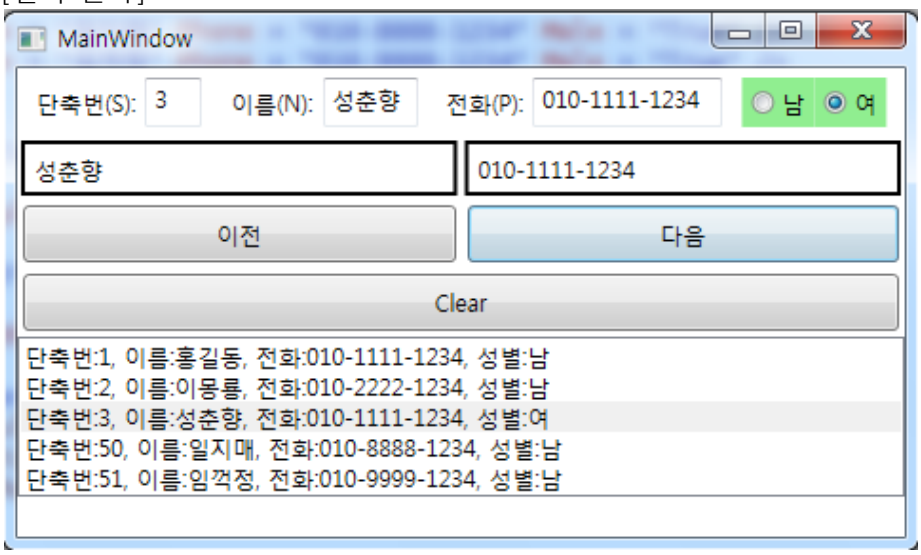
```
                <RadioButton IsChecked="{Binding Path=Male,
        Converter={StaticResource maleConverter}}" Content="여" Margin="5"/>
            </StackPanel>
        </StackPanel>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Border Grid.Column="0" Margin="2" BorderBrush="Black"
                    BorderThickness="2">
                <Label    Height="25" Content="{Binding Path=Name}"/>
            </Border>
            <Border Grid.Column="1" Margin="2" BorderBrush="Black"
                    BorderThickness="2">
                <Label Height="25" Content="{Binding Path=Phone}"/>
            </Border>
        </Grid>
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Button Name="prev" Grid.Column="0" Margin="3" Height="30"
                    Content="이전" Click="prev_Click" />
            <Button Name="next" Grid.Column="1" Margin="3" Height="30"
                    Content="다음" Click="next_Click" />
        </Grid>

public partial class MainWindow : Window
{
    People people = new People();
    public MainWindow()
    {
        InitializeComponent();
        Validation.AddErrorHandler(shortNumber, shortNumver_ValidationError);
    }
    void shortNumver_ValidationError(object sender,
                ValidationErrorEventArgs e)
    {
        MessageBox.Show((string)e.Error.ErrorContent, "유효성  검사  실패");

        shortNumber.ToolTip = (string)e.Error.ErrorContent;
    }
    private void eraseButton_Click(object sender, RoutedEventArgs e)
    {
        ICollectionView view =
                CollectionViewSource.GetDefaultView(FindResource("people"));
        Person person = (Person)view.CurrentItem;
        person.ShortNumber = null;
        person.Name = "";
        person.Phone = "";
        person.Male = null;
    }

    private void prev_Click(object sender, RoutedEventArgs e)
    {
```

```
            ICollectionView view =
                  CollectionViewSource.GetDefaultView(FindResource("people"));
            view.MoveCurrentToPrevious();
            if (view.IsCurrentBeforeFirst)
            {
                  view.MoveCurrentToFirst();
            }
      }

      private void next_Click(object sender, RoutedEventArgs e)
      {
            ICollectionView view =
                  CollectionViewSource.GetDefaultView(FindResource("people"));
            view.MoveCurrentToNext();
            if (view.IsCurrentAfterLast)
            {
                  view.MoveCurrentToLast();
            }
      }
}
```

</코드>

<결과>

[출력 결과]



</결과>

이전 버튼이나 다음 버튼을 누르면 모든 UI 의 내용이 현재 아이템으로 변경되는 것을 확인할 수 있다.

다음 코드는

```
<StackPanel Grid.Column="3" Margin="5,5,0,5"
      Orientation="Horizontal" Background="LightGreen">
      <RadioButton IsChecked="{Binding Path=Male}" Content="남"
            Margin="5" />
</StackPanel>
<StackPanel Grid.Column="3" Margin="0,5,5,5"
      Orientation="Horizontal" Background="LightGreen">
      <RadioButton IsChecked="{Binding Path=Male,
Converter={StaticResource maleConverter}}" Content="여" Margin="5"/>
```

```
            </StackPanel>
```

남, 여를 구분하는 RadioButton 을 부모 요소를 다르게 설정하여 같은 그룹이 아닌 별도의 그룹으로 설정하도록 한다. 그룹이 같아 두 요소 서로 간섭하는 것을 제가하기 위해서다.

다음 코드는

```
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Button Name="prev" Grid.Column="0" Margin="3" Height="30"
                    Content="이전" Click="prev_Click" />
            <Button Name="next" Grid.Column="1" Margin="3" Height="30"
                    Content="다음" Click="next_Click" />
        </Grid>
```

이전 버튼이나 이후 버튼을 클릭하면 각각의 이벤트가 발생되도록 핸들러를 등록하다.

다음 코드는

```
    private void eraseButton_Click(object sender, RoutedEventArgs e)
        {
            ICollectionView view =
                    CollectionViewSource.GetDefaultView(FindResource("people"));
            Person person = (Person)view.CurrentItem;
            person.ShortNumber = null;
            person.Name = "";
            person.Phone = "";
            person.Male = null;
        }
```

Clear 버튼이 클릭되면 Window 리소스의 people 컬렉션 객체를 얻어 뷰 컬렉션의 인터페이스 ICollectionView 를 얻고 CurrentItem 에 설정된 현재 아이템의 내용을 Clear 하는 코드다. 컬렉션 뷰는 CollectionViewSource.GetDefaultView() 메소드를 사용하여 얻고 이 메소드가 반환하는 인터페이스의 ICollectionView.CurrentItem 속성에서 현재 아이템을 얻는다. Clear 버튼으로 현재 아이템 원소를 Clear 하면 ListBox 에는 갱신되지 않는데 이것은 단지 문자열 형태로 ListBox 아이템이 출력되었기 때문이고 다시 재바인딩을 통해 해결할 수는 있지만 뒤에서 다루는 방법으로 더 쉽게 해결되므로 지금은 그냥 진행하도록 한다.

다음 코드는

```
        private void prev_Click(object sender, RoutedEventArgs e)
        {
            ICollectionView view =
                    CollectionViewSource.GetDefaultView(FindResource("people"));
            view.MoveCurrentToPrevious();
            if (view.IsCurrentBeforeFirst)
            {
                view.MoveCurrentToFirst();
            }
```

```
        }
```

이전 코드가 눌렸을 때 컬렉션 뷰 인터페이스를 얻고 MoveCurrentToPrvious() 메소드로 이전 아이템으로 현재 아이템을 이동한다. 현재 아이템이 더 이상 이전 원소로 이동할 수 없을 때는 IsCurrentBeforeFirst 속성이 true 이므로 이때는 MoveCurrentToFirst() 메소드를 사용하여 현재 아이템을 첫 원소에 위치 시킨다.

다음 코드는

```
        private void next_Click(object sender, RoutedEventArgs e)
        {
            ICollectionView view =
                CollectionViewSource.GetDefaultView(FindResource("people"));
            view.MoveCurrentToNext();
            if (view.IsCurrentAfterLast)
            {
                view.MoveCurrentToLast();
            }
        }
```

이전 버튼과 반대로 MoveCurrentToNext() 메소드로 다음 원소로 현재 아이템의 위치를 이동시키고 더 이상 이동할 수 없다면 IsCurrentAfterLast 가 true 이므로 MoveCurrentToLast() 메소드로 마지막 원소에 위치하도록 한다.

ListBox 컨트롤의 아이템을 출력할 때 ToString()을 사용하면 문자열 외에 다양하고 강력한 기능을 사용하지 못한다. 일반적으로 두 개 이상의 속성들을 보여주거나 문자열이 아닌 더 다양한 형태의 리스트 아이템을 표현하고자 한다면 데이터 템플릿(Data Template)을 이용한다. 테이터 템플릿은 표현될 수 있는 요소들의 집합이다.

다음은 데이터 템플릿을 이용한 ListBox 컨트롤의 아이템을 출력한 예제다.

<코드>

[예제 05-23] 데이터 템플릿 ListBox 컨트롤

```
<StackPanel DataContext="{StaticResource people}">
        <StackPanel Orientation="Horizontal">
            <StackPanel Grid.Column="0" Margin="5" Orientation="Horizontal">
                <Label >단축번(_S):</Label>
                <TextBox Name="shortNumber"    Width="30">
                    <TextBox.Text>
                        <Binding Path="ShortNumber"
                                 NotifyOnValidationError="True">
                            <Binding.ValidationRules>
                                <ExceptionValidationRule />
                            </Binding.ValidationRules>
                        </Binding>
                    </TextBox.Text>
                </TextBox>
            </StackPanel>
            <StackPanel Grid.Column="1" Margin="5" Orientation="Horizontal">
```

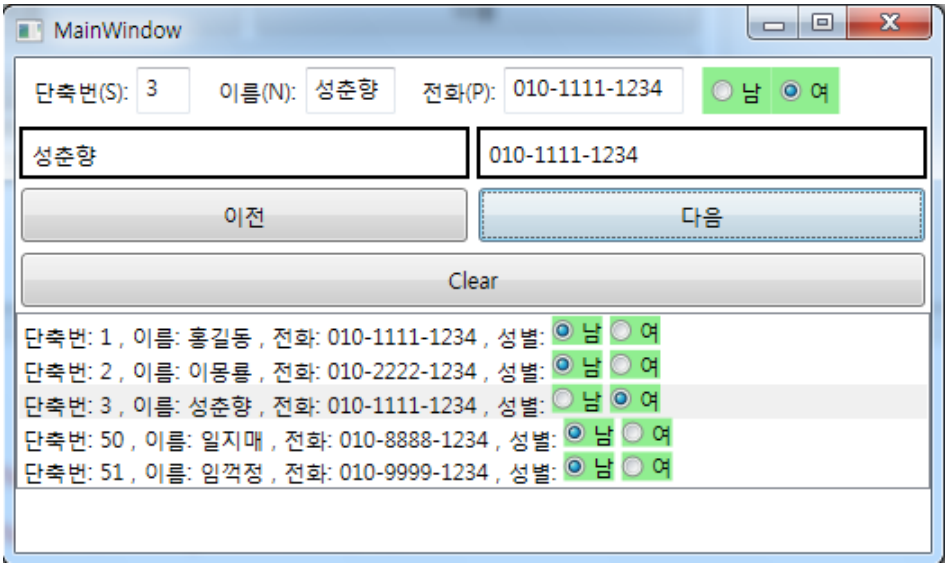
```
            <Label >이름(_N):</Label>
            <TextBox Text="{Binding Path=Name}" Width="50"/>
        </StackPanel>
        <StackPanel Grid.Column="2" Margin="5" Orientation="Horizontal">
            <Label >전화(_P):</Label>
            <TextBox Text="{Binding Path=Phone}" Width="100"/>
        </StackPanel>
        <StackPanel Grid.Column="3" Margin="5,5,0,5"
                Orientation="Horizontal" Background="LightGreen">
            <RadioButton IsChecked="{Binding Path=Male}" Content="남"
                Margin="5" />
        </StackPanel>
        <StackPanel Grid.Column="3" Margin="0,5,5,5"
                Orientation="Horizontal" Background="LightGreen">
            <RadioButton IsChecked="{Binding Path=Male,
  Converter={StaticResource maleConverter}}" Content="여" Margin="5"/>
        </StackPanel>
</StackPanel>
<Grid >
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Border Grid.Column="0" Margin="2" BorderBrush="Black"
                BorderThickness="2">
            <Label    Height="25" Content="{Binding Path=Name}"/>
        </Border>
        <Border Grid.Column="1" Margin="2" BorderBrush="Black"
                BorderThickness="2">
            <Label Height="25" Content="{Binding Path=Phone}"/>
        </Border>
</Grid>
<Grid >
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Button Name="prev" Grid.Column="0" Margin="3" Height="30"
                Content="이전" Click="prev_Click" />
        <Button Name="next" Grid.Column="1" Margin="3" Height="30"
                Content="다음" Click="next_Click" />
</Grid>
<Grid >
        <Button Name="eraseButton" Margin="3" Height="30" Grid.Column="2"
                Content="Clear" Click="eraseButton_Click" />
</Grid>
<ListBox ItemsSource="{Binding}" IsSynchronizedWithCurrentItem="True">
        <ListBox.ItemTemplate>
            <DataTemplate>
                <TextBlock>
                    단축번:
                    <TextBlock Text="{Binding Path=ShortNumber}" />
                    , 이름:
                    <TextBlock Text="{Binding Path=Name}" />
                    , 전화:
```

```
                    <TextBlock Text="{Binding Path=Phone}" />
                    , 성별:
                    <StackPanel Grid.Column="3" Orientation="Horizontal"
                                Background="LightGreen">
                        <RadioButton IsChecked="{Binding Path=Male}"
                                     Content="남" />
                    </StackPanel>
                    <StackPanel Grid.Column="3" Orientation="Horizontal"
                                Background="LightGreen">
                        <RadioButton IsChecked="{Binding Path=Male,
                      Converter={StaticResource maleConverter}}" Content="여" />
                    </StackPanel>
                </TextBlock>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
</StackPanel>
```

</코드>

<결과>

[출력 결과]



</결과>

데이터 템플릿을 사용하면 ListBox 컨트롤의 아이템으로 표현될 수 있는 모든 요소를 사용할 수 있으므로 더 다양하고 강력한 기능을 구현할 수 있다.

다음 코드는

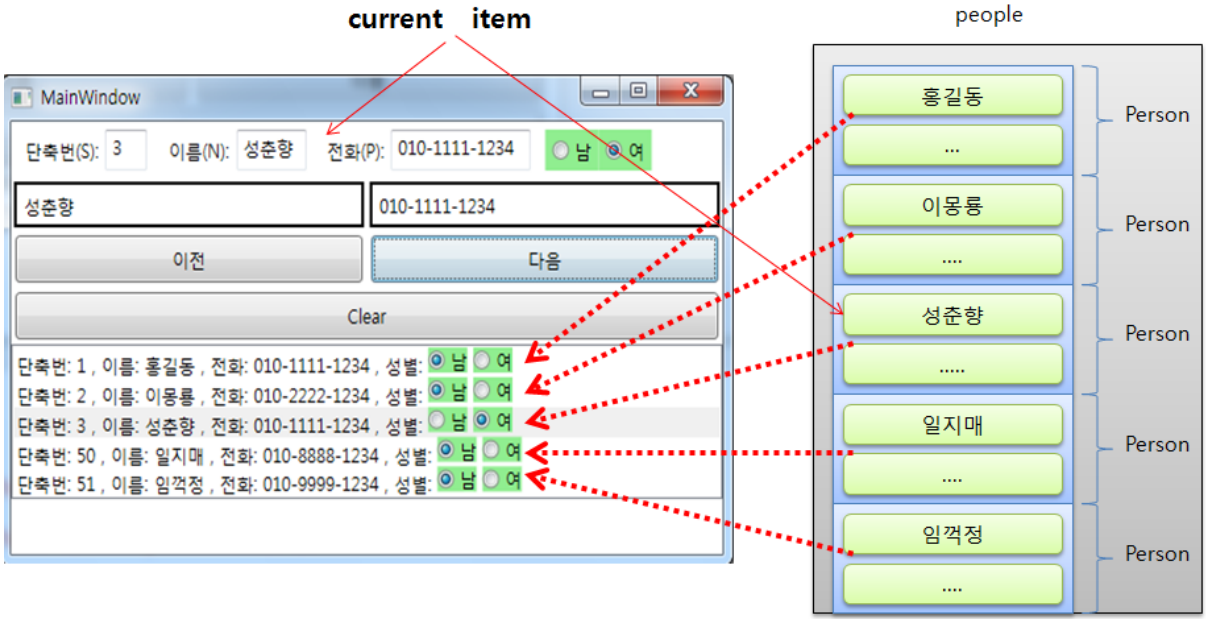
```
<ListBox ItemsSource="{Binding}" IsSynchronizedWithCurrentItem="True">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <TextBlock>
                단축번:
                <TextBlock Text="{Binding Path=ShortNumber}" />
                , 이름:
                <TextBlock Text="{Binding Path=Name}" />
                , 전화:
```

```
                    <TextBlock Text="{Binding Path=Phone}" />
                    , 성별:
                    <StackPanel Grid.Column="3" Orientation="Horizontal"
                                Background="LightGreen">
                        <RadioButton IsChecked="{Binding Path=Male}"
                                Content="남" />
                    </StackPanel>
                    <StackPanel Grid.Column="3" Orientation="Horizontal"
                                Background="LightGreen">
                        <RadioButton IsChecked="{Binding Path=Male,
                      Converter={StaticResource maleConverter}}" Content="여" />
                    </StackPanel>
                </TextBlock>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
```

ListBox.ItemTemplate 속성을 사용하여 DataTemplate 을 정의한다. DataTemplate 은 아이템의 인라인 요소를 표현하기 위해 TextBlock 을 사용하고 있으며 TextBlock 은 표현될 수 있는 인라인 요소들 만들고자 할 때 사용된다.

여기서 주요 내용은 ListBox 의 아이템으로 RadioBox 가 사용되고 있으며 각각을 여전히 바인딩 문법으로 표현될 수 있다는 것이다.

다음은 ListBox 와 매핑되는 컬렉션 원소의 바인딩을 표현한 그림이다.



데이터 템플릿을 요소의 리소스로 정의하고 DataType 속성에 데이터 원본의 객체가 리스트형 아이템에 어떻게 표현(랜더링이라 한다)되어야 할 지를 형식화하면 같은 형식(DataType 속성에 설정된)의 모든 데이터이 표현을 형식화 할 수있다.

다음은 요소의 리소스에 데이터 템플릿을 정의하고 형식화한 예제다.

<코드>

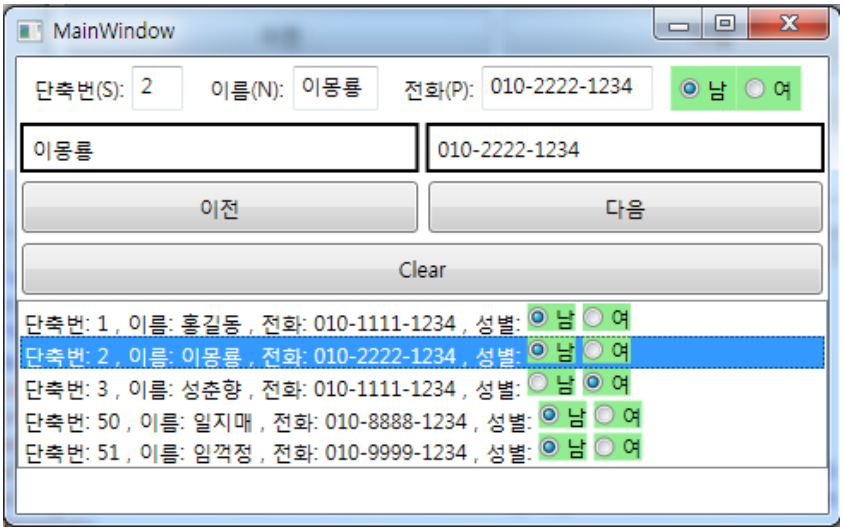
[예제 05-24] 리소스를 사용한 데이터 템플릿 형식화

```
<Window.Resources>
        <local:People x:Key="people">
            <local:Person ShortNumber = "50" Name = "일지매"
                Phone = "010-8888-1234" Male = "True" />
            <local:Person ShortNumber = "51" Name = "임꺽정"
                Phone = "010-9999-1234" Male = "True" />
        </local:People>
        <local:MaleToFemaleConverter x:Key="maleConverter" />

        <DataTemplate DataType="{x:Type local:Person}">
            <TextBlock>
                        단축번:
                        <TextBlock Text="{Binding Path=ShortNumber}" />
                        , 이름:
                        <TextBlock Text="{Binding Path=Name}" />
                        , 전화:
                        <TextBlock Text="{Binding Path=Phone}" />
                        , 성별:
                        <StackPanel Grid.Column="3" Orientation="Horizontal"
                                Background="LightGreen">
                            <RadioButton IsChecked="{Binding Path=Male}"
                                Content="남" />
                        </StackPanel>
                        <StackPanel Grid.Column="3" Orientation="Horizontal"
                                Background="LightGreen">
                            <RadioButton IsChecked="{Binding Path=Male,
                          Converter={StaticResource maleConverter}}" Content="여" />
                        </StackPanel>
                    </TextBlock>
        </DataTemplate>
    </Window.Resources>
...
<ListBox ItemsSource="{Binding}" IsSynchronizedWithCurrentItem="True" />
```

</코드>

<결과>

[출력 결과]



</결과>

출력 결과는 같으며 다음 코드처럼 ListBox 의 ItemTemplate 속성을 정의하지 않고

```
<ListBox ItemsSource="{Binding}" IsSynchronizedWithCurrentItem="True" />
```

다음과 같이 Window 요소 리소스에서 DataTemplate 을 정의한다.

```
<Window.Resources>
...
        <DataTemplate DataType="{x:Type local:Person}">
            <TextBlock>
                        단축번:
                        <TextBlock Text="{Binding Path=ShortNumber}" />
                        , 이름:
                        <TextBlock Text="{Binding Path=Name}" />
                        , 전화:
                        <TextBlock Text="{Binding Path=Phone}" />
                        , 성별:
                        <StackPanel Grid.Column="3" Orientation="Horizontal"
                                    Background="LightGreen">
                            <RadioButton IsChecked="{Binding Path=Male}"
                                    Content="남" />
                        </StackPanel>
                        <StackPanel Grid.Column="3" Orientation="Horizontal"
                                    Background="LightGreen">
                            <RadioButton IsChecked="{Binding Path=Male,
                      Converter={StaticResource maleConverter}}" Content="여" />
                        </StackPanel>
                    </TextBlock>
        </DataTemplate>
    </Window.Resources>
```

나머지 바인딩 코드는 같고 DataTemplate 의 형식화 형식을 지정하기 위한 DataType 속성을 다음과 같이 <DataTemplate DataType="{x:Type local:Person}"> 정의 한다.

People 컬렉션의 원소가 삭제되면 People 컬렉션을 데이터 원본으로 사용하고 있는 ListBox 컨트롤의 아이템은 어떻게 될까? People 컬렉션은 단지 일반 List<> 컬렉션이므로 바인딩

시스템은 People 컬렉션의 변화를 알지 못하고 ListBox 컨트롤의 아이템도 변경 시킬 수 없다. 컬렉션의 변경을 바인딩된 리스트 형태의 UI 에 적용하려면 INotifyCollectionChanged 인터페이스를 구현한 바인딩 전용 컬렉션을 사용하던가 직접 인터페이스를 구현한 컬렉션을 사용해야 한다.

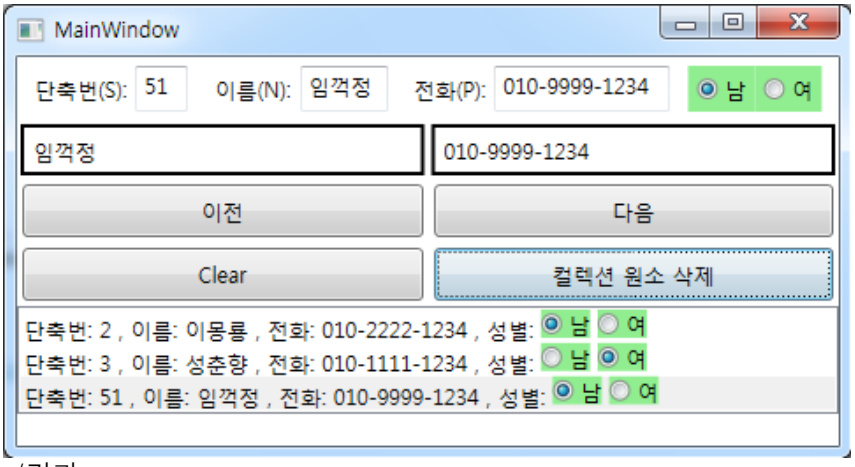
일반적으로 이미 INotifyCollectionChanged 인터페이스를 구현한 바인딩 전용 컬렉션 ObservableCollection<>을 사용해 쉽게 구현할 수 있다.

다음은 ObservableCollection<> 컬렉션을 사용한 People 객체의 ListBox 컨트롤 예제다.

<코드>
[예제 05-25]

```
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Button Name="eraseButton" Grid.Column="0" Margin="3" Height="30"
                Content="Clear" Click="eraseButton_Click" />
            <Button Name="removeButton" Grid.Column="1" Margin="3" Height="30"
                Content="컬렉션 원소 삭제" Click="removeButton_Click" />
        </Grid>
        <ListBox ItemsSource="{Binding}" Name="listbox"
                IsSynchronizedWithCurrentItem="True" />

        private void removeButton_Click(object sender, RoutedEventArgs e)
        {
            People people = (People)FindResource("people");

            if (listbox.SelectedIndex >= 0)
                people.RemoveAt(listbox.SelectedIndex);
        }

//public class People : List<Person>
    public class People : ObservableCollection<Person>
    {
        public People()
        {
            Add(new Person() { ShortNumber = 1, Name = "홍길동",
                Phone = "010-1111-1234", Male = true });
            Add(new Person() { ShortNumber = 2, Name = "이몽룡",
                Phone = "010-2222-1234", Male = true });
            Add(new Person() { ShortNumber = 3, Name = "성춘향",
                Phone = "010-1111-1234", Male = false });
        }
    }
```

</코드>
<결과>
[출력 결과]



</결과>

결과는 선택한 컬렉션 원소 두 개를 삭제한 후의 결과로 ListBox 컨트롤의 두 아이템이 삭제된 내용을 확인할 수 있다.

INotifyCollectionChanged 인터페이스를 구현한 바인딩 전용 컬렉션 ObservableCollection<>을 상속 받아 People 컬렉션이 다음과 같이 정의되며

```
public class People : ObservableCollection<Person>
```

다음과 같이 원소 삭제 버튼이 클릭되면 다음 코드를 실행하여

```
private void removeButton_Click(object sender, RoutedEventArgs e)
{
    People people = (People)FindResource("people");

    if (listbox.SelectedIndex >= 0)
        people.RemoveAt(listbox.SelectedIndex);
}
```

ListBox 컨트롤의 현재 선택된 아이템 번호에 해당하는 컬렉션 원소를 삭제한다. 만약 People 컬렉션이 List<>를 상속받아 다음처럼 구현되었다면 아무 변화가 없을 것이다.

```
public class People : List<Person>
```

일반 .Net 컬렉션은 INotifyCollectionChanged 인터페이스를 구현하지 않으므로 ListBox 컨트롤의 변경이 없다.

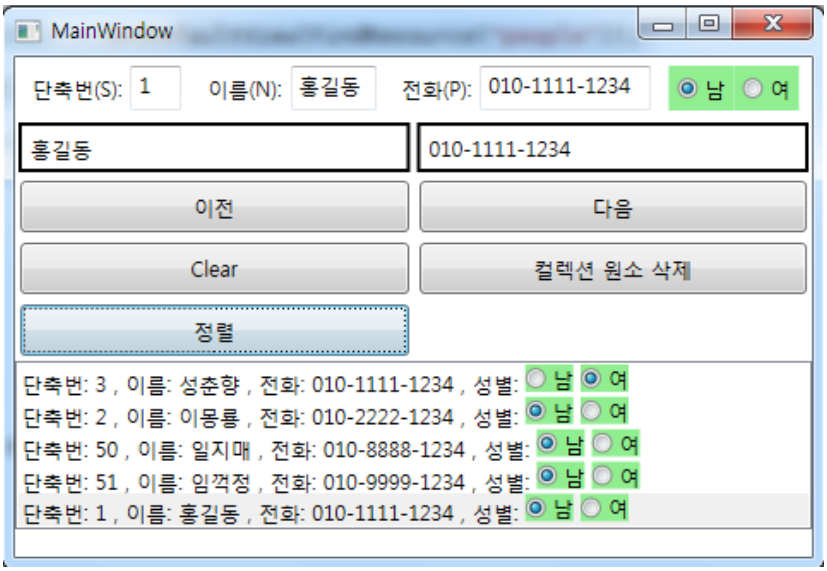
WPF 의 바인딩 시스템은 리스트 형태의 UI 컨트롤의 정렬이나 필터링 및 그룹화 등의 기능도 제공하며 이 기능들의 핵심이 뷰(view) 객체다. 뷰는 바인딩 시스템이 제공하는 UI 요소와 데이터 원본 사이의 인터페이스를 수행하는 객체로 리스트 형태의 요소들이 아이템들을 어떻게 보이게 할 것인지를 결정한다.

다음은 뷰를 이용하여 데이터를 정렬하는 예제로 뷰의 SortDescriptions 속성을 사용하여 데이터의 정렬을 결정한다.

<코드>
[예제 05-26] 뷰를 이용한 ListBox 컨트롤 정렬

```
<Grid >
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Button Name="sortButton" Grid.Column="0" Margin="3" Height="30"
        Content="정렬" Click="sortButton_Click" />
</Grid>

private void sortButton_Click(object sender, RoutedEventArgs e)
{
    ICollectionView view =
        CollectionViewSource.GetDefaultView(FindResource("people"));

    if (view.SortDescriptions.Count == 0)
    {
        view.SortDescriptions.Add(new SortDescription("Name",
            ListSortDirection.Ascending));
        view.SortDescriptions.Add(new SortDescription("Male",
            ListSortDirection.Ascending));
    }
    else
    {
        view.SortDescriptions.Clear();
    }
}
```

</코드>
<결과>
[출력 결과]



</결과>

결과는 정렬 버튼을 클릭하면 ListBox 컨트롤의 아이템을 이름별로 먼저 정렬한 후 이름이 같으면 성별로 정렬한다.

다음 코드는

```
ICollectionView view =
    CollectionViewSource.GetDefaultView(FindResource("people"));

if (view.SortDescriptions.Count == 0)
{
    view.SortDescriptions.Add(new SortDescription("Name",
        ListSortDirection.Ascending));
    view.SortDescriptions.Add(new SortDescription("Male",
        ListSortDirection.Ascending));
}
else
{
    view.SortDescriptions.Clear();
}
```

people 컬렉션의 View 를 얻고 SortDescrptions 속성에 내장 형식인 SortDescription 객체를 생성하여 추가하는 코드로 SortDescription 객체의 첫 인수는 정렬할 Person 객체의 속성이며 두 번째 인수는 내림차순, 오름차순 정렬 규칙이다.

사용자 정의 정렬 규칙을 만들고 싶다면 IComparer 인터페이스를 구현한 객체를 뷰의 CustomSort 속성에 추가하여 사용할 수 있다.

다음은 사용자 정의 정렬 규칙을 적용한 바인딩 예제다.

<코드>
[예제 05-27] 사용자 정의 정렬 규칙을 사용한 정렬

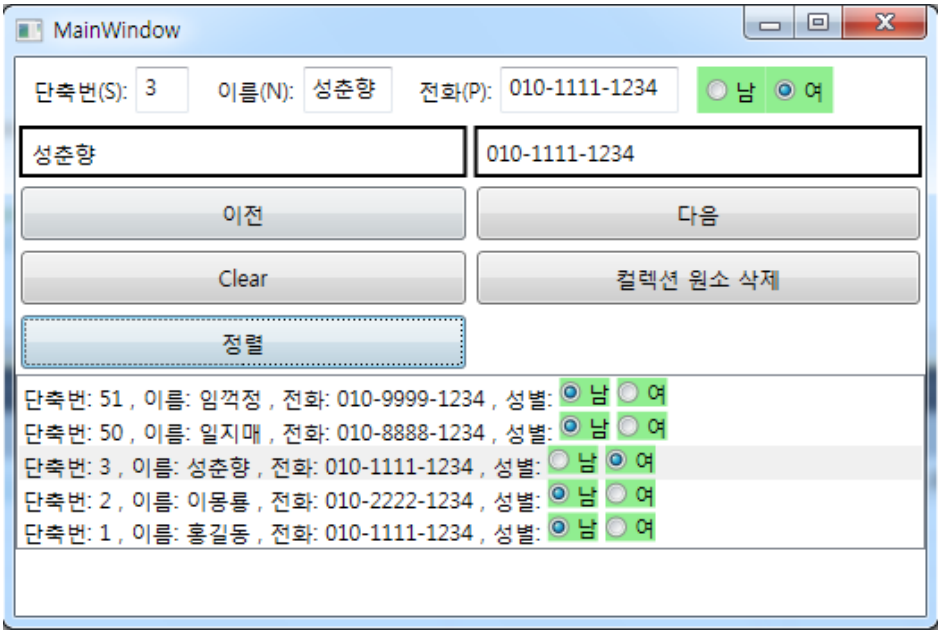
```
private void sortButton_Click(object sender, RoutedEventArgs e)
{
    ICollectionView view =
        CollectionViewSource.GetDefaultView(FindResource("people"));
    ListCollectionView listView = (ListCollectionView)view;

    if (listView.CustomSort == null)
    {
        listView.CustomSort = new CustomSorter();
    }
    else
    {
        listView.CustomSort = null;
    }
}

class CustomSorter : IComparer
{
    public int Compare(object x, object y)
```

```
        {
            Person per1 = (Person)x;
            Person per2 = (Person)y;

            return (int)per2.ShortNumber - (int)per1.ShortNumber;
        }
    }
```

</코드>

<결과>

[출력 결과]



</결과>

정렬은 단축번호를 기준으로 정렬하도록 하며 다음 원소와 이전 원소 차를 이용해 정렬하므로 내림 차순 정렬로 보일 것이다.

다음 코드는

```
        ICollectionView view =
                CollectionViewSource.GetDefaultView(FindResource("people"));
        ListCollectionView listView = (ListCollectionView)view;

        if (listView.CustomSort == null)
        {
            listView.CustomSort = new CustomSorter();
        }
        else
        {
            listView.CustomSort = null;
        }
```

사용자 정의 정렬 규칙을 사용하기 위해 뷰 객체를 얻고 ListCollectionView 객체로 ICollectionView 뷰 인터페이스를 형식 변환한다. 어떤 뷰 객체는 사용자 정의 정렬 규칙을 제공하지 않으므로 CustomSort 속성을 가지고 있지 않을 수 있다. 정렬 규칙이 없으면

CustomSort 속성에 CustomSorter 사용자 정렬 규칙을 추가하고 있으면 다시 정렬 규칙을 제거한다.

다음 코드는

```
class CustomSorter : IComparer
{
    public int Compare(object x, object y)
    {
        Person per1 = (Person)x;
        Person per2 = (Person)y;

        return (int)per2.ShortNumber - (int)per1.ShortNumber;
    }
}
```

사용자 정렬 규칙을 생성하는 클래스 정의로 IComparer 인터페이스를 구현한다. Compare(x,y)는 1, -1, 0을 반환하며 1은 앞 원소가 크게 정렬되며 -1은 뒤 원소가 크게 정렬, 0은 같게 정렬하는 규칙을 만든다.

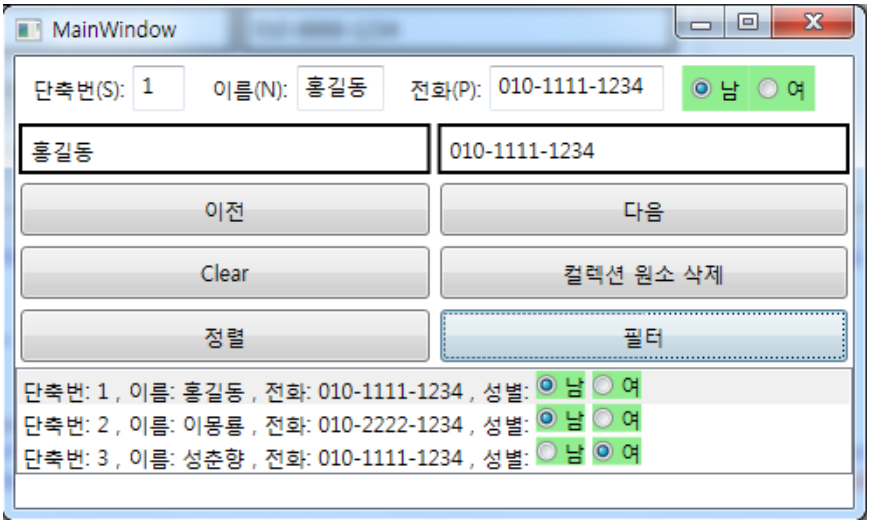
<코드>

[예제 05-28] 필터링을 사용한 ListBox 컨트롤

```
<Grid >
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Button Name="sortButton" Grid.Column="0" Margin="3" Height="30"
        Content="정렬" Click="sortButton_Click" />
    <Button Name="filterButton" Grid.Column="1" Margin="3" Height="30"
        Content="필터" Click="filterButton_Click" />
</Grid>

private void filterButton_Click(object sender, RoutedEventArgs e)
{
    ICollectionView view =
        CollectionViewSource.GetDefaultView(FindResource("people"));
    if (view.Filter == null)
    {
        view.Filter = delegate(object obj)
        {
            return ((Person)obj).ShortNumber < 50;
        };
    }
    else
    {
```

```
                view.Filter = null;
            }
        }
```

</코드>
<결과>
[출력 결과]



</결과>
필터링을 했을 때의 결과로 단축 번호가 50 이상인 원소는 필터링되어 보이지 않는다.

다음 코드는

```
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Button Name="sortButton" Grid.Column="0" Margin="3" Height="30"
                    Content="정렬" Click="sortButton_Click" />
            <Button Name="filterButton" Grid.Column="1" Margin="3" Height="30"
                    Content="필터" Click="filterButton_Click" />
        </Grid>
```

정렬 버튼과 필터 버튼을 생성하고 클릭 이벤트가 발생하면 각각의 sortButton_Click 핸들러와 filterButton_Click 핸들러를 설정하는 코드다.

다음 코드는

```
        private void filterButton_Click(object sender, RoutedEventArgs e)
        {
            ICollectionView view = CollectionViewSource.GetDefaultView(FindResource("people"));
            if (view.Filter == null)
            {
                view.Filter = delegate(object obj)
                {
                    return ((Person)obj).ShortNumber < 50;
                };
            }
```

```
            else
            {
                view.Filter = null;
            }
        }
```

people 컬렉션의 뷰를 얻어 Filter 속성에 필터링하기 위한 predicate 를 대리자 형태로 설정하는 코드로 Filter 가 없다면 단축 번호가 50이하인 predicate 를 설정하며 이미 있다면 피터를 제거한다.

Filter 는 bool 형식을 반환하는 predicate 로 Predicate<Object> 형태의 대리자 제너릭이다.

리스트 형태의 데이터를 일정한 그룹으로 나누고자 한다면 뷰의 그룹화 기능을 이용할 수 있다. WPF 는 다양한 그룹화 기능을 제공하며 우리는 간단한 그룹화 기능을 살펴볼 것이다. ItemsControl 클래스를 상속한 클래스라면 모두 출려하는 아이템을 그룹화할 수 잇으며 여러가지 그룹 스타일을 적용하여 보여줄 수 있다.

다음은 그룹화 기능을 추가한 ListBox 컨트롤 예제다.

<코드> 뷰의 그룹화 기능 이용하기
[예제 05-29]

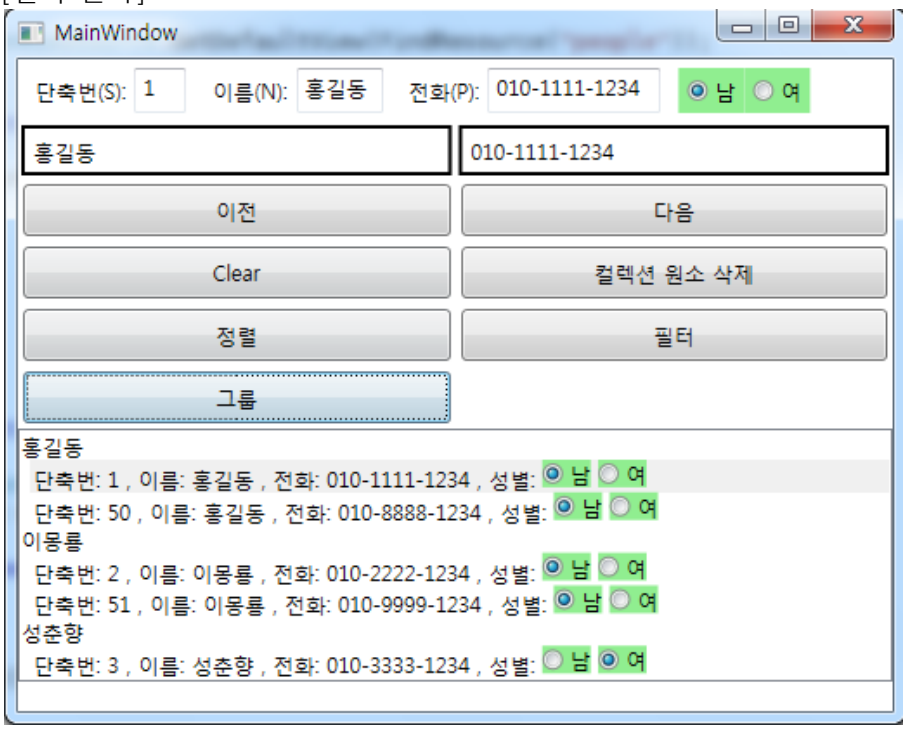
```
<Window.Resources>
        <local:People x:Key="people">
            <local:Person ShortNumber = "50" Name = "홍길동"
                Phone = "010-8888-1234" Male = "True" />
            <local:Person ShortNumber = "51" Name = "이몽룡"
                Phone = "010-9999-1234" Male = "True" />
        </local:People>
        <local:MaleToFemaleConverter x:Key="maleConverter" />

        <DataTemplate DataType="{x:Type local:Person}">
            <TextBlock>
                            단축번:
                            <TextBlock Text="{Binding Path=ShortNumber}" />
                            , 이름:
                            <TextBlock Text="{Binding Path=Name}" />
                            , 전화:
                            <TextBlock Text="{Binding Path=Phone}" />
                            , 성별:
                            ...
                        </TextBlock>
        </DataTemplate>
    </Window.Resources>
    <StackPanel DataContext="{StaticResource people}">
        ...
        <Grid >
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
```

```
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Button Name="prev" Grid.Column="0" Margin="3" Height="30"
                Content="이전" Click="prev_Click" />
        <Button Name="next" Grid.Column="1" Margin="3" Height="30"
                Content="다음" Click="next_Click" />
    </Grid>
    <Grid >
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Button Name="eraseButton" Grid.Column="0" Margin="3" Height="30"
                Content="Clear" Click="eraseButton_Click" />
        <Button Name="removeButton" Grid.Column="1" Margin="3" Height="30"
                 Content="컬렉션 원소 삭제" Click="removeButton_Click" />
    </Grid>
    <Grid >
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Button Name="sortButton" Grid.Column="0" Margin="3" Height="30"
                Content="정렬" Click="sortButton_Click" />
        <Button Name="filterButton" Grid.Column="1" Margin="3" Height="30"
                Content="필터" Click="filterButton_Click" />
    </Grid>
    <Grid >
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Button Name="groupButton" Grid.Column="0" Margin="3" Height="30"
                Content="그룹" Click="groupButton_Click" />
    </Grid>
    <ListBox ItemsSource="{Binding}" Name="listbox"
                IsSynchronizedWithCurrentItem="True">
        <ListBox.GroupStyle>
            <x:Static Member="GroupStyle.Default" />
        </ListBox.GroupStyle>
    </ListBox>
</StackPanel>
```

```
    private void groupButton_Click(object sender, RoutedEventArgs e)
    {
        ICollectionView view =
            CollectionViewSource.GetDefaultView(FindResource("people"));
        if (view.GroupDescriptions.Count == 0)
        {
            view.GroupDescriptions.Add(new PropertyGroupDescription("Name"));
        }
        else
        {
            view.GroupDescriptions.Clear();
        }
    }
```

```
public class People : ObservableCollection<Person>
{
    public People()
    {
        Add(new Person() { ShortNumber = 1, Name = "홍길동",
            Phone = "010-1111-1234", Male = true });
        Add(new Person() { ShortNumber = 2, Name = "이몽룡",
            Phone = "010-2222-1234", Male = true });
        Add(new Person() { ShortNumber = 3, Name = "성춘향",
            Phone = "010-3333-1234", Male = false });
    }
}
```

</코드>
<결과>
[출력 결과]



</결과>

이름으로 그룹화를 수행했을 때의 결과로 같은 이름에 해당하는 홍길동과 이몽룡은 두 아이템들이 그룹화 되었다.

다음 코드는

```
<local:People x:Key="people">
    <local:Person ShortNumber = "50" Name = "홍길동"
        Phone = "010-8888-1234" Male = "True" />
    <local:Person ShortNumber = "51" Name = "이몽룡"
        Phone = "010-9999-1234" Male = "True" />
</local:People>
```

그룹화를 테스트 하기 위해 XAML 코드에서 정의한 Person 객체들이며 다음 코드는

```
public class People : ObservableCollection<Person>
{
    public People()
    {
        Add(new Person() { ShortNumber = 1, Name = "홍길동",
            Phone = "010-1111-1234", Male = true });
        Add(new Person() { ShortNumber = 2, Name = "이몽룡",
            Phone = "010-2222-1234", Male = true });
        Add(new Person() { ShortNumber = 3, Name = "성춘향",
            Phone = "010-3333-1234", Male = false });
    }
}
```

생성자에서 Person 객체를 추가한 코드다. 홍길동의 이름을 갖는 Person 객체가 두명이며 이몽룡의 이름을 갖는 Person 객체가 두명이다.

다음 코드는

```
<ListBox ItemsSource="{Binding}" Name="listbox"
        IsSynchronizedWithCurrentItem="True">
    <ListBox.GroupStyle>
        <x:Static Member="GroupStyle.Default" />
    </ListBox.GroupStyle>
</ListBox>
```

그룹화를 위해 설정해야하는 GroupStyle 속성으로 GroupStyle.Default 설정은 그룹화 후 그룹명과 간단하게 들여 쓰기를 한 아이템들의 목록을 보여준다.

다음 코드는

```
private void groupButton_Click(object sender, RoutedEventArgs e)
{
    ICollectionView view =
        CollectionViewSource.GetDefaultView(FindResource("people"));
    if (view.GroupDescriptions.Count == 0)
    {
        view.GroupDescriptions.Add(new PropertyGroupDescription("Name"));
    }
    else
    {
        view.GroupDescriptions.Clear();
    }
}
```

people 컬렉션의 뷰를 얻고 GroupDescriptions 속성의 그룹을 추가하기 위한 코드로 PropertyGroupDescription 객체를 사용하여 그룹화하려는 속성의 이름을 인수로 전달한다. 그룹화 객체가 없으면 추가하고 있으면 제거한다.

기본적인 그룹 스타일(GroupStyle)로도 한 눈에 그룹핑을 확인할 수 있지만 사용자가 필요한 내용을 더 추가하여 그룹 스타일을 정의하고자 한다면 사용자 정의 데이터 템플릿을 이용할 수 있다.

다음은 사용자 정의 데이터 템플릿을 적용한 그룹 스타일 예제다.

<코드>
[예제 05-30]

```
        <ListBox ItemsSource="{Binding}" Name="listbox"
                 IsSynchronizedWithCurrentItem="True">
            <ListBox.GroupStyle>
                <GroupStyle>
                    <GroupStyle.HeaderTemplate>
                        <DataTemplate>
                            <TextBlock Foreground="White" Background="Green">
                            <TextBlock Text="{Binding Path=Name}" />
                          [Item Count:<TextBlock Text="{Binding Path=ItemCount}" />]
                            </TextBlock>
                        </DataTemplate>
                    </GroupStyle.HeaderTemplate>
                </GroupStyle>
            </ListBox.GroupStyle>
        </ListBox>
```

</코드>
<결과>
[출력 결과]

</결과>
GroupStyle.HeaderTemplate 속성에 DataTemplate 을 정의하고 그룹명을 초록색 바탕의 흰색 텍스트로 이름을 출력하고 각 그룹의 아이템 개수를 출력한다.

다음 코드는

```
<GroupStyle.HeaderTemplate>
        <DataTemplate>
            <TextBlock Foreground="White" Background="Green">
            <TextBlock Text="{Binding Path=Name}" />
          [Item Count:<TextBlock Text="{Binding Path=ItemCount}" />]
            </TextBlock>
        </DataTemplate>
    </GroupStyle.HeaderTemplate>
```

그룹명을 표혀하기 위한 데이터 템플릿의 정의로 그룹명은 Name 속성으로 바인딩하며 아이템의 개수는 그룹의 아이템 개수를 보관하는 ItemCount 속성에 바인딩한다.