C#WPF の道【XAML に慣れる編】

WindowsForms プログラマーが WPF を書く方法

ピーコックアンダーソン 著

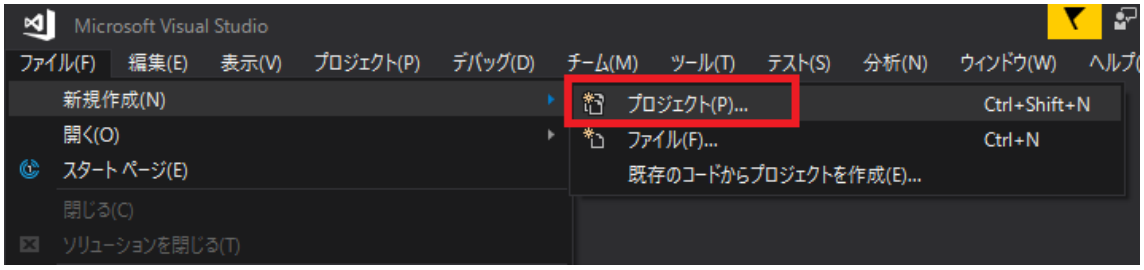
目次

#1　WPF プロジェクトの作成

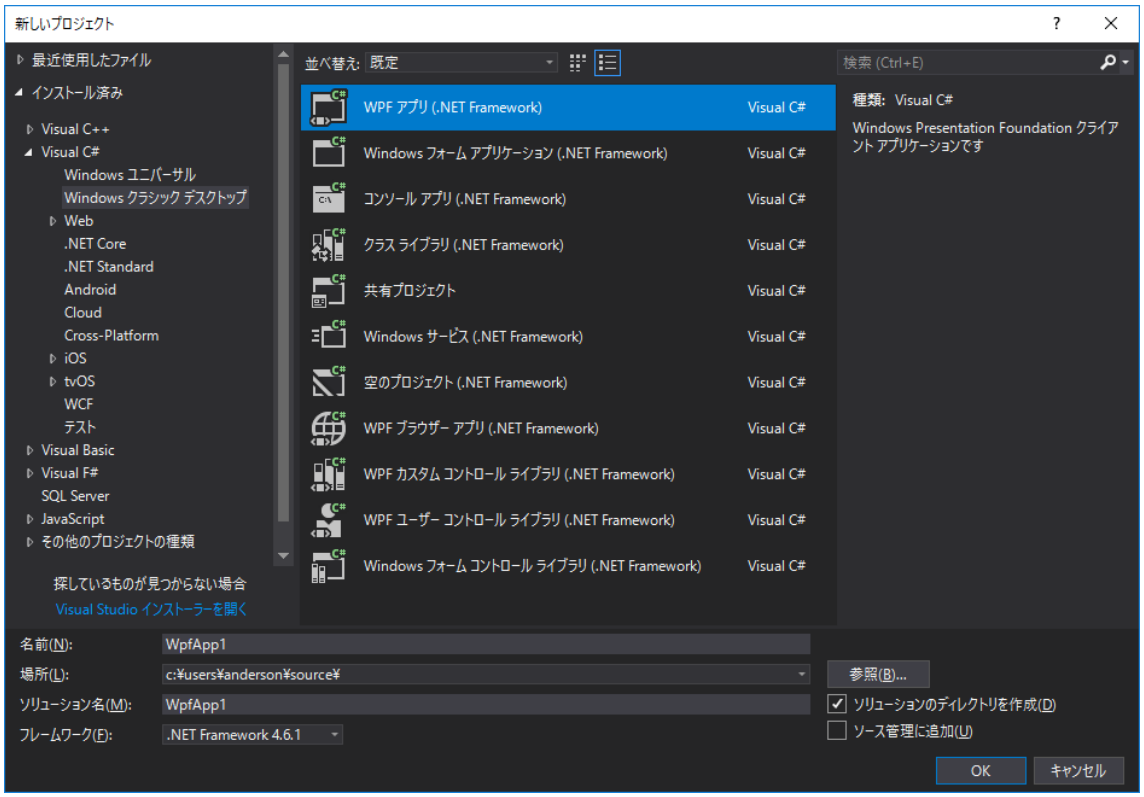
WPF のプログラムを作成するには、WPF のプロジェクトを作成する必要があります。今回は VisualStudio2017 で作成していきます。

WPF プロジェクトの作成方法

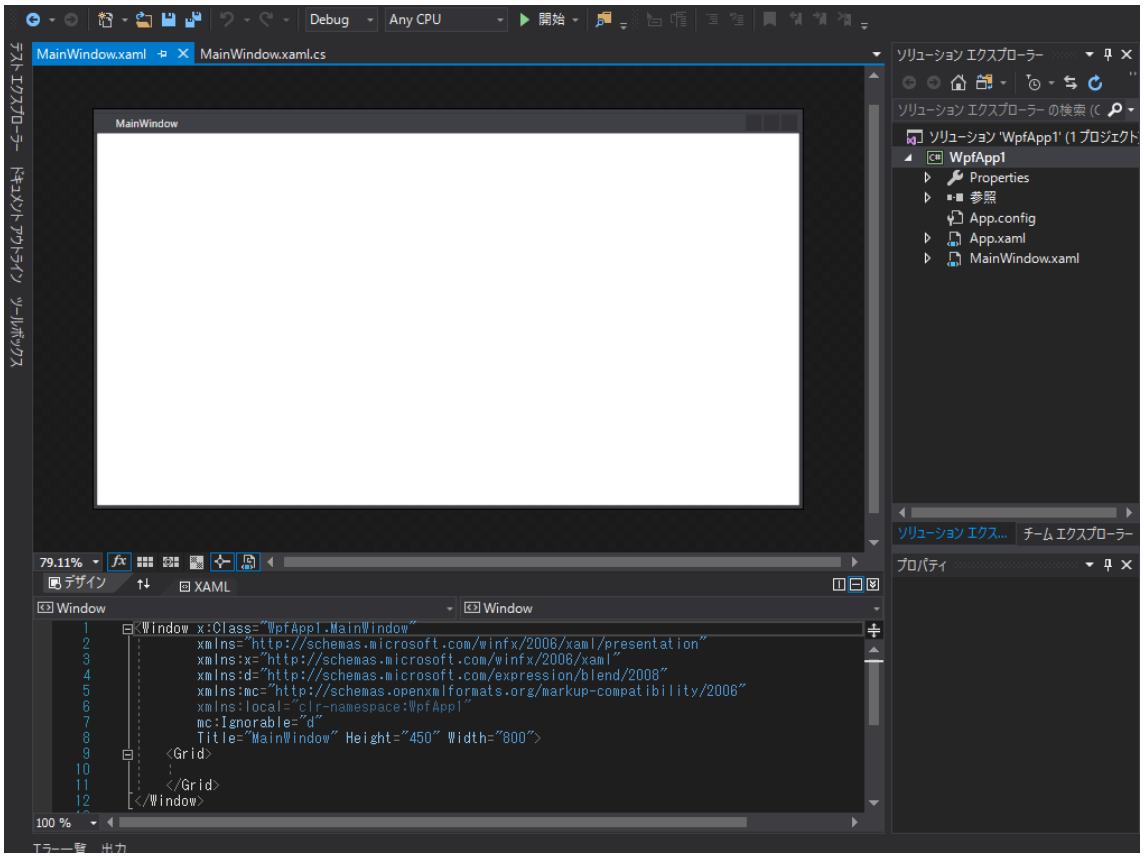
「ファイル」「新規作成」「プロジェクト」の順に選択していきます。



「Visual C#」の「Windows クラシックデスクトップ」を選択し、「WPF アプリ(.NETFramework)」を選択し、任意の名前と場所、ソリューション名を設定して OK ボタンを押下します。



次のように画面が表示されれば、プロジェクトの作成は完了です。



#2　StackPanel

StackPanel とは

StackPanel とは、コントロールを縦方向か横方向に整列して配置してくれるコントロールです。Orientation プロパティを指定して、縦方向に整列するのか、横方向に整列するのかを決定します。

縦方向に並べる場合：Vertical（何も指定しなければ、こちらが選択されます）
横方向に並べる場合：Horizontal

StackPanel の書き方

WPF では Xaml という言語を使って、画面のデザインをしていきます。StackPanel を配置する場合は次のようになります。

横方向にコントロールを配置する場合

```
<StackPanel Orientation=" Horizontal ">
    <Label Height="30"/>
    <TextBox  Height="30"/>
    <TextBox Height="30"/>
    <Button  Height="30"/>
</StackPanel>
```

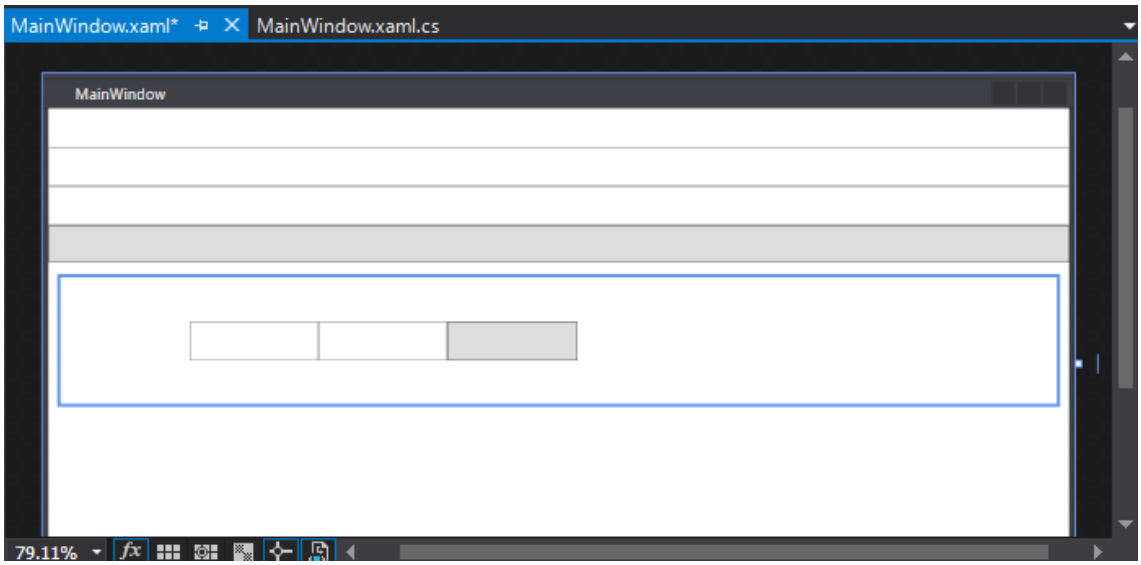
サンプルコード

縦向きと横向きでコントロールを配置する場合、次のように StackPanel を入れ子にします。最初の StackPanel でコントロールを縦向きに配置し、その下から横方向にコントロールを配置していくために、別の StackPanel を配置し、その中にコントロールを配置しています。また、各コントロールは、Height と Width プロパティを調整して、サイズを設定しています。

```
<Window x:Class="WPF002.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF002"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <StackPanel>
            <Label Height="30"/>
            <TextBox Height="30"/>
```

```
            <TextBox Height="30"/>
            <Button Height="30"/>

            <StackPanel
                Orientation="Horizontal"
                Height="100"
                Margin="10">

                <Label Height="30" Width="100"/>
                <TextBox Height="30" Width="100"/>
                <TextBox Height="30" Width="100"/>
                <Button Height="30" Width="100"/>
            </StackPanel>
        </StackPanel>

    </Grid>
</Window>
```

この Xaml を記述したときの画面レイアウトは次のようになります。



VerticalAlignment と HorizontalAlignment

VerticalAlignment プロパティを指定することで、コントロールをどこから並べるかを指定できます。StackPanel の上からなら Top、下からなら Bottom、真ん中からなら Center を選びます。

HorizontalAlignment プロパティを指定することで、コントロールをどこから並べるかを指定できます。StackPanel の左からなら Left、右に寄せるなら Right、中心なら Center です。Stretch を選ぶと全体にひきのばされます。

Margin について

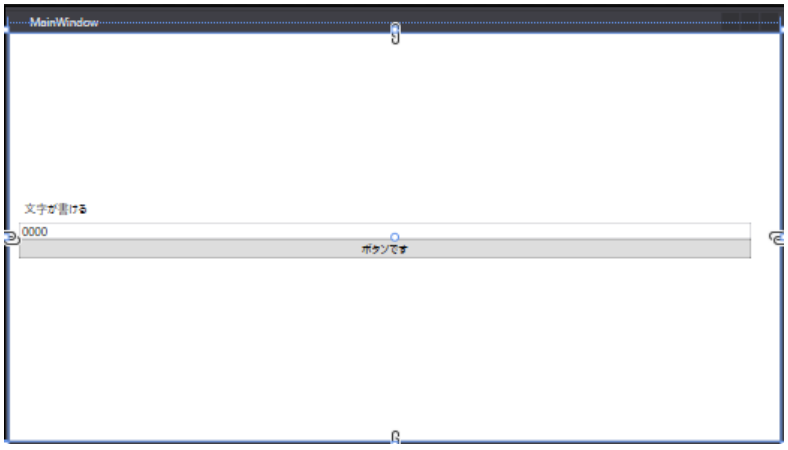
Margin を設定することで余白の設定をすることができます。Margin="10,20,30,40"と指定することで、左、上、右、下の順で余白が設定できます。

各コントロールのテキスト表示

StackPanel とは直接関係ないですが、Label 等の各コントロールにテキストを設定する方法を解説します。WindowsForms の場合はすべて Text プロパティに設定していましたが、Label や Button の場合は Content というプロパティに設定します。TextBox は WindowsForms と同様に Text プロパティとなります。

```
<Window x:Class="WPF002_2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF002_2"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <StackPanel VerticalAlignment="Center"
                    HorizontalAlignment="Stretch"
                    Margin="10,20,30,40">
            <Label Content="文字が書ける"/>
            <TextBox Text="0000"/>
            <Button Content="ボタンです"/>
        </StackPanel>
    </Grid>
</Window>
```

この Xaml を記述したときの画面レイアウトは次のようになります。



#3　Grid

Grid とは？

Grid とは、画面レイアウトを行と列で構成し、柔軟のあるグリッド領域を作り、そこにコントロールを配置することで、キレイな画面レイアウトを作成することができます。
イメージ的には Excel のセルのような感じで、「何列目の何番目に Button コントロールを設置する」といった使い方で、コントロールを配置することができます。Excel のセル結合のようなこともできるので、希望のマス構成で、レイアウトを作成することができます。

列の定義の書き方

Grid.ColumnDefinitions の定義の中に、必要な列の数だけ「ColumnDefinition」を記述します。列のはばは Width プロパティに絶対値で「100」等のピクセルを指定することもできますが、「*」を指定することで、均等な比率で横幅を調整することができます。例えば 3 列宣言していて、すべての列の定義が「＊」の場合は、均等に 3 列の幅が作られます。「２＊」などと、数字を＊の前に記述すると、その列のみ 2 倍などの指定した値の倍率で確保されます。Width プロパティに「Auto」を指定すると、設置したコントロールの幅で動的に変動します。

列の定義の書き方の例

```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="123"/>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="*"/>
    <ColumnDefinition Width="2*"/>
</Grid.ColumnDefinitions>
```

行の定義の書き方

Grid.RowDefinitions の定義の中に、必要な列の数だけ「RowDefinition」を記述します。列のはばは Height プロパティに絶対値で「100」等のピクセルを指定することもできますが、「*」を指定することで、均等な比率で高さを調整することができます。例えば 6 行宣言していて、すべての行の定義が「＊」の場合は、均等に 6 行の幅が作られます。「２＊」などと、数字を＊の前に記述すると、その行のみ 2 倍などの指定した値の倍率で確保されます。Height プロパティに「Auto」を指定すると、設置したコントロールの高さで動的に変動します。

行の定義の書き方の例

```
        <Grid.RowDefinitions>
            <RowDefinition Height="123"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="2*"/>
        </Grid.RowDefinitions>
```

コントロールをグリッドのどのマスに設置するかの指定方法

コントロールをグリッドのマスに設置する場合は、Grid.Row と Grid.Column で配置する場所を指定します。インデックスは左上から 0 始まりです。

ボタンを 1 行目の 2 列目に配置する例（0 行目＆0 列目始まり）

```
        <Button Grid.Row="1"
                Grid.Column="2"/>
```

セルを結合するときは、• Grid.ColumnSpan および Grid.RowSpan を使用します。指定した値だけセルを結合します。

サンプルコード

次のサンプルコードでは、電卓イメージのレイアウトを作成しています。最初に値を表示するラベルを設置し、その下に、電卓ボタンを設置しています。各ボタンは Margin を 5 にすることで、コントロールの周りの余白を 5 ピクセル確保しています。また上部のラベルと「0」のボタンは ColumnSpan でセルの結合をしています。

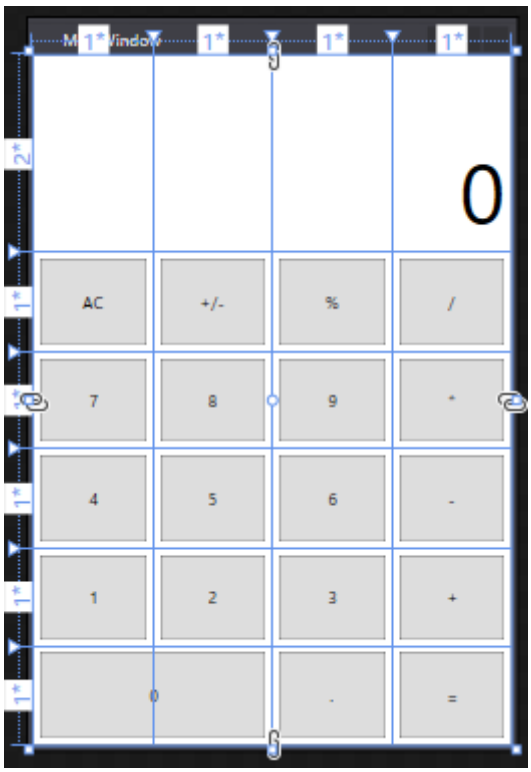
電卓イメージのレイアウト Xaml

```
<Window x:Class="WPF003.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF003"
        mc:Ignorable="d"
        Title="MainWindow" Height="525" Width="350">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="2*"/>
            <RowDefinition Height="*"/>
```

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

        <Label Content="0"
               HorizontalAlignment="Right"
               VerticalAlignment="Bottom"
               FontSize="60"
               Grid.ColumnSpan="4"/>

        <Button Content="AC"
                Margin="5"
                Grid.Column="0"
                Grid.Row="1"/>

        <Button Content="+/-"
                Margin="5"
                Grid.Row="1"
                Grid.Column="1"/>
        <Button Content="%"
                Margin="5"
                Grid.Row="1"
                Grid.Column="2"/>
        <Button Content="/"
                Margin="5"
                Grid.Row="1"
                Grid.Column="3"/>

        <Button Content="7"
                Margin="5"
                Grid.Row="2"
                Grid.Column="0"/>
        <Button Content="8"
                Margin="5"
                Grid.Row="2"
                Grid.Column="1"/>
        <Button Content="9"
                Margin="5"
                Grid.Row="2"
                Grid.Column="2"/>
        <Button Content="*"
                Margin="5"
                Grid.Row="2"
                Grid.Column="3"/>

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

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

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

        <Button Content="0"
                Margin="5"
                Grid.Row="5"
                Grid.Column="0"
                Grid.ColumnSpan="2"/>
        <Button Content="."
                Margin="5"
                Grid.Row="5"
                Grid.Column="2"/>
        <Button Content="="
                Margin="5"
                Grid.Row="5"
                Grid.Column="3"/>

    </Grid>
</Window>
```

このXamlを記述したときの画面レイアウトは次のようになります。


1*
1*
1*
1*
2*
0
AC
+/-
%
/
7
8
9
*
4
5
6
-
1
2
3
+
0
.
=

#4　コントロールの名前の付け方

コントロールの名前とは？

WindowsForms で UI を作成する場合は、コントロールに名前を付けるというのが常識でした。すべてのコントロールは名前を指定して各種設定を書いていました。

WPF は Xaml でコントロールのプロパティ設定等を行えるので、名前を付けなくても動作します。ただ、コントロールに対して、コードビハインド側でアクセスをしたい場合があります。そういった場合はコントロールに名前を付ける必要があります。※コードビハインドとは、MainWindow.xaml などのファイルを右クリックして「コード表示」としたときに出てくる、イベントの中身などを記述するエリアの事です。

コントロールの名前の付け方（書き方）

Xaml の Label の名前を ResultLabel にしている例

```
        <Label x:Name="ResultLabel"
               Content="0"
               HorizontalAlignment="Right"
               VerticalAlignment="Bottom"
               FontSize="60"
               Grid.ColumnSpan="4"/>
```

コードビハインド側

```
namespace WPF004
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            ResultLabel.Content = "12345";
        }
    }
}
```

コードビハインド側では InitializeComponent の後に ResultLabel の Content プロパティにアクセスしています。ここで Content の値を変更することで、Xaml では Content に「0」を設定していましたが、実行すると、画面上には「12345」と表示されることから、Xaml での Content 設定を上書きしていることが確認できます。


MainWindow
12345
AC
+/1
%
/
7
8
9
*
4
5
6
-
1
2
3
+
0
.
=

#5　イベント

イベントとは？

イベントとは「ボタンがクリックされたとき！」のように、なにかのきっかけで動作するタイミングの事です。C#は WindowsForms の頃からこのようにイベント駆動でそれぞれのタイミングごとに処理を記述するプログラミング手法となっています。

WPF でも WindowsForms と同様のイベント処理を記述することができます。定義は Xaml に記述して、イベント処理自体はコードビハインドに記述します。

イベントの書き方

Xaml のコントロールの定義をするエリアに、クリックイベントであれば Click=に続けて、イベント名を記述することで定義できます。

AButton にクリックイベントを定義する例

```
<Button x:Name="AButton"
        Content="A"
        Height="50"
        FontSize="40"
        Click="AButton_Click"/>
```

Xaml に Click と打ち込んだときに表示される＜新しいイベントハンドラー＞を選択した状態で Enter を押下すると、コードビハインドにイベントが自動生成されます。

サンプルコード

Xaml

```
<Window x:Class="WPF005.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF005"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel>
            <Label x:Name="ResultLabel"
```

```
                    Content=""
                    Height="100"
                    HorizontalAlignment="Center"
                    FontSize="60"/>
            <Button x:Name="AButton"
                    Content="A"
                    Height="50"
                    FontSize="40"
                    Click="AButton_Click"/>
            <Button x:Name="ClearButton"
                    Content="Clear"
                    Height="50"
                    FontSize="40"
                    Click="ClearButton_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

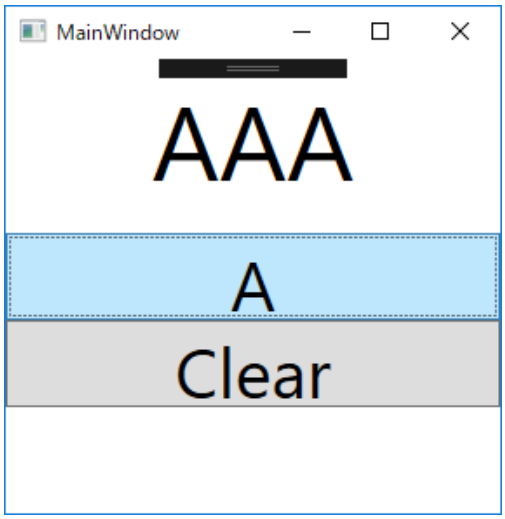
Label 1 つと Button 2 つを定義。ResultLabel と AButton と ClearButton。

コードビハインド側

```
using System.Windows;

namespace WPF005
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void AButton_Click(object sender, RoutedEventArgs e)
        {
            ResultLabel.Content += "A";
        }

        private void ClearButton_Click(object sender, RoutedEventArgs e)
        {
            ResultLabel.Content = "";
        }
    }
}
```

AButton と ClearButton のクリックイベントを自動生成させ、それぞれにロジックを記述。AButton をクリックすると、ResultLabel に A の文字が追加されていき、ClearButton をクリックするとクリアーされる処理となる。

＜実行例＞


MainWindow
AAA
A
Clear

#6　StaticResource

StaticResource とは？

WPF では複数の UI 要素で 1 つのオブジェクトを共有するための仕組みがあり、それをリソースと呼んでいます。例えば複数のボタンがあり、それぞれに同じ背景色を設定する場合、背景色の変更のたびにすべてのボタンの背景色を再設定していたのでは、手間が増えますし、間違いのもとでもあります。リソースを定義しておくと、1 つの定義を複数のコントロールで共有できるため、変更が生じた場合も 1 か所を変更するだけですべてのコントロールに反映されます。StaticResource とは、定義されたリソースをコントロール等に設定する際に使用します。

リソースの定義の書き方

リソースの定義は Window.Resources で囲って、その中に必要な定義を記述します。色の定義を行う場合は SolidColorBrush に対して、Key と Color を設定します。

色のリソースを定義する例

```
<Window.Resources>
    <SolidColorBrush x:Key="numbersColor" Color="#666666"/>
    <SolidColorBrush x:Key="operatorsColor" Color="Green"/>
</Window.Resources>
```

リソースを使う側の書き方

リソースを使う側は StaticResource キーワードで、リソースの Key を指定します。

リソースを使う側の書き方

```
Background="{StaticResource operatorsColor}“
```

色の設定方法

ちなみに背景色や文字の色を設定する際は、Background と Foreground プロパティを設定します。

色の設定例

```
Background="DarkBlue"
Foreground="White"
```

サンプルコード

次のサンプルは、Window.Resources で背景色用と文字色用の 2 つのリソースを定義し、画

面の３つのボタンに対して、それらのリソースを，StaticResource キーワードを使用して設定しています。SolidColorBrush の Color を変更することで、一度に背景色や文字色を変更することが可能になります。

リソース設定例

```
<Window x:Class="WPF006.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF006"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">

    <Window.Resources>
        <SolidColorBrush x:Key="ButtonBackColor" Color="DarkBlue"/>
        <SolidColorBrush x:Key="ButtonForeColor" Color="White"/>
    </Window.Resources>

    <Grid>
        <StackPanel>
            <Button Content="AAA"
                    FontSize="40"
                    Background="{StaticResource ButtonBackColor}"
                    Foreground="{StaticResource ButtonForeColor}"/>
            <Button Content="BBB"
                    FontSize="40"
                    Background="{StaticResource ButtonBackColor}"
                    Foreground="{StaticResource ButtonForeColor}"/>
            <Button Content="CCC"
                    FontSize="40"
                    Background="{StaticResource ButtonBackColor}"
                    Foreground="{StaticResource ButtonForeColor}"/>
        </StackPanel>
    </Grid>
</Window>
```

この Xaml を記述したときの画面レイアウトは次のようになります。

複数の画面で定義を共有したいとき

Window.Resources を定義すると、複数の UI 要素に同じ定義を反映することが可能ですが、他の画面からは参照できないため、同じ定義を複数の画面で共有したい場合は問題が生じます。各画面で Window.Resources を定義してしまうと、背景色に関する定義が、画面ごとに記述されてしまい、アプリケーション全体で統一した色設定ができなくなってしまいます。

その場合は WPF のプロジェクトを作成したときに自動生成される App.xaml ファイルに記述することで、プロジェクト内のすべての画面から参照することが可能です。

App.xaml ファイルの Application.Resources エリアにリソースを定義します。

Application.Resources のリソース定義例

```
<Application.Resources>
        <SolidColorBrush x:Key= “backgroundColor” Color Orange ”/>
        <SolidColorBrush x:Key=”foregroundColor” Color=”White”/>
</Application.Resources>
```

サンプルコード

サンプルコードでは、MainWindow と Window 1 という 2 つの画面から App.xaml の Application.Resources のリソースを参照することで、共通のリソースを参照しています。

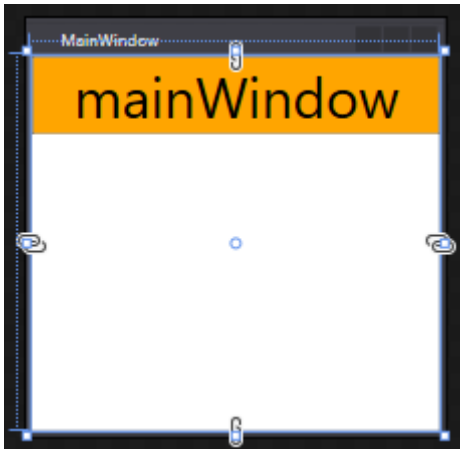
App.xaml

```
<Application x:Class="WPF006_2.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:WPF006_2"
             StartupUri="MainWindow.xaml">
```

```
    <Application.Resources>
        <SolidColorBrush x:Key="backColor" Color="Orange"/>

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

MainWindow.xaml

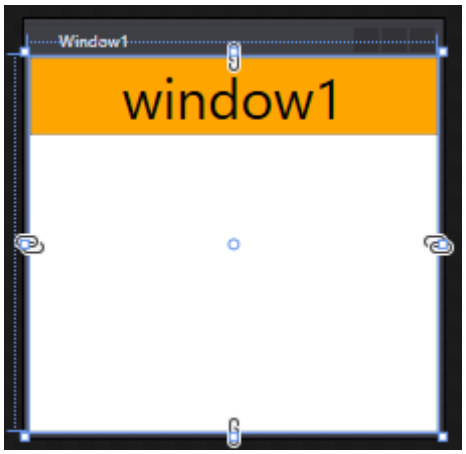
```
<Window x:Class="WPF006_2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF006_2"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">

    <Window.Resources>
    </Window.Resources>
    <Grid>
        <StackPanel>
            <Button Content="mainWindow"
                    FontSize="40"
                    Background="{StaticResource backColor}"
                    />
        </StackPanel>
    </Grid>
</Window>
```



Window1.xaml

```
<Window x:Class="WPF006_2.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF006_2"
        mc:Ignorable="d"
        Title="Window1" Height="300" Width="300">
```

```
    <Window.Resources>
    </Window.Resources>
    <Grid>
        <Grid>
            <StackPanel>
                <Button Content="window1"
                    FontSize="40"
                    Background="{StaticResource backColor}"
                    />
            </StackPanel>
        </Grid>
    </Grid>
</Window>
```



#7　コントロールのスタイル定義

スタイル定義とは？

WPF ではコントロールごとにデフォルトのスタイルを定義することができます。例えばラベルの背景色は「青」、文字の色は「白」という風に決めておくと、どの画面でラベルを生成しても、同じ背景色や文字の色で表示されることになり、アプリケーション全体に統一感を持たすことができます。

スタイル定義の書き方

WPF のプロジェクトを作成したときに、自動で生成される App.xaml ファイルの Application.Resources エリアに、Style TargetType="Label"という感じで記述します。"Label"の部分は、スタイルを設定したコントロール名を指定します。そのあとに Setter Property でプロパティ名を指定し、Value=に値を指定します。

Style を指定する例

```
<Application.Resources>
    <Style TargetType="Label">
        <Setter Property="Background" Value="DarkBlue"/>
    </Style>
</Application.Resources>
```

個別で異なる設定をしたい場合

Style で指定した値はデフォルト値になるので、画面に指定するコントロールに特に指定しなければ、デフォルト値で動作しますが、場合によっては、個別で背景色などを変更したい場合などがよくあります。そういった場合は、Xaml でコントロールを指定するときに背景色などを指定すれば、上書きでプロパティの値が設定されるため、個別での異なる設定が可能です。

サンプルコード

次の例では Label を 3 個、Button を 3 個設置し、Application.Resources のスタイル設定がどのように機能しているかを確認できます。

App.xaml

```
<Application x:Class="WPF007.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:WPF007"
             StartupUri="MainWindow.xaml">
```

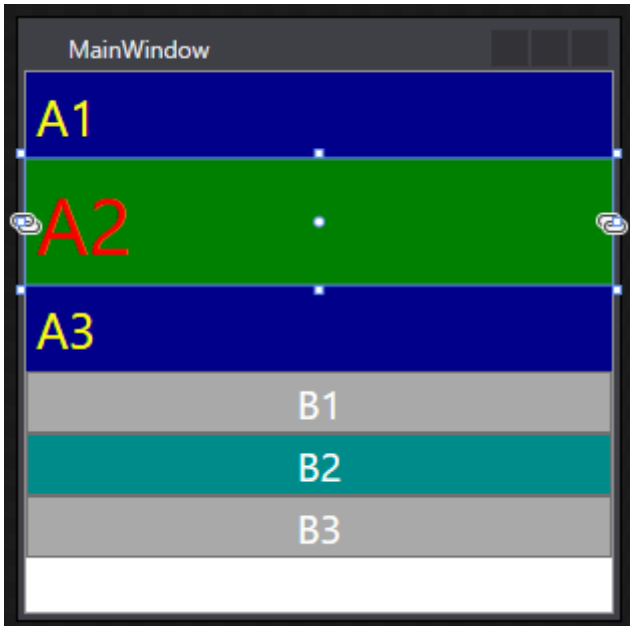
```
    <Application.Resources>
        <Style TargetType="Label">
            <Setter Property="Background" Value="DarkBlue"/>
            <Setter Property="Foreground" Value="Yellow"/>
            <Setter Property="FontSize" Value="25"/>
        </Style>

        <Style TargetType="Button">
            <Setter Property="Background" Value="DarkGray"/>
            <Setter Property="Foreground" Value="White"/>
            <Setter Property="FontSize" Value="20"/>
        </Style>
    </Application.Resources>
</Application>
```

Label 用のスタイルと Button 用のスタイルを作成し、それぞれ背景色、文字色、フォントサイズを指定しています。

MainWindow.xaml

```
<Window x:Class="WPF007.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF007"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel>
            <Label Content="A1"/>
            <Label Content="A2"
                   Background="Green"
                   Foreground="Red"
                   FontSize="40"/>
            <Label Content="A3"/>
            <Button Content="B1"/>
            <Button Content="B2"
                    Background="DarkCyan"/>
            <Button Content="B3"/>
        </StackPanel>
    </Grid>
</Window>
```

Label と Button を 3 個設置していますが、背景色、文字色、フォントサイズを指定していないコントロールは、Application.Resources で設定したそれぞれの値になり、背景色などを指定したコントロールは、それぞれ指定した値に上書きされていることが確認できます。


MainWindow
A1
A2
A3
B1
B2
B3

#8　グループごとのコントロールのスタイル定義の方法

グループごとのコントロールスタイルとは？

Style TargetType="Button"という感じで Application.Resources にスタイルを定義すると、画面の Xaml で何も指定しない場合、すべては Application.Resources に定義されたボタンのレイアウトになってしまいます。しかし、実際の運用ではすべてのボタンスタイルを同じにするだけでなく、異なるスタイルをいくつか使用したい場合がよくあります。

例えば出荷に関連するボタンはオレンジ、受注に関するボタンは赤などとレイアウトを分けたい場合は、Button 共通のスタイル 1 つだけでは、ほとんどのボタンで、背景色などの上書き設定を行う必要が出てきます。

そこで、共通の Button 等に対しての設定に加えて、ある特定のグループごとに、スタイルを設定できるようになっています。

グループごとのスタイルの書き方

グループごとにスタイルを設定する場合は、Style TargetType=”Button”という記述に加えて、Key を設定します。

グループごとのスタイルの定義例

```
<Style TargetType="Button" x:Key="AButtonStyle">
    <Setter Property="Background" Value="Green"/>
</Style>
```

各画面の Xaml 側では、次のようにして、定義したスタイルを指定します。

Xaml で指定する例

```
Style="{StaticResource AButtonStyle}"
```

スタイルの継承

スタイルをいくつか定義する場合、ほとんどのプロパティ設定は同じなのに、一部の設定だけが異なるという事があります。そのために毎回すべてのスタイル定義にすべてのプロパティ設定を書くのは手間であり、コーディングミスの元になるため、そういった場合は、元となるスタイルを継承し、異なる箇所のみを、上書きで設定することが可能です。スタイルを継承させる場合は BasedOn というキーワードを使います。

BasedOn キーワードで継承する例

```
<Style TargetType="Button" x:Key="BButtonStyle" BasedOn="{StaticResource AButtonStyle}">
    <Setter Property="Background" Value="Orange"/>
</Style>
```

BasedOn キーワードで AButtonStyle を継承し、その後に Setter Property で、必要な部分の変更を行っています。

サンプルコード

Style TargetType="Button"は、デフォルトのボタンスタイルになります。画面側で何も指定しないボタンは、この設定になります。

Style TargetType="Button" x:Key="AButtonStyle"を画面側で指定すると、AButtonStyle の定義でボタンレイアウトが表示されます。

BButtonStyle は AButtonStyle を継承し、背景色のみ赤に変更しています。背景色以外は AButtonStyle のプロパティ設定になります。

App.xaml

```
<Application x:Class="WPF008.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:WPF008"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <Style TargetType="Button">
            <Setter Property="Background" Value="DarkGray"/>
            <Setter Property="Foreground" Value="Black"/>
            <Setter Property="FontSize" Value="25"/>
            <Setter Property="Margin" Value="5"/>
        </Style>

        <Style TargetType="Button" x:Key="AButtonStyle">
            <Setter Property="Background" Value="Green"/>
            <Setter Property="Foreground" Value="White"/>
            <Setter Property="FontSize" Value="25"/>
            <Setter Property="Margin" Value="5"/>
        </Style>

        <Style     TargetType="Button"     x:Key="BButtonStyle"     BasedOn="{StaticResource
AButtonStyle}">
            <Setter Property="Background" Value="Red"/>
        </Style>
    </Application.Resources>
</Application>
```

MainWindow.xaml

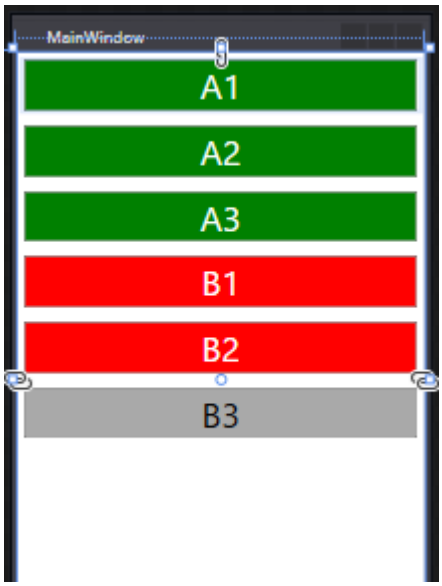
```
<Window x:Class="WPF008.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF008"
        mc:Ignorable="d"
        Title="MainWindow" Height="500" Width="300">
    <Grid>
        <StackPanel>
            <Button Content="A1"
                    Style="{StaticResource AButtonStyle}"/>
            <Button Content="A2"
                    Style="{StaticResource AButtonStyle}"/>
            <Button Content="A3"
                    Style="{StaticResource AButtonStyle}"/>
            <Button Content="B1"
                     Style="{StaticResource BButtonStyle}"/>
            <Button Content="B2"
                    Style="{StaticResource BButtonStyle}"/>
            <Button Content="B3"/>
        </StackPanel>
    </Grid>
</Window>
```

A1、A2、A3 ボタンは AButtonStyle になります。

B1、B2 ボタンは BButtonStyle になります。

B3 ボタンはなにも指定していないため、Style TargetType="Button"の定義になります。



#9　SQLite の使い方

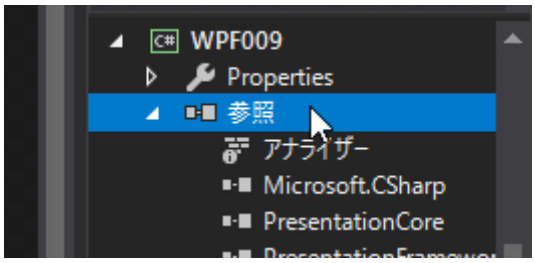
SQLite とは？

SQLite とはフリーで提供されている軽量なデータベースツールです。通常 Oracle や SQLServer であれば、ソフトのインストールや設定が多くあり、バックアップをとるのも結構大変ですが、SQLite は 1 ファイルで完結しているため、バックアップをとるにしても、普通にファイルをコピーするだけで完結します。最近ではスマホアプリを作成する場合に、スマホ端末自体にデータを保存したい場合などに適しています。また、デスクトップアプリでも、自分の端末でのみ使うような小規模なデータベースとして使うのに適しています。今回のようなアプリケーションの勉強用に使うデータベースとしては、大変適しているといえます。

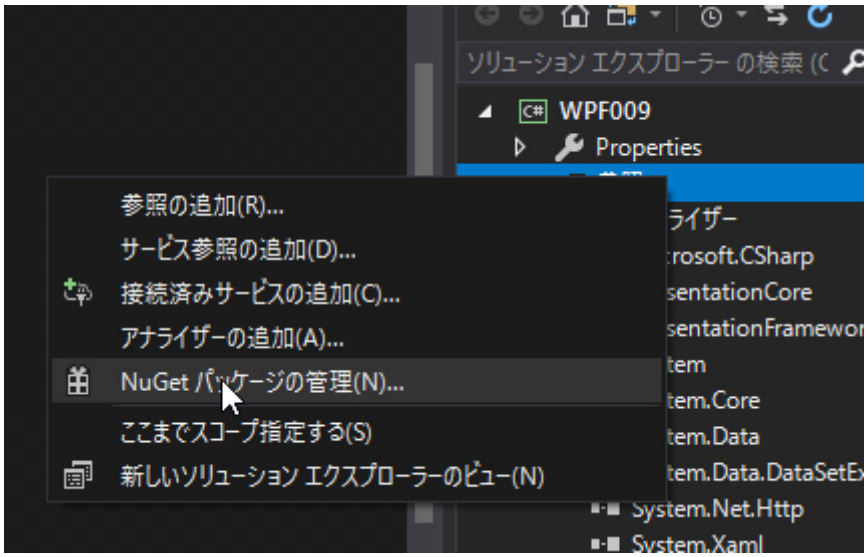
SQLite の導入

SQLite を WPF で使用するには、WPF のプロジェクトに NuGet で SQLite をインストールする必要があります。

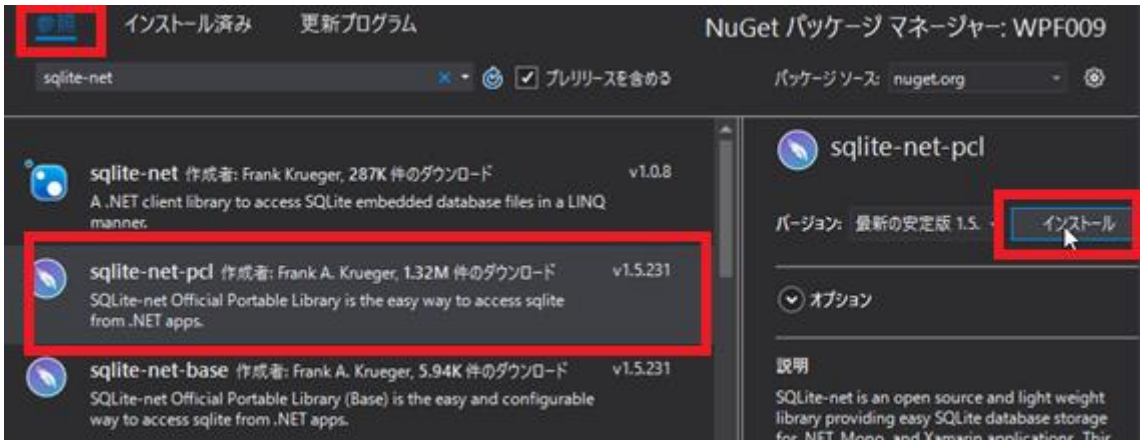
プロジェクトの「参照」を右クリックして



NuGet パッケージの管理を選択します。



NuGet マネージャーの「参照」を選択し、検索窓に「sqlite-net」と入力します。
Sqlite-net-pcl を選択し、バージョンが最新の安定版に選択されていることを確認して「インストール」ボタンを押下し、インストールを行います。



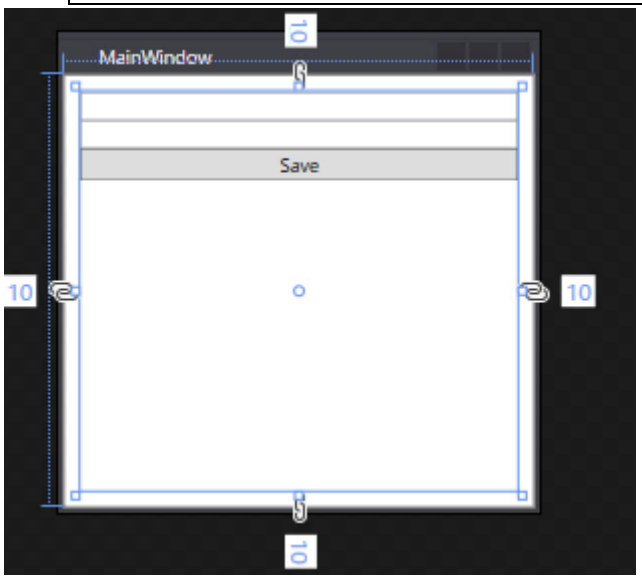
WPF の画面の作成

それでは SQLite のデータ登録とデータ取得のプログラムを作成していきます。はじめに、WPF の画面を作成します。今回は顧客管理システムの一部を作成する想定で行うので、顧客の名前と電話番号を登録する画面を作成します。WPF プロジェクトを作成したときに自動で生成される MainWindow.xaml の Xaml 入力エリアに次のように定義します。StackPanel にテキストボックスを 2 個、ボタンを 1 個置きます。テキストボックスはそれぞれ顧客の名前と電話番号の入力エリアです。ボタンは Save ボタンです。このボタンを押下すると、

SQLite の Customer テーブルにデータを登録する仕様とします。

MainWindow.xaml

```
<Window x:Class="WPF009.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF009"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel Margin="10">
            <TextBox x:Name="NameTextBox"/>
            <TextBox x:Name="PhoneTextBox"/>
            <Button Content="Save"
                    x:Name="SaveButton"
                    Click="SaveButton_Click"/>
        </StackPanel>
    </Grid>
</Window>
```



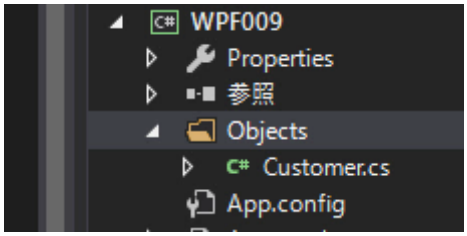
コードファーストによる SQLite の操作

今回は SQLite の操作はすべてコードファーストで行います。コードファーストとは、通常データベースには、テーブルの定義と作成等を行ってから、C#からデータの登録処理等を行いますが、コードファーストでは、テーブルの定義、テーブルの作成、データの作成、データ検索のすべてを C#のプログラムコードで行います。

テーブルとなるクラスを作成

コードファーストで作成するため、SQLite のテーブルに定義を C#のクラスとして作成します。今回は顧客管理システムの一部を作成するサンプルとして、顧客テーブルにあたる「Customer」クラスを作成します。

WPF のプロジェクトに新規のフォルダーを追加し、「Objects」フォルダーとします。その中に、Customer クラスを作成します。



Customer クラス

```
using SQLite;

namespace WPF009.Objects
{
    public class Customer
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }

        /// <summary>
        /// 名前
        /// </summary>
        public string Name { get; set; }

        /// <summary>
        /// 電話番号
        /// </summary>
        public string Phone { get; set; }
    }
}
```

Customer クラスに顧客の名前と電話番号を格納する string のプロパティを作成します。最初の int の Id は画面レイアウトには存在していませんでしたが、Customer テーブルのプライマリキーとして定義しています。プライマリキーとは、そのテーブルの中に重複した値が存在しない列という意味です。これで Customer テーブルに Id の重複したデータは存在できないという定義をしたことになります。AutoIncrement の定義は、オードナンバーの定義です。データが登録されるたびに、１から順番の連番が自動的に Id として割り当てられます。これで Customer クラスの定義は完了です。

Save ボタンクリックイベント

Save ボタンをクリックしたときに、SQLite に画面の顧客名と電話番号を登録するプログラムを記述していきます。

MainWindow.xaml のコードビハインド側

```
using SQLite;
using System;
using System.Windows;
using WPF009.Objects;

namespace WPF009
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void SaveButton_Click(object sender, RoutedEventArgs e)
        {
            var customer = new Customer()
            {
                Name = NameTextBox.Text,
                Phone = PhoneTextBox.Text,
            };

            string databaseName = "Shop.db";
            string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
            string databasePath = System.IO.Path.Combine(folderPath, databaseName);
            using (var connection = new SQLiteConnection(databasePath))
            {
                connection.CreateTable<Customer>();
                connection.Insert(customer);
            }
        }
    }
}
```

まず、var customer = new Customer()のところで、Cusutomer クラスを生成し、画面の顧客名と電話番号を設定しています。

その後の databaseName はデータベースの名前です。任意の名前でいいので今回は Shop.db としています。folderPath はデータベースファイルの場所です。今回はコードファーストでデータベースを作成するため、まだデータベースファイルは存在しません。後述の connection.CreateTable を最初に実行したときに自動生成されるので、任意の場所を指定し

ます。今回は Environment.SpecialFolder.MyDocuments で、自分の PC のマイドキュメントを指定しています。

SQLiteConnection で SQLite と接続します。このクラスを使用するためにクラスに最初に記述する using に SQLite を追加します。SQLiteConnection を生成するためには、データベースのパスが必要になります。前述した databasePath を指定します。
ちなみに SQLiteConnection は Disopose()メソッドが存在することからもわかるように IDisposable インタフェースが定義されています。クラス生成後はメモリの解放が必要なため Disopose()、もしくは Close()を呼び出す必要があります。しかし今回はメソッド内で SQLiteConnection の使用が完結するため using キーワードを使って解放処理を行っています。

「connection.CreateTable<Customer>();」をすることで、データベースや、Cusutomer テーブルが存在しない場合でもエラーにならず、空のテーブルを作成してくれます。存在している場合は何もしないため、このコードを記述しておくことで、初回もエラーにならずに、テーブルが自動生成されます。

「connection.Insert(customer);」で実際に Customer テーブルに customer インスタンスの値がインサートされます。

登録処理は以上です。ただ、これだけだと、実際に登録されたのかを確認できないため、データを取得する処理を、続けて作成していきます。

データの取得処理

MainWindow.xaml に ReadButton を追加し、Click イベントを作成します。

MainWindow.xaml

```
<Window x:Class="WPF009.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF009"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel Margin="10">
            <TextBox x:Name="NameTextBox"/>
            <TextBox x:Name="PhoneTextBox"/>
            <Button Content="Save"
                    x:Name="SaveButton"
```

```
                    Click="SaveButton_Click"/>
            <Button Content="Read"
                    x:Name="ReadButton"
                    Click="ReadButton_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

データベースの接続先は各画面から参照する必要があるためApp.xamlのコードビハインド側にデータベースパスの設定を移動し、共通で使えるように変更しました。

App.xaml のコードビハインド側

```
using System;
using System.Windows;

namespace WPF009
{
    /// <summary>
    /// App.xaml の相互作用ロジック
    /// </summary>
    public partial class App : Application
    {
        static string databaseName = "Shop.db";
        static string folderPath =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
        public static string DatabasePath = System.IO.Path.Combine(folderPath,
databaseName);
    }
}
```

コードビハインド側に ReadButton_Click を追加

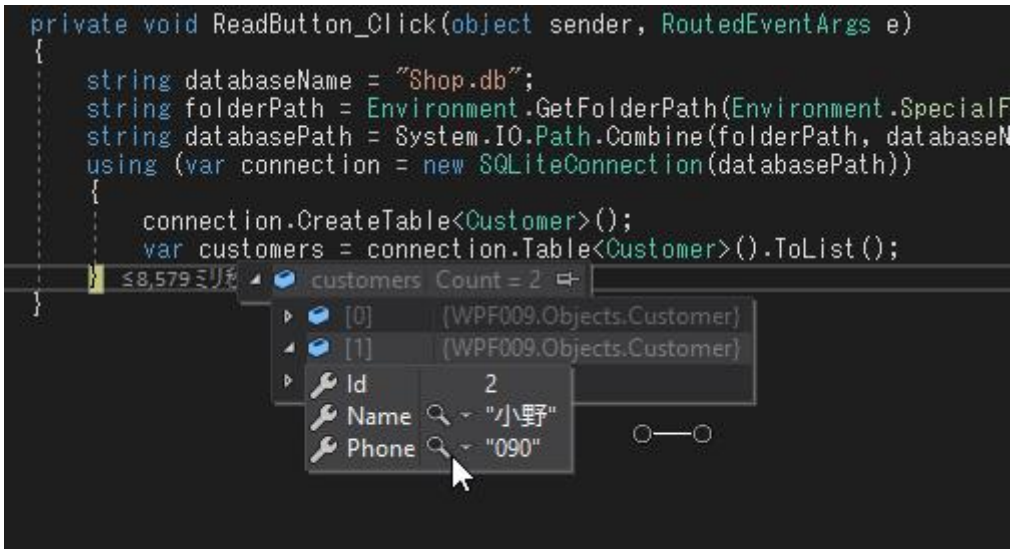
MainWindow のコードビハインド側

```
using SQLite;
using System;
using System.Windows;
using WPF009.Objects;

namespace WPF009
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {

        public MainWindow()
        {
            InitializeComponent();
        }

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

```
            var customer = new Customer()
            {
                Name = NameTextBox.Text,
                Phone = PhoneTextBox.Text,
            };

            using (var connection = new SQLiteConnection(App.DatabasePath))
            {
                connection.CreateTable<Customer>();
                connection.Insert(customer);
            }
        }

        private void ReadButton_Click(object sender, RoutedEventArgs e)
        {
            using (var connection = new SQLiteConnection(App.DatabasePath))
            {
                connection.CreateTable<Customer>();
                var customers = connection.Table<Customer>().ToList();
            }
        }
    }
}
```

Save クリックのときと同じように SQLiteConnection を生成しテーブルがないときのために CreateTable を実行しています。

「var customers = connection.Table<Customer>().ToList();」で SQLite の Customer テーブルから全件を取得し、Customer クラスのリストを取得して customers に代入しています。ブレークポイントを設置して、ウォッチで中を見れば、Save で保存したデータが確認できます。また Id には、連番が設定されていることも確認してください。



#10　ListView

ListView とは？

ListView はデータの一覧を表示するコントロールで、自由なレイアウトで一覧表を作ることができます。

ListView の使い方

ListView の ItemsSource プロパティにカスタムクラスのリスト等をセットすることで、一覧表が表示されます。

今回は顧客クラスである「Customer」クラスを作成し、ID、名前、電話番号を一覧表示する例を見ていきましょう。

ItemsSource プロパティでの一覧表示

Xaml

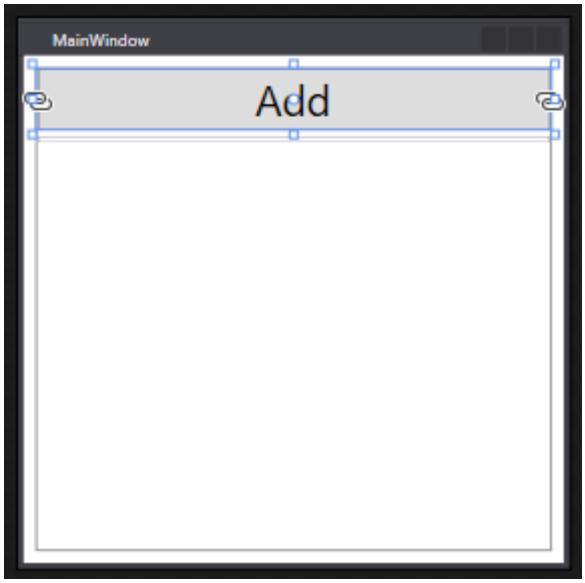
それではまず画面レイアウトを作成します。画面レイアウトにはデータ追加用の Add ボタンと ListView を設置します。Add ボタンには Click イベントを生成しておきます。

MainWindow.xaml

```
<Window x:Class="WPF010.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF010"
        mc:Ignorable="d"
        Title="MainWindow" Height="400" Width="400">
    <Grid>
        <StackPanel Margin="10">
            <Button x:Name="AddButton"
                    Content="Add"
                    FontSize="30"
                    Click="AddButton_Click"/>

            <ListView x:Name="CustomerListView"
                      Margin="0,5,0,0">

            </ListView>

        </StackPanel>
    </Grid>
```

```
</Window>
```

＜Xaml で生成される画面イメージ＞



Customer クラスの作成

顧客クラスの一覧を表示するために、1 データに相当するクラス「Customer」クラスを作成します。ID、名前、電話番号のプロパティがあるだけのシンプルなクラスです

Customer.cs

```
namespace WPF010
{
    public class Customer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Phone { get; set; }
    }
}
```

MainWindow のコードビハインド側

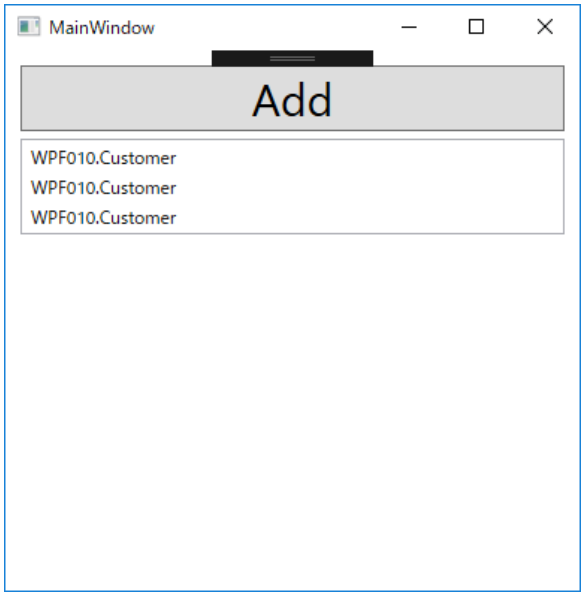
コードビハインド側では、Customer クラスのリストのフィールド作成し、コンストラクタでデータを 3 件追加しています。その後に ListView の ItemsSource プロパティに_customers インスタンスをセットしています。

MainWindow.xaml

```
using System.Collections.Generic;
using System.Windows;

namespace WPF010
{
    /// <summary>
```

```
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        private List<Customer> _customers = new List<Customer>();

        public MainWindow()
        {
            InitializeComponent();

            _customers.Add(new Customer { Id = 1, Name = "name1", Phone = "phone1" });
            _customers.Add(new Customer { Id = 2, Name = "name2", Phone = "phone2" });
            _customers.Add(new Customer { Id = 3, Name = "name3", Phone = "phone3" });

            CustomerListView.ItemsSource = _customers;
        }

        private void AddButton_Click(object sender, RoutedEventArgs e)
        {

        }
    }
}
```

＜実行結果＞



実行すると、データは 3 件表示されますが、「WPF010.Customer」と表示されてしまっています。これでは使いものになりません。これは Customer クラスの ToString()された文字列が表示されているためです。これの解決方法とみていきたいと思います。

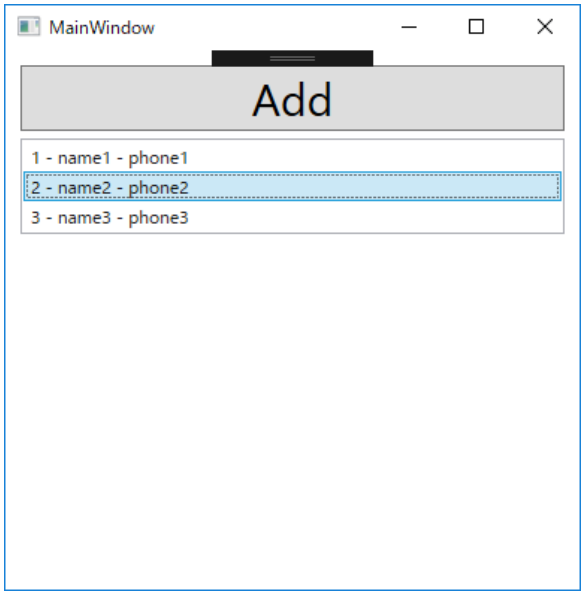
Customer クラスの ToString()問題の解決

次のように Customer クラスに ToString()メソッドをオーバーライドすることで、任意の文字列を表示することができます。この場合、ToString()されると、ID、名前、電話番号がハイフン区切りで文字列が生成されます。

Customer の ToString()をオーバーライド

```
namespace WPF010
{
    public class Customer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Phone { get; set; }

        public override string ToString()
        {
            return $"{Id} - {Name} - {Phone}";
        }
    }
}
```

＜実行結果＞



これで、データを表示することができました。ID、名前、電話番号がハイフン区切りで表示されています。しかし、これだと表現力が低すぎますよね？もっと自由にレイアウトしたいという要望が通常はあると思います。例えば、名前の文字の大きさを大きくして、電話番号は青文字で表示したいなどです。こういった問題に対する対応を TextBlock と Binding という機能を使って解決していきます。

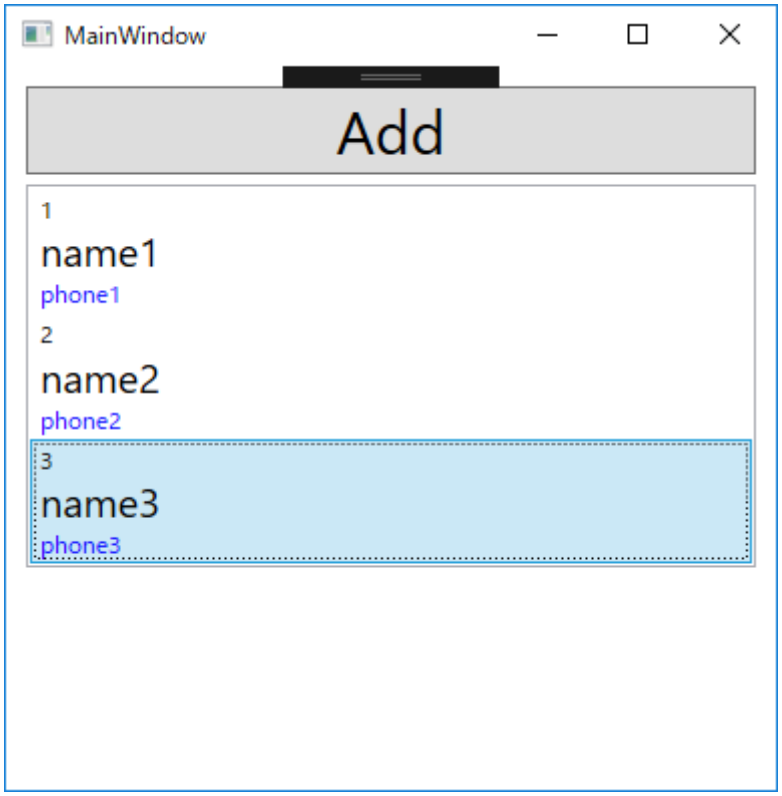
ItemTemplate と DataTemplate と Binding

ListView に対する Xaml を次のように定義します。まず ListView.ItemTemplate ブロックを記述し、その中に DataTemplate ブロックを作ります。その中に StackPanel で自由にコントロールを配置していきます。今回は ID、名前、電話番号の３つの表示ですから、それらのすべてを TextBlock で設定します。それぞれの TextBlock に何を表示するかを設定する必要があるので Text="{Binding Id}"のようにして、Customer クラスのどの値を表示するかを指定します。フォントのサイズや文字の色などを変更したい場合は、TextBlock のそれらのプロパティを変更するだけでよいので、自由なレイアウトを作成することができます。

MainWindow.xaml

```
<Window x:Class="WPF010.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF010"
        mc:Ignorable="d"
        Title="MainWindow" Height="400" Width="400">
    <Grid>
        <StackPanel Margin="10">
            <Button x:Name="AddButton"
                    Content="Add"
                    FontSize="30"
                    Click="AddButton_Click"/>

            <ListView x:Name="CustomerListView"
                      Margin="0,5,0,0">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <StackPanel>
                            <TextBlock Text="{Binding Id}"/>
                            <TextBlock Text="{Binding Name}" FontSize="20"/>
                            <TextBlock Text="{Binding Phone}" Foreground="Blue"/>
                        </StackPanel>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>

        </StackPanel>
    </Grid>
</Window>
```

＜実行結果＞



これで自由にレイアウトすることができるようになりました。

Add ボタンの実装

_index フィールドの追加

Add ボタンが押されたときに、データが追加されていく実装を行います。MainWindow のコードビハインド側に自動生成されている AddButton_Click イベントに、ボタンを押すたびに、Customer クラスが追加されるように記述します。また、連番で追加されるように _index フィールドを作成して、コンストラクタで追加している 3 件のデータも一部変更しています。

ObservableCollection の利用

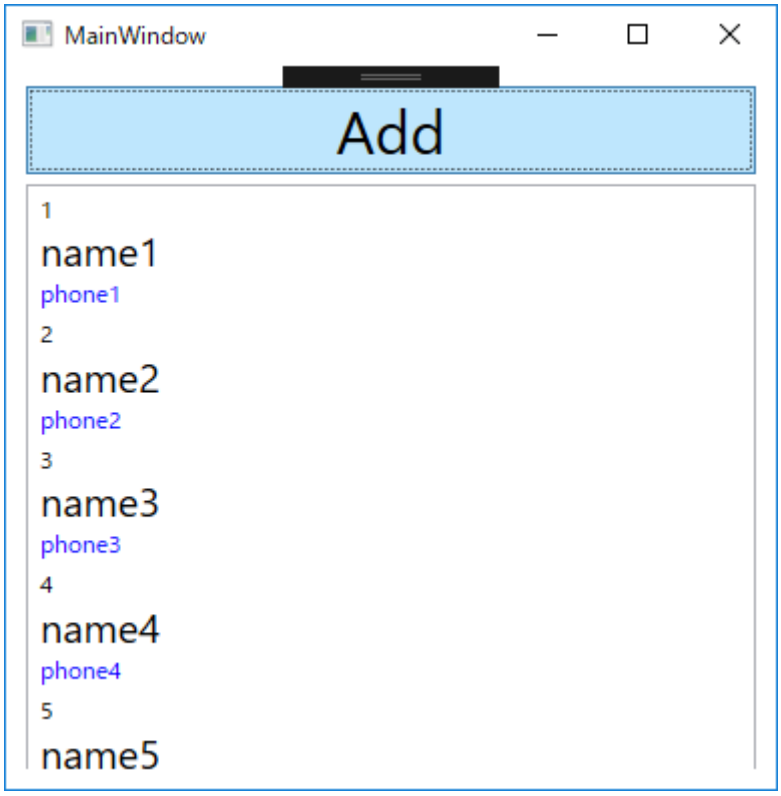
Customer クラスのリストのフィールドを List<T>で宣言していましたが、それでは Add ボタンをクリックしてデータを追加しても変更通知が行われず、画面が変化しないため、ObservableCollection に変更しています。このクラスを使用すると、データの変更通知が行われるため、画面が変更されます。

MainWindow のコードビハインド側

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Windows;

namespace WPF010
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        private ObservableCollection<Customer> _customers = new ObservableCollection<Customer>();
        private int _index = 0;
        public MainWindow()
        {
            InitializeComponent();

            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });

            CustomerListView.ItemsSource = _customers;
        }

        private void AddButton_Click(object sender, RoutedEventArgs e)
        {
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
```

```
            CustomerListView.ItemsSource = _customers;
        }
    }
}
```

＜実行結果＞



Add ボタンを押下するたびにデータが追加されるようになりました。しかし、スクロールバーが出てきませんよね？これに対しては、ListView の Height プロパティを設定することで解決できます。

スクロールバーが出ない問題

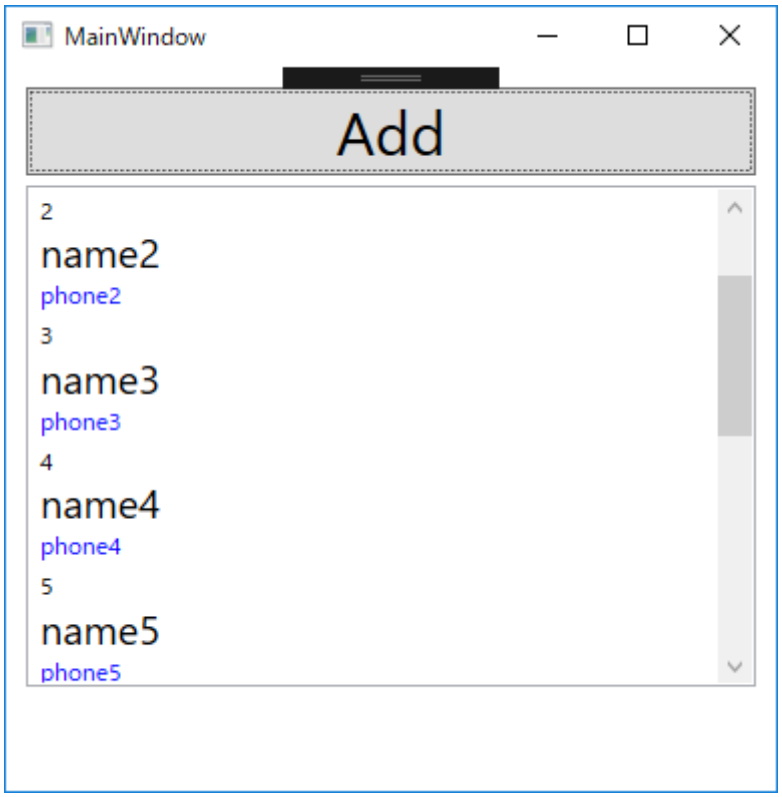
ListView の Height プロパティを設定することでスクロールバーが表示されるようになります。

Xaml

```
<Window x:Class="WPF010.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF010"
        mc:Ignorable="d"
        Title="MainWindow" Height="400" Width="400">
    <Grid>
```

```
        <StackPanel Margin="10">
            <Button x:Name="AddButton"
                    Content="Add"
                    FontSize="30"
                    Click="AddButton_Click"/>

            <ListView x:Name="CustomerListView"
                      Margin="0,5,0,0"
                      Height="250">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <StackPanel>
                            <TextBlock Text="{Binding Id}"/>
                            <TextBlock Text="{Binding Name}" FontSize="20"/>
                            <TextBlock Text="{Binding Phone}" Foreground="Blue"/>
                        </StackPanel>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>

        </StackPanel>
    </Grid>
</Window>
```

＜実行結果＞



#11　ListView のフィルタリング

ListView のフィルタリングとは？

ListView にすべての行が表示されていると不都合が生じることがあります。例えば、一覧にリストされている項目が多い場合は、特定の行を探すのに手間がかかってしまいます。その場合、任意の文字などでフィルタリングをすることで、必要な情報だけが表示され、非常に使いやすくなります。

サンプルコード

次の例では、Customer クラスの一覧を表示している ListView に検索用テキストボックスを設置し、その検索用テキストボックスに入力された文字を Customer の Name に含んでいる行のみが表示されるようにフィルタリングをしています。フィルタリングは Linq 機能の Where を使用しています。

Customer.cs

```
namespace WPF011
{
    public class Customer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Phone { get; set; }

        public override string ToString()
        {
            return $"{Id} - {Name} - {Phone}";
        }
    }
}
```

MainWindow.xaml

```
<Window x:Class="WPF011.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF011"
        mc:Ignorable="d"
        Title="MainWindow" Height="400" Width="400">
    <Grid>
        <StackPanel Margin="10">
            <Button x:Name="AddButton"
                    Content="Add"
                    FontSize="30"
```

```
                    Click="AddButton_Click"/>

            <TextBox x:Name="SearchTextBox"
                     FontSize="20"
                     TextChanged="SearchTextBox_TextChanged"/>

            <ListView x:Name="CustomerListView"
                      Margin="0,5,0,0"
                      Height="250">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <StackPanel>
                            <TextBlock Text="{Binding Id}"/>
                            <TextBlock Text="{Binding Name}" FontSize="20"/>
                            <TextBlock Text="{Binding Phone}" Foreground="Blue"/>
                        </StackPanel>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>

        </StackPanel>
    </Grid>
</Window>
```

SearchTextBox を追加して、Click イベント SearchTextBox_TextChanged を生成しています。

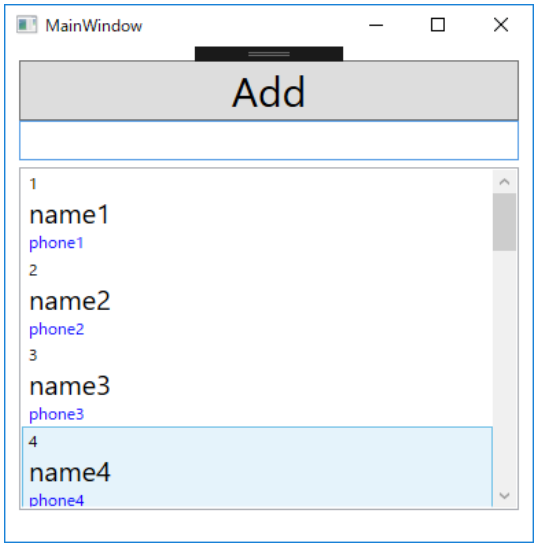
MainWindow.xaml のコードビハインド側

```
using System.Collections.ObjectModel;
using System.Linq;
using System.Windows;
using System.Windows.Controls;

namespace WPF011
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        private ObservableCollection<Customer> _customers = new
ObservableCollection<Customer>();
        private int _index = 0;
        public MainWindow()
        {
            InitializeComponent();

            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
```

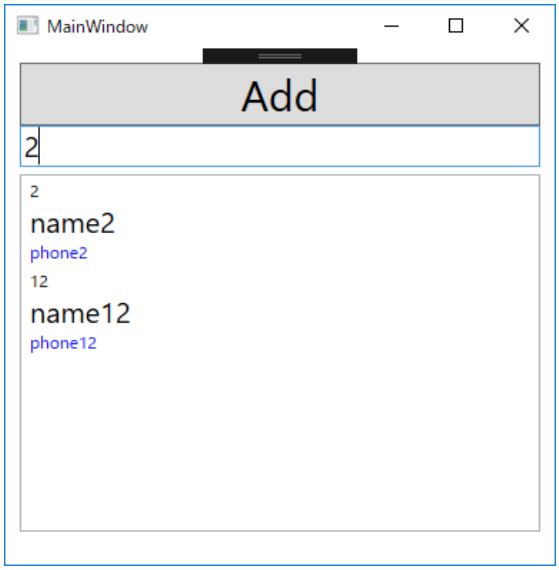
```
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });
            _customers.Add(
new Customer { Id = ++_index, Name = "name" + _index, Phone = "phone" + _index });

            CustomerListView.ItemsSource = _customers;
        }

        private void AddButton_Click(object sender, RoutedEventArgs e)
        {
            _customers.Add(new Customer { Id = ++_index, Name = "name" + _index, Phone =
"phone" + _index });
            //CustomerListView.ItemsSource = _customers;
        }

        private void SearchTextBox_TextChanged(object sender, TextChangedEventArgs e)
        {
            var filterList =
                _customers.Where(x => x.Name.Contains(SearchTextBox.Text)).ToList();
            CustomerListView.ItemsSource = filterList;
        }
    }
}
```

コンストラクタで、_customers インスタンスに 15 件の Customer クラスのインスタンスを追加しています。SearchTextBox_TextChanged では Linq 機能を使用して SearchTextBox の Text プロパティの文字列を含んでいる行のみを取得し、ListView の ItemsSource に設定しています。

＜フィルタリングなし＞



＜文字列「２」でフィルタリングした状態＞



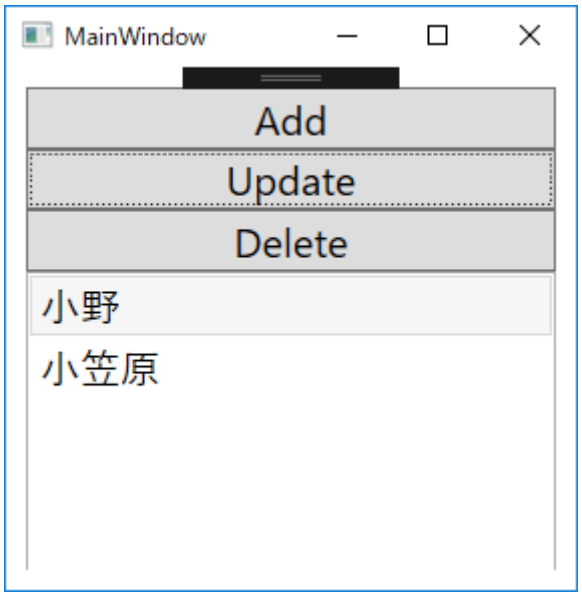
#12　SQLite＆ListView

これまでの知識のまとめ

ListView を学んだので、SQLite と組み合わせることで、ある程度のアプリケーションを作ることができます。これまで学んだ知識のまとめとして、マスター設定画面の作成をしてみましょう。腕試しとして、次の仕様だけを見てご自身で一度プログラミングしてみることをお勧めします。どうしてもわからない場合は解説を見ながら一緒にコーディングしてみてください。ただ、データの更新と削除はまだ解説していないので、解説を見てください。

仕様

＜一覧画面＞



Add ボタン	Save 画面の起動
Update ボタン	Save 画面の起動
Delete ボタン	ListView で選択されている Customer テーブルの行削除
一覧（ListView）	Customer テーブルの一覧を表示

＜Save 画面＞

テキストボックス	Customer の名前を表示 Add モードのときは空文字 Update モードのときは呼び出し元画面で指定した名前
Save ボタン	Add モードのときは Customer の新規追加 Update モードのときは Customer の行更新

Customer テーブル（SQLite）

Id	ID
Name	名前

実装例

App.xaml

```
using System;
using System.Windows;

namespace WPF012
{
    /// <summary>
    /// App.xaml の相互作用ロジック
    /// </summary>
    public partial class App : Application
    {
        static string databaseName = "Shop2.db";
        static string folderPath =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
        public static string DatabasePath = System.IO.Path.Combine(folderPath, databaseName);
```

```
    }
}
```

App.xaml には SQLite の接続先を定義します。場所は任意の場所で問題ありません。

Customer クラス

```
using SQLite;

namespace WPF012
{
    public class Customer
    {
        public Customer()
        {

        }

        public Customer(string name)
        {
            Name = name;
        }

        public Customer(int id ,string name)
        {
            Id = id;
            Name = name;
        }

        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

Customer クラスは Id と Name のみを定義します。コンストラクタは用途別に 3 種類定義しています。コンストラクタは記述しなくても問題はありません。インスタンスを生成時に値を指定するほうが, コードがわかりやすくなるためそうしています。

MainWindow.xaml

```
<Window x:Class="WPF012.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF012"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel Margin="10">
            <Button x:Name="AddButton"
                    FontSize="20"
                    Content="Add"
                    Click="AddButton_Click"/>
```

```
            <Button x:Name="UpdateButton"
                    FontSize="20"
                    Content="Update"
                    Click="UpdateButton_Click"/>
            <Button x:Name="DeleteButton"
                    FontSize="20"
                    Content="Delete"
                    Click="DeleteButton_Click"/>
            <ListView Height="150"
                      x:Name="CustomerListView">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <TextBlock Text="{Binding Name}"
                                   FontSize="20"/>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackPanel>
    </Grid>
</Window>
```

MainWindow では StackPanel 上に、Add、Update、Delete ボタンと ListView を設置しています。

MainWindow.xaml　コードビハインド側

```
using SQLite;
using System.Windows;

namespace WPF012
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            ReadDatabase();
        }

        private void AddButton_Click(object sender, RoutedEventArgs e)
        {
            var f = new SaveWindow(null);
            f.ShowDialog();
            ReadDatabase();
        }

        private void UpdateButton_Click(object sender, RoutedEventArgs e)
        {
            var item = CustomerListView.SelectedItem as Customer;
            if(item == null)
            {
                MessageBox.Show("行を選択してください");
```

```
                return;
            }

            var f = new SaveWindow(item);
            f.ShowDialog();
            ReadDatabase();
        }

        private void DeleteButton_Click(object sender, RoutedEventArgs e)
        {
            var item = CustomerListView.SelectedItem as Customer;
            if (item == null)
            {
                MessageBox.Show("行を選択してください");
                return;
            }

            using (var connection = new SQLiteConnection(App.DatabasePath))
            {
                connection.CreateTable<Customer>();
                connection.Delete(item);
                ReadDatabase();
            }
        }

        private void ReadDatabase()
        {
            using (var connection = new SQLiteConnection(App.DatabasePath))
            {
                connection.CreateTable<Customer>();
                CustomerListView.ItemsSource = connection.Table<Customer>().ToList();
            }
        }
    }
}
```

Add ボタン押下時は Customer を Null で Save 画面を起動。Update 時は選択されている行の Customer クラスを引数に渡しています。ListView の選択行は SelectedItem で取得します。選択されていないときは Null になるため、Null チェックも行っています。
Update ボタン押下時は選択行の Customer クラスを引数として Save 画面を起動しています。Delete ボタン押下時は選択行の Customer クラスを削除しています。削除は connection.Delete で実行しています。

SaveWindow.xaml

```
<Window x:Class="WPF012.SaveWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF012"
        mc:Ignorable="d"
        Title="SaveWindow" Height="300" Width="300">
```

```
    <Grid>
        <StackPanel Margin="10">
            <TextBox x:Name="NameTextBox"
                     FontSize="20"/>
            <Button Content="Save"
                    x:Name="SaveButton"
                    FontSize="20"
                    Click="SaveButton_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

Save 画面は名前のテキストボックスと、Save ボタンを設置しています。

SaveWindow.xaml　コードビハインド側

```
using SQLite;
using System.Windows;

namespace WPF012
{
    /// <summary>
    /// SaveWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class SaveWindow : Window
    {
        private Customer _customer;
        public SaveWindow(Customer customer)
        {
            InitializeComponent();

            _customer = customer;

            if(customer != null)
            {
                this.NameTextBox.Text = customer.Name;
            }
        }

        private void SaveButton_Click(object sender, RoutedEventArgs e)
        {
            if(NameTextBox.Text.Trim().Length < 1)
            {
                MessageBox.Show("名前を入力してください");
                return;
            }

            using (var connection = new SQLiteConnection(App.DatabasePath))
            {
                connection.CreateTable<Customer>();
                if(_customer == null)
                {
                    connection.Insert(new Customer(NameTextBox.Text));
                }
                else
```

```
                {
                    connection.Update(new Customer(_customer.Id, NameTextBox.Text));
                }

                Close();
            }
        }
    }
}
```

Save 画面のコンストラクタでは、Customer クラスを引数にしています。Null の場合は新規追加、Null 以外のときは、更新処理をしています。
更新処理は connection.Update で実行しています。

以上がサンプルコードとなります。

#13　ボタン

ボタンとは？

ボタンはこれまでも使ってきました。基本的にはクリックイベントを実装し、押されたときにデータを保存する処理などを記述します。ここでは、そのボタンの使い方の詳細と種類を見ていきます。

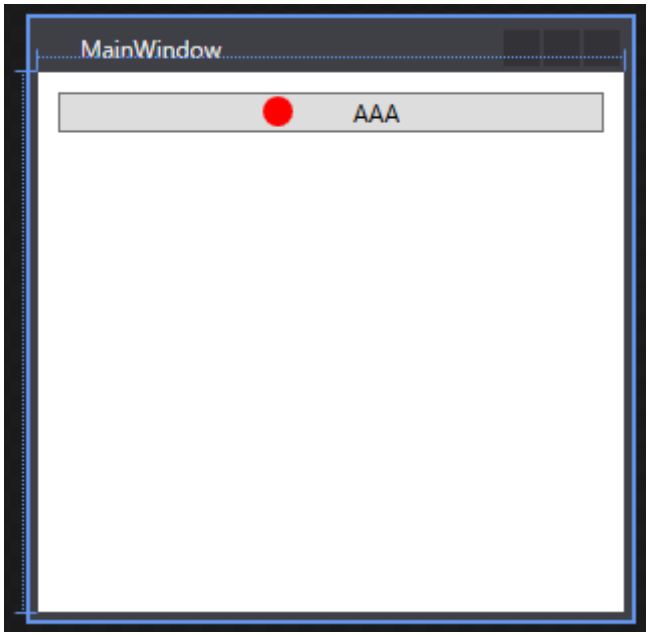
通常のボタン

通常のボタンは基本的に Content に「保存」などの文言を表示して、Click イベントを実装するという使い方になりますが、WindowsForms のボタンコントロールとは異なり、Content が文字だけでなく、StackPanel 等を設定できるため、ボタンの文言表示の場所に、画像と文字を並べて表示するという事が、簡単にできるようになりました。

ボタンの Content に StackPanel を設置し、画像とテキストをならべて表示する例

```
<Button Margin="10"
        x:Name="NormalButton"
        Click="NormalButton_Click">
    <Button.Content>
        <StackPanel Orientation="Horizontal">
            <Ellipse Fill="Red"
                     Width="15"
                     Height="15"
                     Margin="0,0,30,0"/>
            <TextBlock Text="AAA"/>
        </StackPanel>
    </Button.Content>
</Button>
```

この例では、通常<Button Content="AAA"/>と定義していた書き方をそうではなく、Button.Content のエリアを作り、その中に StackPanel 設置して、横向きに赤丸マークと TextBlock にテキストを”AAA”と表示しています。
この Xaml のレイアウトは次のようになります。



RepeatButton

RepeatButton はボタンを押下している間中、クリックイベントを通知するボタンです。RepeatButton を押下し続けると、最初に 1 回クリックイベントが通知され、その後、Delay に設定されているミリ秒間なにもせず待機されます。Delay ミリ秒が経過するとクリックイベントが通知され、その後は Interval に設定されている値ミリ秒間隔で、クリックイベントが通知され、ボタンの押し下げを止めると、クリックイベントも通知されなくなります。

Interval	クリックイベントが通知される間隔（ミリ秒）
Delay	リピートを開始するまでの待機時間（ミリ秒）

次の例で、RepeatButton を押し続けると、最初に 1 回クリックイベントが通知され、その 5 秒後から、2 秒間隔でクリックイベントが通知され続けます。

MainWindow.xaml

```
<RepeatButton Content="repeat"
              FontSize="20"
              x:Name="RepeatButton"
              Click="RepeatButton_Click"
              Interval="2000"
              Delay="5000"/>
```

MainWindow.xaml.cs

```
private void RepeatButton_Click(object sender, RoutedEventArgs e)
{
    Console.WriteLine(DateTime.Now.ToString("yyyy/MM/dd HH:mm:ss") + "RepeatButton");
}
```

ToggleButton

ToggleButton とは、ボタンを押し下げている状態と押し上げている（何もしていない）状態の 2 つの状態を表すボタンです。見た目はボタンですが、CheckBox のチェックありとチェックなしの状態のボタン版と考えていただいて OK です。

サンプルコード

次の例では、ボタンを押すたびに、MyToggleBtton.IsChecked が Ture と False に変化し、ボタンは True のときは押したままの描画となります。

MainWindow.xaml

```
<ToggleButton Content="toggle"
              FontSize="20"
              x:Name="MyToggleBtton"
              Click="MyToggleBtton_Click"/>
```

MainWindow.xaml.cs

```
private void MyToggleBtton_Click(object sender, RoutedEventArgs e)
{
    Console.WriteLine("toggle button click:" + MyToggleBtton.IsChecked);
}
```

各種ボタンサンプルコード全体

3 種類のボタンのサンプルコードをまとめます。

MainWindow.xaml

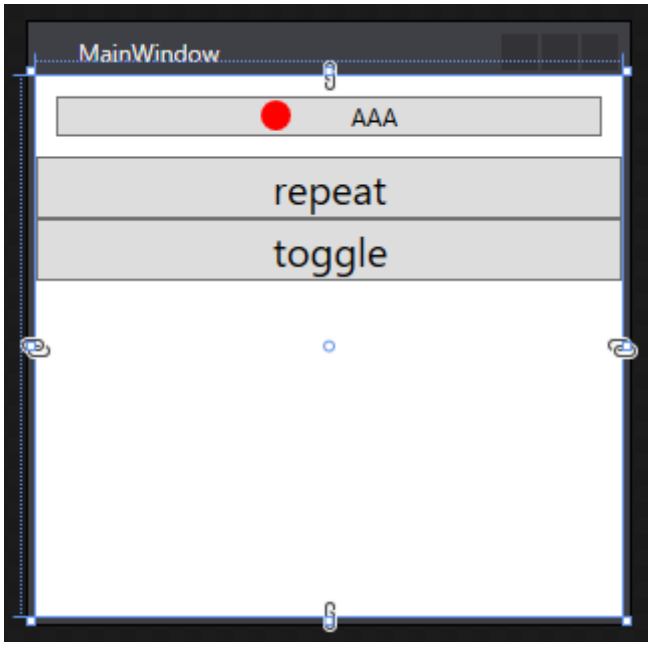
```
<Window x:Class="WPF013.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF013"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel>
            <Button Margin="10"
                    x:Name="NormalButton"
                    Click="NormalButton_Click">
                <Button.Content>
                    <StackPanel Orientation="Horizontal">
                        <Ellipse Fill="Red"
                                 Width="15"
                                 Height="15"
                                 Margin="0,0,30,0"/>
                        <TextBlock Text="AAA"/>
```

```
                    </StackPanel>
                </Button.Content>
            </Button>

            <RepeatButton Content="repeat"
                          FontSize="20"
                          x:Name="RepeatButton"
                          Click="RepeatButton_Click"
                          Interval="2000"
                          Delay="5000"/>

            <ToggleButton Content="toggle"
                          FontSize="20"
                          x:Name="MyToggleBtton"
                          Click="MyToggleBtton_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System;
using System.Windows;

namespace WPF013
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void NormalButton_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Normal button click!!");
        }

        private void RepeatButton_Click(object sender, RoutedEventArgs e)
        {
            Console.WriteLine(DateTime.Now.ToString("yyyy/MM/dd HH:mm:ss") + "RepeatButton");
        }

        private void MyToggleBtton_Click(object sender, RoutedEventArgs e)
        {
            Console.WriteLine("toggle button click:" + MyToggleBtton.IsChecked);
        }
    }
}
```

Xaml レイアウトイメージ



#14　CheckBox

CheckBox とは？

CheckBox（チェックボックス）とは、ON か OFF を選択させるためのコントロールです。

チェックボックスの状態

チェックボックスの状態は IsChecked プロパティで取得･設定できます。状態は ON､OFF、Indeterminate（不確定）の三種類です。不確定は ON でも OFF でもない状態であり、IsChecked は null となります。

IsThreeState

IsThreeState プロパティを True にしたときのみ、状態を ON、OFF、Indeterminate（不確定）の 3 種類となり、これを False にした場合は ON か OFF の 2 種類のみの選択となります。

Indeterminate（不確定）の使い道

Indeterminate（不確定）は未選択状態を表すため、確実にユーザーの意思で選択してもらいたいときに有効です。初期値を Indeterminate(不確定)にしておき､Save に Indeterminate（不確定）の場合は、未選択ということでエラーチェックに引っ掛けることが可能です。IsThreeState を False にしていても IsChecked の初期値に Null を設定することが可能なため、初期値のみ Null とし、一度選択したら、ON か OFF のいずれかという操作をさせることが可能です。

イベント

IsChecked の状態が変わったときに、Checked、Unchecked、Indeterminate の 3 つのイベントのいずれかが通知されます。

サンプルコード

MainWindow.xaml
<Window x:Class="WPF014.MainWindow" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

```
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF014"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <CheckBox x:Name="MyCheckBox"
                  FontSize="20"
                  Margin="10"
                  Content="check box"
                  VerticalContentAlignment="Center"
                  Height="30"
                  VerticalAlignment="Top"

                  IsThreeState="False"
                  Checked="MyCheckBox_Checked"
                  Unchecked="MyCheckBox_Unchecked"
                  Indeterminate="MyCheckBox_Indeterminate"
                  IsChecked="{x:Null}"/>
    </Grid>
</Window>
```

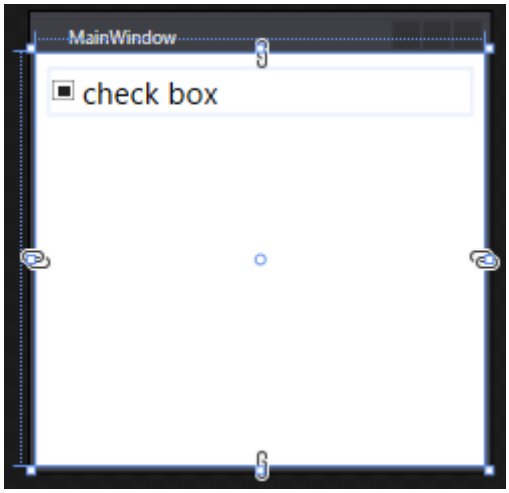
IsThreeState を True にすると 3 種類のイベントが通知されます。

MainWindow.xaml.cs

```
using System;
using System.Windows;

namespace WPF014
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void MyCheckBox_Checked(object sender, RoutedEventArgs e)
        {
            Console.WriteLine("MyCheckBox_Checked:" + MyCheckBox.IsChecked);
        }

        private void MyCheckBox_Unchecked(object sender, RoutedEventArgs e)
        {
            Console.WriteLine("MyCheckBox_Unchecked:" + MyCheckBox.IsChecked);
        }

        private void MyCheckBox_Indeterminate(object sender, RoutedEventArgs e)
        {
            Console.WriteLine("MyCheckBox_Indeterminate:" + MyCheckBox.IsChecked);
```

```
        }
    }
}
```

画面イメージ



#15　RadioButton

RadioButton（ラジオボタン）とは

RadioButton（ラジオボタン）とは、とあるグループの中から１つの項目を選択させるためのコントロールです。例えば、性別グループの「男性」「女性」「その他」を選ばせるみたいな感じのやつです。

グルーピング

基本的に同一のコンテナ（パネル等）上にあるラジオボタンは同一のグループと認識され、その中の１つの項目のみを選択することができます。

１つのパネル上に２つのラジオボタンを設置

```
<StackPanel Margin="10">
    <RadioButton x:Name="ARadioButton"
                 Content="AAA" FontSize="20"/>
    <RadioButton x:Name="BRadioButton"
                 Content="BBB"
                 FontSize="20"/>
</StackPanel>
```

パネルを２つに分けた例

```
<StackPanel>
        <StackPanel Margin="10">
            <RadioButton x:Name="ARadioButton"
                         Content="AAA" FontSize="20"/>
            <RadioButton x:Name="BRadioButton"
                         Content="BBB"
                         FontSize="20"/>
        </StackPanel>

        <StackPanel Margin="10">
            <RadioButton x:Name="CRadioButton"
                         Content="CCC" FontSize="20"/>
            <RadioButton x:Name="DRadioButton"
                         Content="DDD" FontSize="20"/>
        </StackPanel>
</StackPanel>
```

パネルを２つに分けたため、AAA、BBB のグループと CCC、DDD のグループで別々に動作します。

GroupName

GroupName プロパティを指定すると、同一のコンテナ（パネル等）上にあっても、別のグ

ループとして動作させることができます。

GroupName を指定する例

```
<StackPanel Margin="10">
    <RadioButton x:Name="CRadioButton"
                 Content="CCC" FontSize="20"/>
    <RadioButton x:Name="DRadioButton"
                 Content="DDD" FontSize="20"/>

    <RadioButton x:Name="ERadioButton"
                 Content="EEE" FontSize="20"
                 GroupName="1"/>
    <RadioButton x:Name="FRadioButton"
                 Content="FFF" FontSize="20"
                 GroupName="1"/>
</StackPanel>
```

4 つのラジオボタンが同一のパネル上にありますが、EEE と FFF に GroupName を指定しているため、CCC、DDD のグループと、EEE、FFF のグループは別グループとして動作します。

状態

状態は、チェックボックスと同様に IsChecked で確認できます。内容は True、False、null（未確定）の 3 種類です。

イベント

イベントは Checked、Unchecked で通知されます。Indeterminate の通知も受けることができますが、あまり用途はないような気がします。

サンプルコード

MainWindow.xaml

```
<Window x:Class="WPF015.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF015"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel>
            <StackPanel Margin="10">
                <RadioButton x:Name="ARadioButton"
                             Content="AAA" FontSize="20"
                             Checked="ARadioButton_Checked"/>
```

```
                <RadioButton x:Name="BRadioButton"
                             Content="BBB"
                             FontSize="20"
                             Checked="BRadioButton_Checked"/>
            </StackPanel>

            <StackPanel Margin="10">
                <RadioButton x:Name="CRadioButton"
                             Content="CCC" FontSize="20"
                             Checked="CRadioButton_Checked"/>
                <RadioButton x:Name="DRadioButton" Content="DDD" FontSize="20"
                             Checked="CRadioButton_Checked"/>

                <RadioButton x:Name="ERadioButton" Content="EEE" FontSize="20"
                             GroupName="1"/>
                <RadioButton x:Name="FRadioButton" Content="FFF" FontSize="20"
                             GroupName="1"/>
            </StackPanel>
        </StackPanel>

    </Grid>
</Window>
```

AAA と BBB はそれぞれに Checked イベントを実装。CCC、DDD は CRadioButton_Checked という同じ Checked イベントを実装。

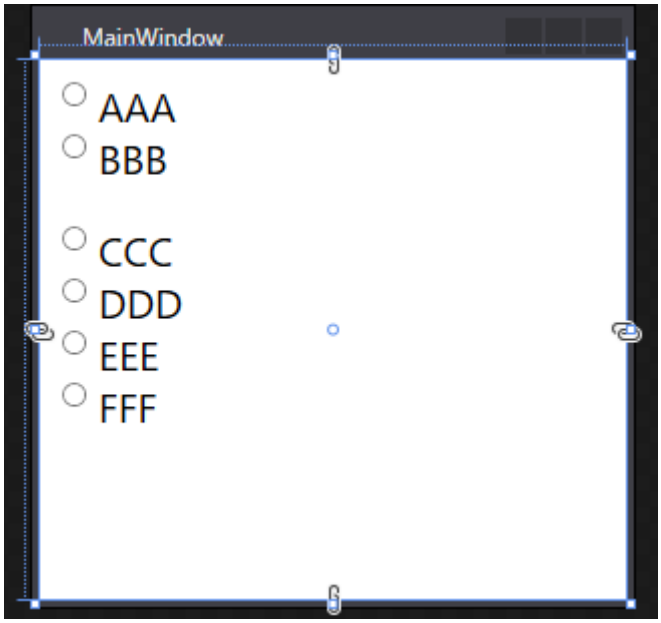
MainWindow.xaml.cs

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

namespace WPF015
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void ARadioButton_Checked(object sender, RoutedEventArgs e)
        {

        }

        private void BRadioButton_Checked(object sender, RoutedEventArgs e)
        {

        }

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

```
            var radioButton = sender as RadioButton;
            if(radioButton == CRadioButton)
            {

            }
            else if(radioButton == DRadioButton)
            {

            }
        }
    }
}
```

ARadioButton と BRadioButton はそれぞれの Checked イベントにチェック状態になると通知されますが、CRadioButton と DRadioButton は同一の CRadioButton_Checked に通知されるため、イベントの中で、どちらの通知かを判断して処理をすることができます。

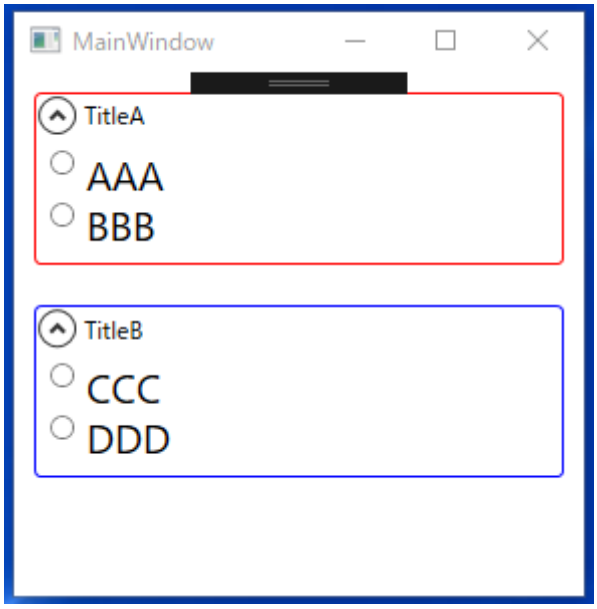
＜画面イメージ＞

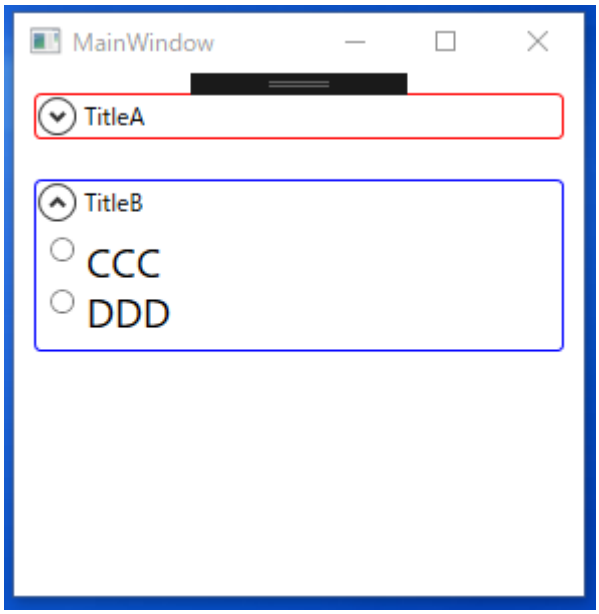


#16　Expander

Expander とは？

Expander コントロールとは、エリアの開閉ができるコントロールです。





「TitleA」などと書かれている横のボタンを押すと、開いたり閉じたりします。

プロパティ

Header	タイトルを設定します

IsExpanded	開閉状態を設定できます。True のときに開いた状態になります。
BorderBrush	ボーダーラインの色を設定します

注意点

Expander 自体の Height（高さ）を指定しない場合は、Expander を閉じたときに、エリアは縮小され、その下にあるコントロールも追従して上に上がります。Height（高さ）を指定すると、閉じた場合もエリアの高さはそのままとなります。

サンプルコード

MainWindow.xaml

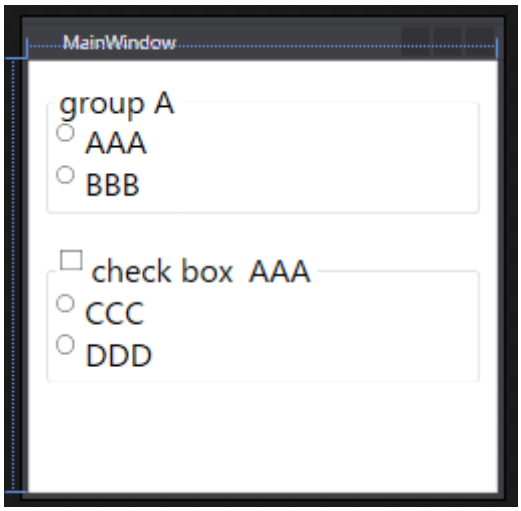
```
<Window x:Class="WPF016.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF016"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel>
            <Expander Header="TitleA"
                      Margin="10"
                      IsExpanded="True"
                      BorderBrush="Red">
                <StackPanel Margin="6"
                            HorizontalAlignment="Left"
                            Width="200">
                    <RadioButton Content="AAA"
                                 FontSize="20"/>
                    <RadioButton Content="BBB"
                                 FontSize="20"/>
                </StackPanel>
            </Expander>

            <Expander Header="TitleB"
                      Margin="10"
                      IsExpanded="True"
                      BorderBrush="Blue">
                <StackPanel Margin="6"
                            HorizontalAlignment="Left"
                            Width="200">
                    <RadioButton Content="CCC"
                                 FontSize="20"/>
                    <RadioButton Content="DDD"
                                 FontSize="20"/>
```

```
                </StackPanel>
            </Expander>
        </StackPanel>
    </Grid>
</Window>
```

#17　GroupBox（グループボックス）

GroupBox（グループボックス）とは？

GroupBox とは、グループ単位でコントロールを整理して配置するパネル的なコントロールです。



GroupBox の使い方

基本的にはタイトルとしてのヘッダーに文言を設置し、グループの中にラジオボタンなどのコントロールを設置して使います。

ヘッダーが文字のみの例

```
<GroupBox Margin="10"
          Header="group A"
          FontSize="20">
    <StackPanel>
        <RadioButton FontSize="20" Content="AAA"/>
        <RadioButton FontSize="20" Content="BBB"/>
    </StackPanel>
</GroupBox>
```

ヘッダーに複数のコントロールを組み合わせる方法

ヘッダー部分に文字のみを表示する場合はいいのですが、画像と文字や、チェックボックスと文字など、コントロールを組み合して表示したい場合があります。その場合は GroupBox.Header というキーワードを使用します。

ヘッダーに複数のコントロールを設置する例

```
<GroupBox Margin="10">
```

```
    <GroupBox.Header>
        <StackPanel Orientation="Horizontal">
            <CheckBox Content="check box" FontSize="20"/>
            <TextBlock Text="AAA" FontSize="20" Margin="10,0,0,0"/>
        </StackPanel>
    </GroupBox.Header>
    <StackPanel>
        <RadioButton FontSize="20" Content="CCC"/>
        <RadioButton FontSize="20" Content="DDD"/>
    </StackPanel>
</GroupBox>
```

サンプルコード

MainWindow.xaml

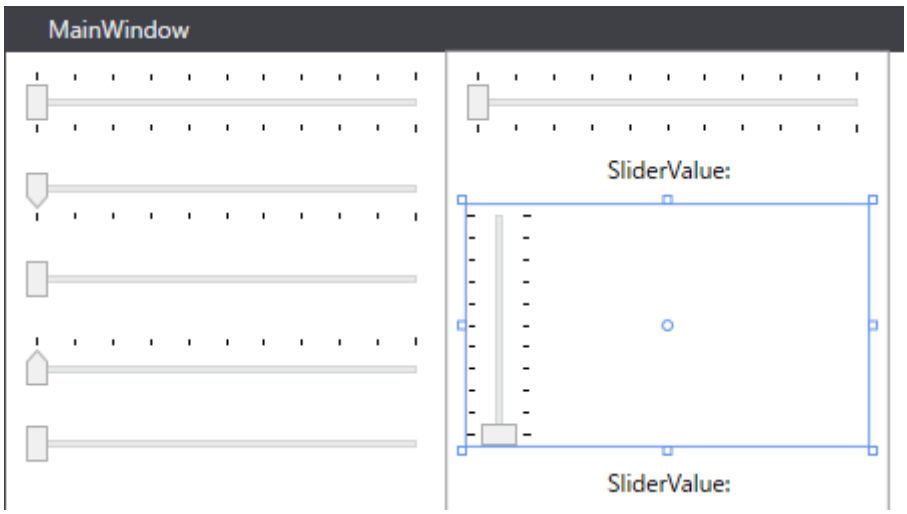
```
<Window x:Class="WPF017.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF017"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel>
            <GroupBox Margin="10"
                      Header="group A"
                      FontSize="20">
                <StackPanel>
                    <RadioButton FontSize="20" Content="AAA"/>
                    <RadioButton FontSize="20" Content="BBB"/>
                </StackPanel>
            </GroupBox>

            <GroupBox Margin="10">
                <GroupBox.Header>
                    <StackPanel Orientation="Horizontal">
                        <CheckBox Content="check box" FontSize="20"/>
                        <TextBlock Text="AAA" FontSize="20" Margin="10,0,0,0"/>
                    </StackPanel>
                </GroupBox.Header>
                <StackPanel>
                    <RadioButton FontSize="20" Content="CCC"/>
                    <RadioButton FontSize="20" Content="DDD"/>
                </StackPanel>
            </GroupBox>
        </StackPanel>
    </Grid>
</Window>
```

#18　Slider（スライダー）

Slider とは？

Slider とは、ボリュームのレベルを調整するようなコントロールで、目盛りとつまみのあるものです。ユーザーに任意の値を調整させるときなどに使えます。

コントロールのイメージ



Slider の使い方

TickPlacement

目盛りの位置は TickPlacement プロパティで設定し、次の 4 種類から選べます。

- Both
  - 上下に目盛りが表示される（画面左の 1 番目）
- BottomRight
  - 下に目盛りが表示される（画面左の 2 番目）
- None
  - 目盛りなし（画面左の 3 番目）
- TopLeft
  - 上に目盛りが表示される（画面左の 4 番目）
- 設定しないとき
  - None と同じ（画面左の 5 番目）

IsSnapToTickEnabled

近くの値にスナップするときは True にします。スナップとは例えば, 移動の最小単位を 10 と設定している場合、カーソル移動で 17 を指定すると、一番近い最小単位の 20 が選択されます。

TickFrequency

カーソルでドラッグしながらレベルを調整するときの、増減する値を指定します。

SmallChange

キーボードの矢印キーを押して、レベルを調整するとき増減する値を設定します。

LargeChange

スライダーの任意の位置をクリックしたときに増減する値を設定します。

Orientation

スライダーの向きを縦か横で指定できます。

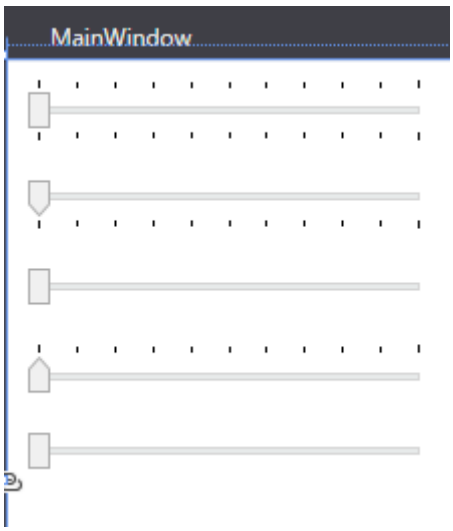
サンプルコード

横向きスライダー

TickPlacement の 4 種類と指定なしをそれぞれ記述した例を示します。

TickPlacement の 4 種類と指定なし

```
<StackPanel>
    <Slider TickPlacement="Both" Width="200" Foreground="Black" Margin="10"/>
    <Slider TickPlacement="BottomRight" Width="200" Foreground="Black" Margin="10"/>
    <Slider TickPlacement="None" Width="200" Foreground="Black" Margin="10"/>
    <Slider TickPlacement="TopLeft" Width="200" Foreground="Black" Margin="10"/>
    <Slider Width="200" Foreground="Black" Margin="10"/>
</StackPanel>
```



上から順番に、目盛りの位置が変化していることが確認できます。また、指定しない場合は None と同じ状態であることがわかります。

MainWindow.xaml

```
<Slider Width="200"
        TickPlacement="Both"
        Foreground="Black"
        Margin="10"
        IsSnapToTickEnabled="True"
        TickFrequency="10"
        SmallChange="20"
        LargeChange="50"
        Minimum="0"
        Maximum="100"
        ValueChanged="Slider_ValueChanged"/>
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
    <TextBlock Text="SliderValue:"/>
    <TextBlock x:Name="ASlider"/>
</StackPanel>
```

MainWindow.xaml.cs

```
private void Slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    ASlider.Text = e.NewValue.ToString();
}
```

- IsSnapToTickEnabled を True にしているので、近くの値にスナップされます。
- つまみをもってドラッグすると TickFrequency の値「１０」単位で移動します。
- キーボードの左右キーで移動させた場合は SmallChange の「２０」単位で移動します。
- スライダー上をクリックしてレベルを調整する場合は LargeChange の「５０」単位で移動します。
- スライダーの最小値と最大値は Minimum と Maximum で指定するので 0 から１００

までが選択できる状態になっています。

- スライダーの下に、スライダーの値を表示するための TextBlock を設置しています。値が変わると Slider_ValueChanged で値を設定しています。

縦向きスライダー

MainWindow.xaml

```
<Slider Width="200"
        TickPlacement="Both"
        Foreground="Black"
        Margin="10"
        IsSnapToTickEnabled="False"
        TickFrequency="10"
        SmallChange="20"
        LargeChange="50"
        Minimum="0"
        Maximum="100"
        ValueChanged="BSlider_ValueChanged"
        Orientation="Vertical"
        Height="120"/>
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
    <TextBlock Text="SliderValue:"/>
    <TextBlock x:Name="BSlider"/>
</StackPanel>
```

MainWindow.xaml.cs

```
private void BSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    BSlider.Text = e.NewValue.ToString();
}
```

- IsSnapToTickEnabled を False にしているのでスナップされないことが確認できます。
- Orientation を Vertical にすることで、スライダーが縦向きになります。

サンプルコード全体

MainWindow.xaml

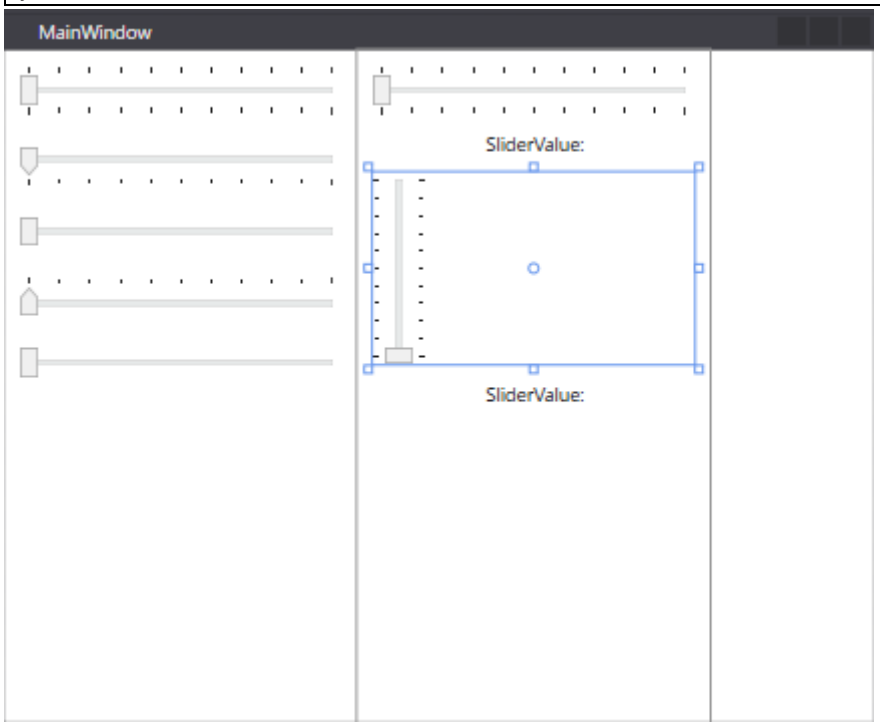
```
<Window x:Class="WPF018.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF018"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="550">
    <Grid>
        <StackPanel Orientation="Horizontal">
            <StackPanel>
                <Slider TickPlacement="Both" Width="200" Foreground="Black" Margin="10"/>
```

```
            <Slider TickPlacement="BottomRight" Width="200" Foreground="Black" Margin="10"/>
            <Slider TickPlacement="None" Width="200" Foreground="Black" Margin="10"/>
            <Slider TickPlacement="TopLeft" Width="200" Foreground="Black" Margin="10"/>
            <Slider Width="200" Foreground="Black" Margin="10"/>
        </StackPanel>

        <StackPanel>
            <Slider Width="200"
                    TickPlacement="Both"
                    Foreground="Black"
                    Margin="10"
                    IsSnapToTickEnabled="True"
                    TickFrequency="10"
                    SmallChange="20"
                    LargeChange="50"
                    Minimum="0"
                    Maximum="100"
                    ValueChanged="Slider_ValueChanged"/>
            <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
                <TextBlock Text="SliderValue:"/>
                <TextBlock x:Name="ASlider"/>
            </StackPanel>

            <Slider Width="200"
                    TickPlacement="Both"
                    Foreground="Black"
                    Margin="10"
                    IsSnapToTickEnabled="False"
                    TickFrequency="10"
                    SmallChange="20"
                    LargeChange="50"
                    Minimum="0"
                    Maximum="100"
                    ValueChanged="BSlider_ValueChanged"
                    Orientation="Vertical"
                    Height="120"/>
            <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
                <TextBlock Text="SliderValue:"/>
                <TextBlock x:Name="BSlider"/>
            </StackPanel>
        </StackPanel>
    </StackPanel>
</Grid>
</Window>
```

MainWindow.xaml.cs

```
using System.Windows;

namespace WPF018
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
```

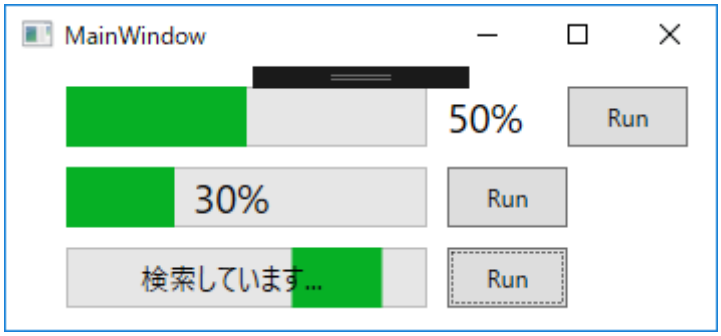
```
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
        {
            ASlider.Text = e.NewValue.ToString();
        }

        private void BSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
        {
            BSlider.Text = e.NewValue.ToString();
        }
    }
}
```



#19　ProgressBar（プログレスバー）

ProgressBar（プログレスバー）とは？

ProgressBar とは時間がかかる処理を非同期でさせている最中に，進捗状況を画面に表示するためのコントロールです。



主なプロパティ

- 進捗状況を Value に設定します。
- 最小値を Minimum，最大値を Maximum に設定します。パーセントで表示する場合は 0 から 100 で設定します。ファイル数等，処理の最大値がわかっている場合はその値を表示することで，どの程度作業が終わっているのかがユーザーに分かりやすくなります。

主なイベント

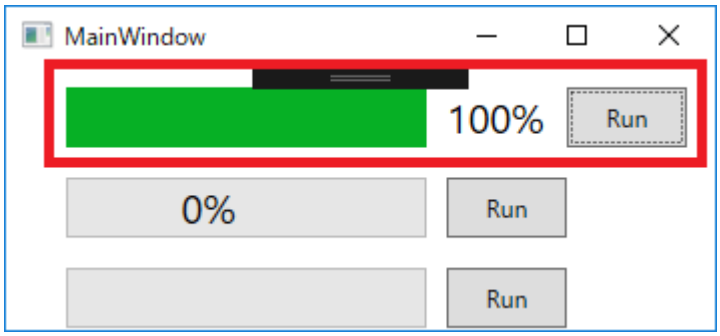
ValueChanged

Value が変更されたときに通知されるので，このタイミングで TextBlock の Text プロパティに文字を設定することで，プログレスバーの目盛りだけではなく，文字でも表示することができるので，よりユーザーに分かりやすくなります。

表示形式

最初の画像で示している通り，主な表現の方法は 3 パターンあります。

パターン１：プログレスバーとテキストを別で表示する



この方法が一番簡単で一般的です。プログレスバーの周りに TextBlock などを設置して，プログレスバーとテキスト表示を別で表示します。もちろんプログレスバー単体での表示も可能ですが，その場合，ユーザーは大体の値しか把握できなくなります。

MainWindow.xaml

```
<ProgressBar x:Name="AProgressBar"
                  Margin="25,0,0,0"
                  Height="30"
                  Width="180"
                  Minimum="0"
                  Maximum="100"
                  HorizontalAlignment="Left"
                  ValueChanged="AProgressBar_ValueChanged"
                  />
    <TextBlock x:Name="ATextBlock"
               Margin="10,0,0,0"
               HorizontalAlignment="Center"
               VerticalAlignment="Center"
               Text="xxx"
               Width="50"
               FontSize="20"/>
    <Button x:Name="AButton"
            Margin="10,0,0,0"
            Click="AButton_Click"
            Height="30"
            Width="60"
            Content="Run"/>

</StackPanel>
```

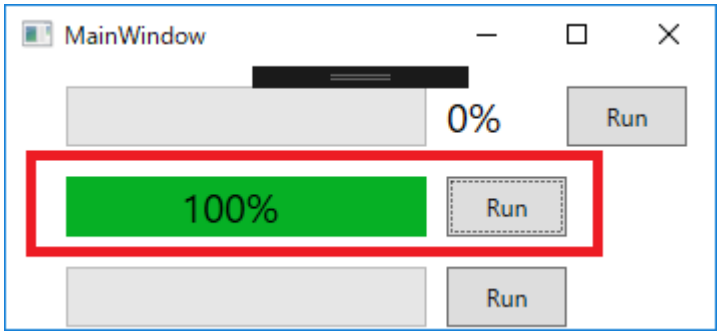
MainWindow.xaml.cs

```
public MainWindow()
{
    InitializeComponent();
    ATextBlock.Text = AProgressBar.Value.ToString() + "%";
}

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

```
{
    ATextBlock.Text = AProgressBar.Value.ToString() + "%";
}

private void AButton_Click(object sender, RoutedEventArgs e)
{
    Task.Run(() =>
    {
        for (int i = 0; i < 10; i++)
        {
            System.Threading.Thread.Sleep(500);

            Application.Current.Dispatcher.Invoke(() =>
            {
                AProgressBar.Value += 10;
            });
        }
    });
}
```

- AButton_Click の中では，時間のかかる処理を作るために， 500 ミリ秒の Sleep を 10 回実施しています。
- Sleep 1 回実施されるたびに ProgressBar の Value を 10 ずつ増やしています。
- 非同期処理の中でコントロールを操作できないため，Application.Current.Dispatcher.Invoke で，UI スレッドに戻してから，プログレスバーの値を変更しています。

パターン２：プログレスバーとテキストを重ねて表示する



この表示の仕方も結構ニーズがあると思います。ProgressBar 自体にこの方法を実現するプロパティは実装されていないので，自分で実装する必要があります。やり方は ProgressBar と TextBlock を同一の Grid 上に配置し，同一の列と行の設定にします。同一の Grid にこの２つしかコントロールがない場合はどちらも行列がゼロになっているので，あえて行列の設定をする必要はありません。

MainWindow.xaml
<StackPanel Margin="5,10,5,5"

```
            Orientation="Horizontal">
    <Grid>
        <ProgressBar x:Name="BProgressBar"
                 Margin="25,0,0,0"
                 Height="30"
                 Width="180"
                 Minimum="0"
                 Maximum="100"
                 HorizontalAlignment="Left"
                 ValueChanged="BProgressBar_ValueChanged"
                 />
        <TextBlock x:Name="BTextBlock"
               Margin="10,0,0,0"
               HorizontalAlignment="Center"
               VerticalAlignment="Center"
               Text="xxx"
               Width="50"
               FontSize="20"/>
    </Grid>

    <Button x:Name="BButton"
            Margin="10,0,0,0"
            Click="BButton_Click"
            Height="30"
            Width="60"
            Content="Run"/>

</StackPanel>
```

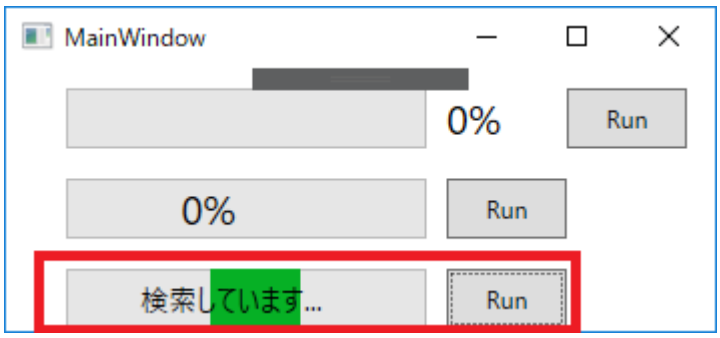
パターン 1 とほとんど同じです。違いは ProgressBar と TextBlock を同一の Grid 上に載せているだけです。

MainWindow.xaml.cs

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

    BTextBlock.Text = BProgressBar.Value.ToString() + "%";
}

private void BProgressBar_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    BTextBlock.Text = BProgressBar.Value.ToString() + "%";
}

private void BButton_Click(object sender, RoutedEventArgs e)
{
    Task.Run(() =>
    {
        for (int i = 0; i < 10; i++)
        {
            System.Threading.Thread.Sleep(500);

            Application.Current.Dispatcher.Invoke(() =>
```

```
            {
                BProgressBar.Value += 10;
            });
        }
    });
}
```

パターン３：終了するタイミングが不明な時



すべての処理が，終了するタイミングがわかるわけではありません。データベースの問い合わせなどは，データベースからの返答がいつになるかわかりません。そういった場合は，処理中であることだけを示し，画面が固まっていないことを表現するためにも，何かしらのアニメーションが動作しているほうが望ましいです。そういった場合にプログレスバーの目盛りが，左から右に永遠と流れるだけの機能が実装されています。やり方は IsIndeterminate というプロパティを True にするだけです。処理が終わったら False にすれば止まります。パターン２の TextBlock を重ねる方法を利用して「検索しています…」等と表示すれば，一層の効果があります。

MainWindow.xaml

```
<StackPanel Margin="5,10,5,5"
            Orientation="Horizontal">
    <Grid>
        <ProgressBar x:Name="CProgressBar"
                 Margin="25,0,0,0"
                 Height="30"
                 Width="180"
                 Minimum="0"
                 Maximum="100"
                 HorizontalAlignment="Left"
                 />
        <TextBlock x:Name="CTextBlock"
               Margin="10,0,0,0"
               HorizontalAlignment="Center"
               VerticalAlignment="Center"
               Text=""
               FontSize="14"/>
```

```
    </Grid>

    <Button x:Name="CButton"
            Margin="10,0,0,0"
            Click="CButton_Click"
            Height="30"
            Width="60"
            Content="Run"/>

</StackPanel>
```

MainWindow.xaml.cs

```
private void CButton_Click(object sender, RoutedEventArgs e)
{
    CProgressBar.IsIndeterminate = true;
    CTextBlock.Text = "検索しています...";
}
```

処理を実行したタイミングで IsIndeterminate を True にして，TextBlock に検索中の文言を表示しています。処理が終わったタイミングで IsIndeterminate を False にしたり，TextBlock の文言を消したり必要があります。

サンプルコード全体

MainWindow.xaml

```
<Window x:Class="WPF019.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF019"
        mc:Ignorable="d"
        Title="MainWindow" Height="170" Width="370">
    <Grid>
        <StackPanel>
            <StackPanel Margin="5,10,5,5"
                        Orientation="Horizontal">
                <ProgressBar x:Name="AProgressBar"
                             Margin="25,0,0,0"
                             Height="30"
                             Width="180"
                             Minimum="0"
                             Maximum="100"
                             HorizontalAlignment="Left"
                             ValueChanged="AProgressBar_ValueChanged"
                             />
                <TextBlock x:Name="ATextBlock"
                           Margin="10,0,0,0"
                           HorizontalAlignment="Center"
                           VerticalAlignment="Center"
                           Text="xxx"
```

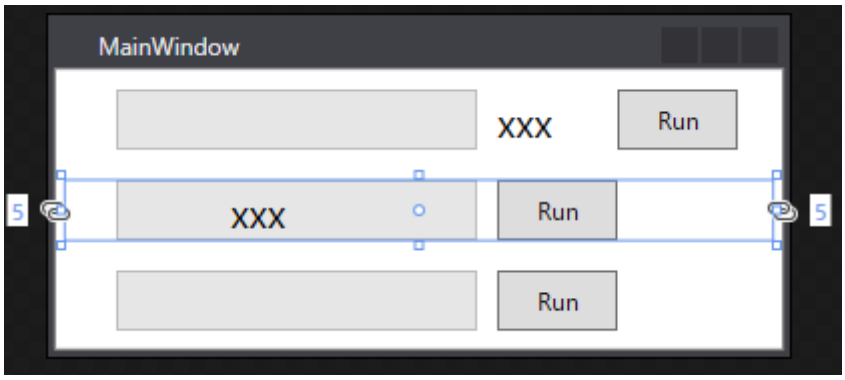
```
                    Width="50"
                    FontSize="20"/>
        <Button x:Name="AButton"
                Margin="10,0,0,0"
                Click="AButton_Click"
                Height="30"
                Width="60"
                Content="Run"/>

    </StackPanel>

    <StackPanel Margin="5,10,5,5"
                Orientation="Horizontal">
        <Grid>
            <ProgressBar x:Name="BProgressBar"
                     Margin="25,0,0,0"
                     Height="30"
                     Width="180"
                     Minimum="0"
                     Maximum="100"
                     HorizontalAlignment="Left"
                     ValueChanged="BProgressBar_ValueChanged"
                     />
            <TextBlock x:Name="BTextBlock"
                   Margin="10,0,0,0"
                   HorizontalAlignment="Center"
                   VerticalAlignment="Center"
                   Text="xxx"
                   Width="50"
                   FontSize="20"/>
        </Grid>

        <Button x:Name="BButton"
                Margin="10,0,0,0"
                Click="BButton_Click"
                Height="30"
                Width="60"
                Content="Run"/>

    </StackPanel>

    <StackPanel Margin="5,10,5,5"
                Orientation="Horizontal">
        <Grid>
            <ProgressBar x:Name="CProgressBar"
                     Margin="25,0,0,0"
                     Height="30"
                     Width="180"
                     Minimum="0"
                     Maximum="100"
                     HorizontalAlignment="Left"
                     />
            <TextBlock x:Name="CTextBlock"
                   Margin="10,0,0,0"
```

```
                            HorizontalAlignment="Center"
                            VerticalAlignment="Center"
                            Text=""
                            FontSize="14"/>
                </Grid>

                <Button x:Name="CButton"
                        Margin="10,0,0,0"
                        Click="CButton_Click"
                        Height="30"
                        Width="60"
                        Content="Run"/>

            </StackPanel>
        </StackPanel>
    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System.Threading.Tasks;
using System.Windows;

namespace WPF019
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            ATextBlock.Text = AProgressBar.Value.ToString() + "%";
            BTextBlock.Text = BProgressBar.Value.ToString() + "%";
        }

        private void AProgressBar_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            ATextBlock.Text = AProgressBar.Value.ToString() + "%";
        }

        private void AButton_Click(object sender, RoutedEventArgs e)
        {
            Task.Run(() =>
            {
                for (int i = 0; i < 10; i++)
                {
                    System.Threading.Thread.Sleep(500);

                    Application.Current.Dispatcher.Invoke(() =>
                    {
                        AProgressBar.Value += 10;
```

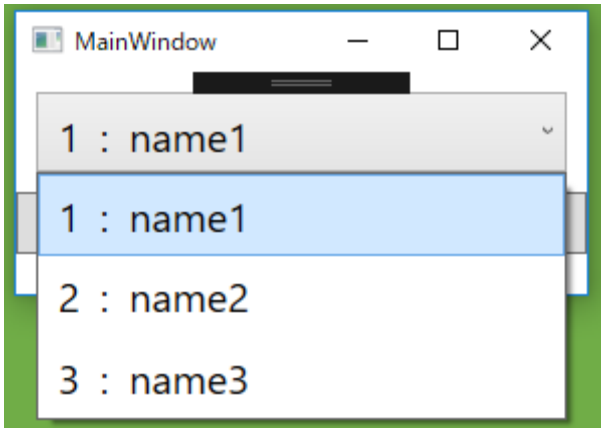
```
                    });
                }
            });
        }

        private void BProgressBar_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            BTextBlock.Text = BProgressBar.Value.ToString() + "%";
        }

        private void BButton_Click(object sender, RoutedEventArgs e)
        {
            Task.Run(() =>
            {
                for (int i = 0; i < 10; i++)
                {
                    System.Threading.Thread.Sleep(500);

                    Application.Current.Dispatcher.Invoke(() =>
                    {
                        BProgressBar.Value += 10;
                    });
                }
            });
        }

        private void CButton_Click(object sender, RoutedEventArgs e)
        {
            CProgressBar.IsIndeterminate = true;
            CTextBlock.Text = "検索しています...";
        }
    }
}
```



#20　ComboBox（コンボボックス）

ComboBox（コンボボックス）とは？

ComboBox とは，一覧の中から，任意の項目を選択するときに使うコントロールです。



主な使い方

使い方は，単純に Items に文字を Add して使うやり方もできますが，実際のプログラミングでは，表示されている文字とは別に，内部では Id で扱うことがほとんどなので，基本的には ItemsSource にデータバインディングをして使うことになります。今回はそれぞれのやり方を見ていきます。

パターン１：単純に Items に文字を Add して使う

MainWindow.xaml

```
<StackPanel Orientation="Horizontal">
    <ComboBox x:Name="MyComboBox"
              Height="40"
              Width="200"
              Margin="10"
              FontSize="20"
              VerticalAlignment="Center"/>
    <Button Margin="10"
            Width="50"
            Content="check"
            Click="MyButton_Click"
            />
</StackPanel>
```

- Xaml にコンボボックスとボタンを設置

MainWindow.xaml.cs

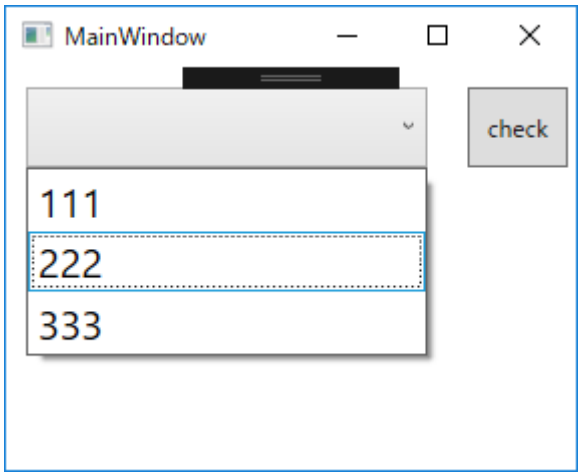
```
public MainWindow()
{
```

```
    InitializeComponent();

    MyComboBox.Items.Add("111");
    MyComboBox.Items.Add("222");
    MyComboBox.Items.Add("333");
}

private void MyButton_Click(object sender, RoutedEventArgs e)
{
    var sb = new StringBuilder();
    sb.AppendLine("MyComboBox.SelectedIndex:" + MyComboBox.SelectedIndex);
    sb.AppendLine("MyComboBox.SelectedValue:" + MyComboBox.SelectedValue);
    sb.AppendLine("MyComboBox.Text:" + MyComboBox.Text);
    MessageBox.Show(sb.ToString());
}
```

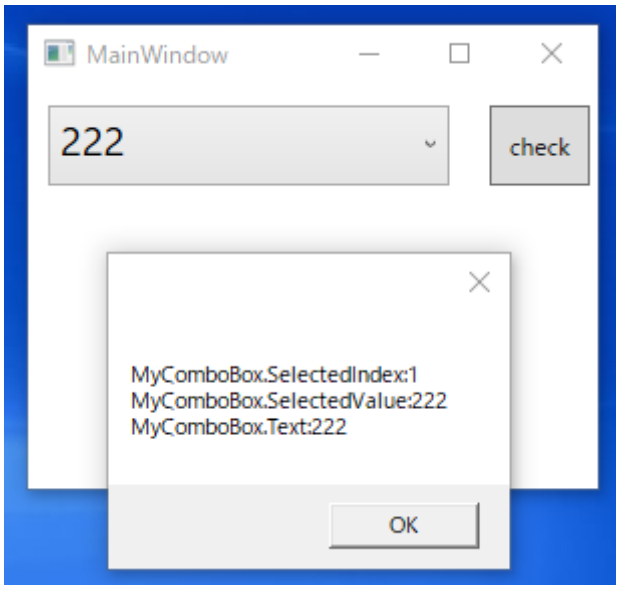
- コンストラクタでコンボボックスに文字列を３つ追加
- ボタンクリックイベントで各プロパティの内容を確認するメッセージボックスを表示

実行結果

- シンプルに文字列が３つ追加されている。



「222」を選択して Check ボタンを押下したときのメッセージボックス



- SelectedIndex は「１」。これはインデックスが０始まりなので２番目を選択したためインデックスは１となる。
- SelectedValue と Text は両方「２２２」となる。

この方法は非常にシンプルだが，任意の文字列を表示して，内部では ID で管理するというコーディングができない。実際のプログラミングでは使用することはないと思います。私は基本的に使いません。プロトタイプで取り急ぎ表示させる時くらいにしか用途はないかと思います。

パターン２：データバインディング（単一項目表示）

ここからはデータバインディングする方法を見ていきたいと思います。データバインディングする場合は，任意のクラスを作成して，そのリストをコンボボックスの ItemsSource に設定します。Id と表示文字をセットにして，コンボボックスで扱うだけであれば，SelectedValuePath と DisplayMemberPath を使用することで実現できます。

- ItemsSource
  - コンボボックスにバインディングするリスト
- SelectedValuePath
  - 選択されているときに SelectedValue で取得される項目を設定

- DisplayMemberPath
  - 表示する文字の項目を設定。Text で取得される値。

Customer.cs

```
namespace WPF020
{
    public class Customer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Phone { get; set; }
    }
}
```

- コンボボックスにデータバインディングするためのクラスです。ID と名前と電話番号のクラスです。

MainWindow.xaml

```
<StackPanel Orientation="Horizontal">
    <ComboBox x:Name="AComboBox"
              Height="40"
              Width="200"
              Margin="10"
              FontSize="20"
              VerticalAlignment="Center"
              SelectedValuePath="Id"
              DisplayMemberPath="Name"/>
    <Button Margin="10"
            Width="50"
            Content="check"
            Click="AButton_Click"
            />
</StackPanel>
```

- SelectedValuePath と DisplayMemberPath を設定しています。
- 表示文字は Name を表示し，内部的には Id で処理をすることができます。
- 選択されている Id は SelectedValue で取得できます。
- 選択されている Name は Text で取得することができます。

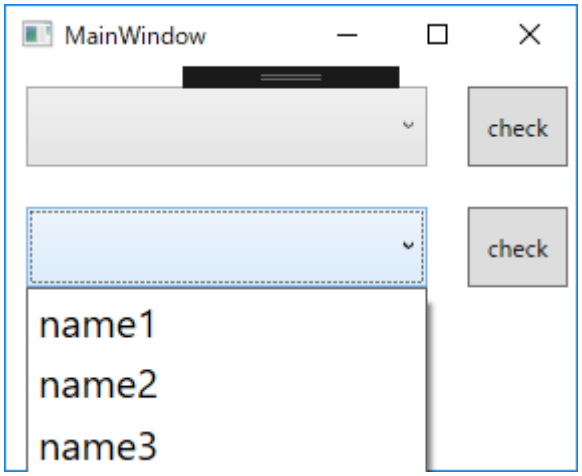
MainWindow.xaml.cs

```
private ObservableCollection<Customer> _customers = new ObservableCollection<Customer>();
public MainWindow()
{
    InitializeComponent();

    _customers.Add(new Customer { Id = 1, Name = "name1", Phone = "Phone1" });
    _customers.Add(new Customer { Id = 2, Name = "name2", Phone = "Phone2" });
    _customers.Add(new Customer { Id = 3, Name = "name3", Phone = "Phone3" });

    AComboBox.ItemsSource = _customers;
```

```
}

private void AButton_Click(object sender, RoutedEventArgs e)
{
    var item = AComboBox.SelectedItem as Customer;
    if (item != null)
    {
        var sb = new StringBuilder();
        sb.AppendLine("AComboBox.SelectedIndex:" + AComboBox.SelectedIndex);
        sb.AppendLine("AComboBox.SelectedValue:" + AComboBox.SelectedValue);
        sb.AppendLine("AComboBox.Text:" + AComboBox.Text);
        sb.AppendLine("-------");
        sb.AppendLine("SelectedItem.Id : " + item.Id);
        sb.AppendLine("SelectedItem.Name : " + item.Name);
        sb.AppendLine("SelectedItem.Phone : " + item.Phone);
        MessageBox.Show(sb.ToString());
    }
}
```

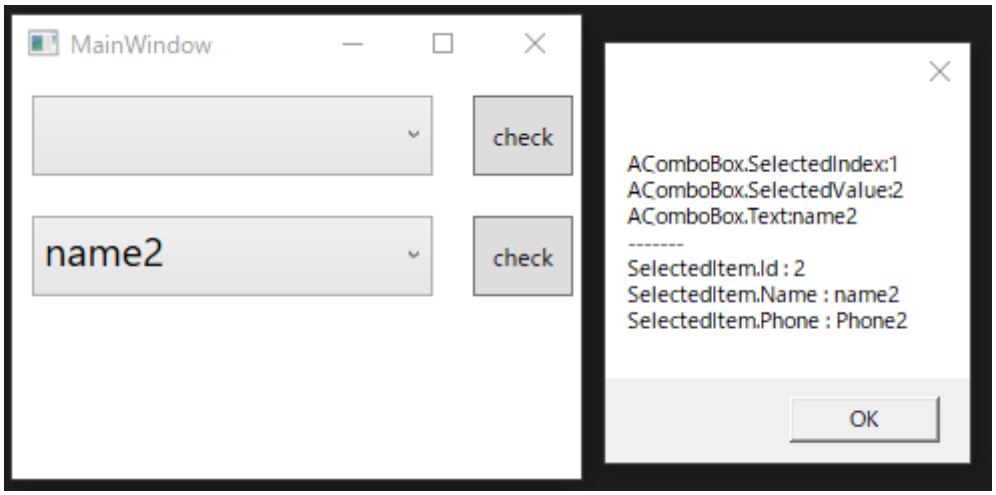
- Private フィールドに ObservableCollection で Customer のリストを宣言しています。
- コンストラクタで 3 つの Customer を生成して_customers に Add しています。
- ボタンクリック時に各プロパティの値を確認しています。
- SelectedItem は Object 型の為 Customer クラスに変換し，取得できた場合のみ処理しています。

実行結果



- ObservableCollection に追加した Customer クラスの Name の値がリストされています。これは DisplayMemberPath に「Name」を設定しているためです。

「name2」を選択して「check」ボタンを押下したとき



- SelectedIndex には「１」が設定されています。
- SelectedValue には「２」が設定されています。これは SelectedValuePath で「Id」を設定しているためです。
- Text には「name2」が設定されています。これは DisplayMemberPath に「Name」を設定しているためです。
- SelectedItem には選択されている Customer の値が取得できています。

この方法を使えば，任意の表示項目と，内部で扱う Id を分けてコーディングすることができます。ただ，表示する項目を加工したい場合や，複数のコントロールに分けて表示するレイアウトにしたい場合は実現できないので，次に紹介する ItemTemplate と DataTemplate を使用するやり方で実現します。

パターン３：リストに複数のコントロールを表示する場合

リストに複数のコントロールを置く場合は ItemTemplate と DataTemplate を使って実現できます。

MainWindow.xaml

```
<StackPanel Orientation="Horizontal">
    <ComboBox x:Name="BComboBox"
              Height="40"
              Width="200"
              Margin="10"
              FontSize="20"
              VerticalAlignment="Center">
        <ComboBox.ItemTemplate>
            <DataTemplate>
                <StackPanel Orientation="Horizontal">
                    <TextBlock Text="{Binding Id}" FontSize="20" Margin="5"/>
```

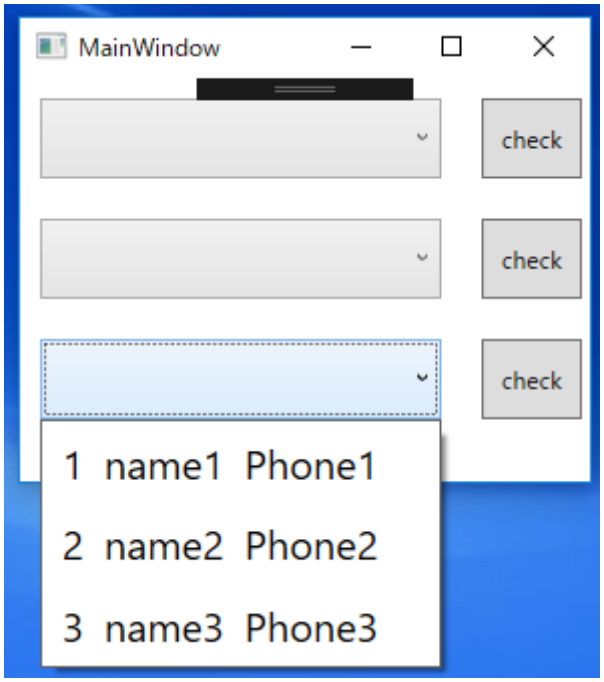
```
                    <TextBlock Text="{Binding Name}" FontSize="20" Margin="5"/>
                    <TextBlock Text="{Binding Phone}" FontSize="20" Margin="5"/>
                </StackPanel>
            </DataTemplate>
        </ComboBox.ItemTemplate>
    </ComboBox>
    <Button Margin="10"
            Width="50"
            Content="check"
            Click="BButton_Click"
            />
</StackPanel>
```

- ComboBox.ItemTemplate と DataTemplate エリアを記述します。
- DataTemplate の中に任意のコントロールを設置します。ここでは StackPanel の中に複数の TextBlock を設置しています。
- TextBlock の Text には ItemsSource にデータバインディングする値に対応するプロパティをバインドしています。

MainWindow.xaml.cs

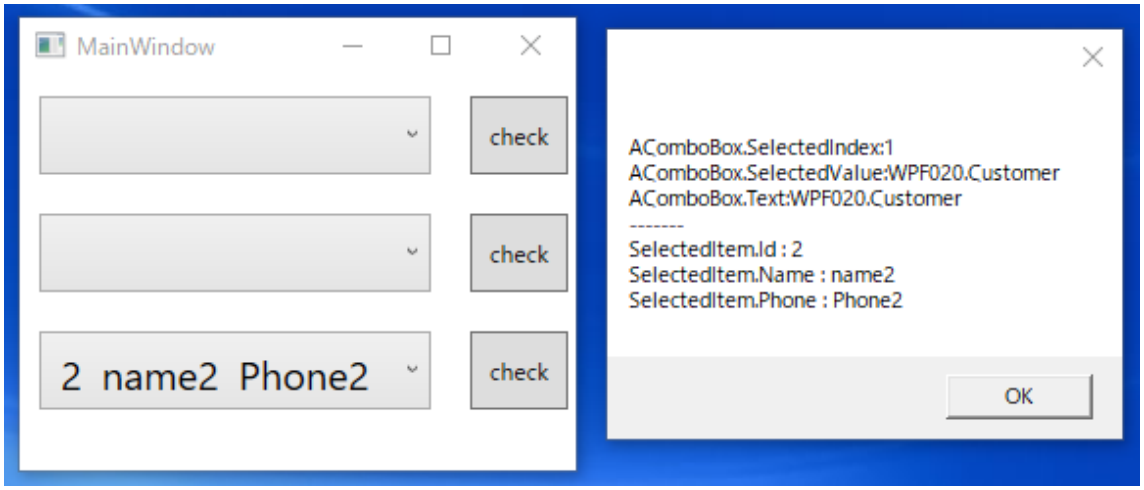
```
private ObservableCollection<Customer> _customers
    = new ObservableCollection<Customer>();
public MainWindow()
{
    InitializeComponent();

    _customers.Add(new Customer { Id = 1, Name = "name1", Phone = "Phone1" });
    _customers.Add(new Customer { Id = 2, Name = "name2", Phone = "Phone2" });
    _customers.Add(new Customer { Id = 3, Name = "name3", Phone = "Phone3" });

    BComboBox.ItemsSource = _customers;
}

private void BButton_Click(object sender, RoutedEventArgs e)
{
    var item = BComboBox.SelectedItem as Customer;
    if (item != null)
    {
        var sb = new StringBuilder();
        sb.AppendLine("AComboBox.SelectedIndex:" + BComboBox.SelectedIndex);
        sb.AppendLine("AComboBox.SelectedValue:" + BComboBox.SelectedValue);
        sb.AppendLine("AComboBox.Text:" + BComboBox.Text);
        sb.AppendLine("-------");
        sb.AppendLine("SelectedItem.Id : " + item.Id);
        sb.AppendLine("SelectedItem.Name : " + item.Name);
        sb.AppendLine("SelectedItem.Phone : " + item.Phone);
        MessageBox.Show(sb.ToString());
    }
}
```

実行結果



- 複数の TextBlock が表示されています。

2 番目の値を選択して「check」を押下したとき



- SelectedIndex には「 1 」が設定されています。
- SelectedValue と Text には「WPF020.Customer」という文字が表示されています。これは Customer クラスを ToString()したときの文字列です。今回は SelectedValuePath と DisplayMemberPath を設定していないためこうなっています。SelectedValuePath を Id にしておくことで，SelectedValue を Id で取得することは可能です。

- SelectedItem には選択されている Customer の値が取得できています。この値を常に参照するという事であれば，SelectedValuePath を設定しなくてもコーディングは可能です。

サンプルコード全体

Customer.cs

```
namespace WPF020
{
    public class Customer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Phone { get; set; }
    }
}
```

MainWindow.xaml

```
<Window x:Class="WPF020.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF020"
        mc:Ignorable="d"
        Title="MainWindow" Height="240" Width="300">
    <Grid>
        <StackPanel>
            <StackPanel Orientation="Horizontal">
                <ComboBox x:Name="MyComboBox"
                          Height="40"
                          Width="200"
                          Margin="10"
                          FontSize="20"
                          VerticalAlignment="Center"/>
                <Button Margin="10"
                        Width="50"
                        Content="check"
                        Click="MyButton_Click"
                        />

            </StackPanel>

            <StackPanel Orientation="Horizontal">
                <ComboBox x:Name="AComboBox"
                          Height="40"
                          Width="200"
                          Margin="10"
                          FontSize="20"
                          VerticalAlignment="Center"
                          SelectedValuePath="Id"
```

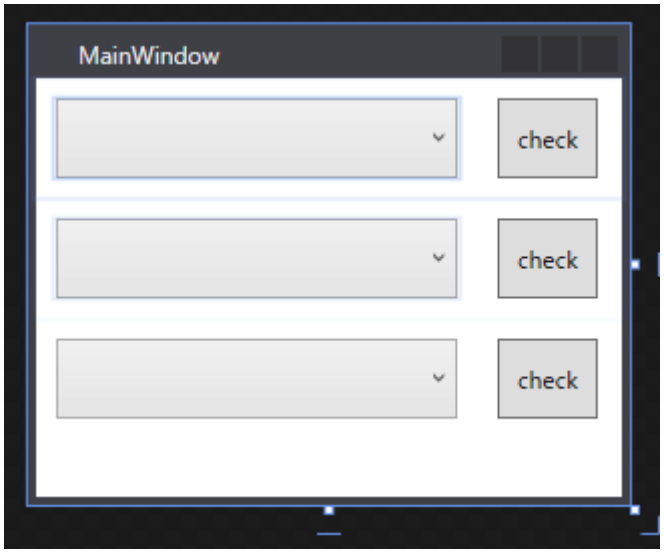
```
                          DisplayMemberPath="Name"/>
                <Button Margin="10"
                        Width="50"
                        Content="check"
                        Click="AButton_Click"
                        />

            </StackPanel>

            <StackPanel Orientation="Horizontal">
                <ComboBox x:Name="BComboBox"
                          Height="40"
                          Width="200"
                          Margin="10"
                          FontSize="20"
                          VerticalAlignment="Center">
                    <ComboBox.ItemTemplate>
                        <DataTemplate>
                            <StackPanel Orientation="Horizontal">
                                <TextBlock Text="{Binding Id}" FontSize="20" Margin="5"/>
                                <TextBlock Text="{Binding Name}" FontSize="20" Margin="5"/>
                                <TextBlock Text="{Binding Phone}" FontSize="20" Margin="5"/>
                            </StackPanel>
                        </DataTemplate>
                    </ComboBox.ItemTemplate>
                </ComboBox>
                <Button Margin="10"
                        Width="50"
                        Content="check"
                        Click="BButton_Click"
                        />

            </StackPanel>
        </StackPanel>
    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System.Collections.ObjectModel;
using System.Text;
using System.Windows;

namespace WPF020
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        private ObservableCollection<Customer> _customers
            = new ObservableCollection<Customer>();
        public MainWindow()
        {
            InitializeComponent();
```

```
        MyComboBox.Items.Add("111");
        MyComboBox.Items.Add("222");
        MyComboBox.Items.Add("333");

        _customers.Add(new Customer { Id = 1, Name = "name1", Phone = "Phone1" });
        _customers.Add(new Customer { Id = 2, Name = "name2", Phone = "Phone2" });
        _customers.Add(new Customer { Id = 3, Name = "name3", Phone = "Phone3" });

        AComboBox.ItemsSource = _customers;
        BComboBox.ItemsSource = _customers;
    }

    private void MyButton_Click(object sender, RoutedEventArgs e)
    {
        var sb = new StringBuilder();
        sb.AppendLine("MyComboBox.SelectedIndex:" + MyComboBox.SelectedIndex);
        sb.AppendLine("MyComboBox.SelectedValue:" + MyComboBox.SelectedValue);
        sb.AppendLine("MyComboBox.Text:" + MyComboBox.Text);
        MessageBox.Show(sb.ToString());
    }

    private void AButton_Click(object sender, RoutedEventArgs e)
    {
        var item = AComboBox.SelectedItem as Customer;
        if (item != null)
        {
            var sb = new StringBuilder();
            sb.AppendLine("AComboBox.SelectedIndex:" + AComboBox.SelectedIndex);
            sb.AppendLine("AComboBox.SelectedValue:" + AComboBox.SelectedValue);
            sb.AppendLine("AComboBox.Text:" + AComboBox.Text);
            sb.AppendLine("-------");
            sb.AppendLine("SelectedItem.Id : " + item.Id);
            sb.AppendLine("SelectedItem.Name : " + item.Name);
            sb.AppendLine("SelectedItem.Phone : " + item.Phone);
            MessageBox.Show(sb.ToString());
        }
    }

    private void BButton_Click(object sender, RoutedEventArgs e)
    {
        var item = BComboBox.SelectedItem as Customer;
        if (item != null)
        {
            var sb = new StringBuilder();
            sb.AppendLine("AComboBox.SelectedIndex:" + BComboBox.SelectedIndex);
            sb.AppendLine("AComboBox.SelectedValue:" + BComboBox.SelectedValue);
            sb.AppendLine("AComboBox.Text:" + BComboBox.Text);
            sb.AppendLine("-------");
            sb.AppendLine("SelectedItem.Id : " + item.Id);
            sb.AppendLine("SelectedItem.Name : " + item.Name);
            sb.AppendLine("SelectedItem.Phone : " + item.Phone);
            MessageBox.Show(sb.ToString());
        }
```

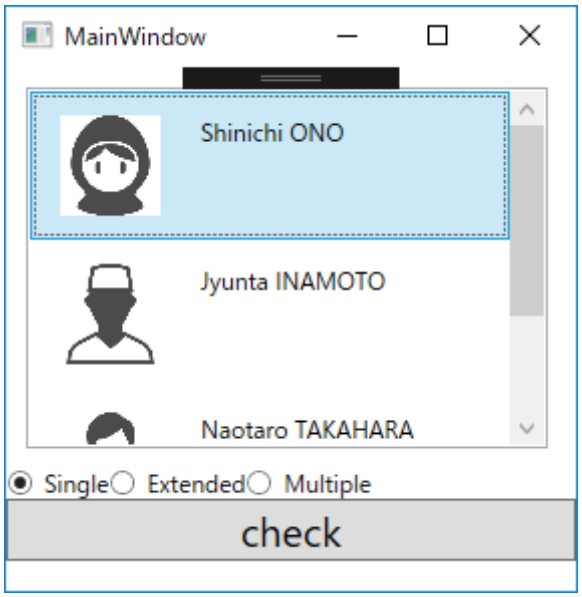
```
            }
        }
}
```


MainWindow
check
check
check

#21　ListBox（リストボックス）

ListBox（リストボックス）とは？

ListBox とは，一覧表示から任意の行を選択できるコントロールです。
選択は単一でも，複数でも可能です。



ListBox の使い方

ListBox 上のコントロールを追加して並べることもできますが，基本的には一覧表示に使うので，データバインディングをして使うことになると思います。データバインディングの方法は，ListView や ComboBox と同様に ItemTemplate や DataTemplate を使って実装します。

SelectionMode

ListView や ComboBox と異なる点として，SelectionMode というプロパティがあるので，その部分を説明します。SelectionMode は，ListBox の項目を単一で選択させるか，複数選択を可能とするかの設定です。設定は次の 3 パターンになります。

Single

1 つの項目のみ選択可能となります。

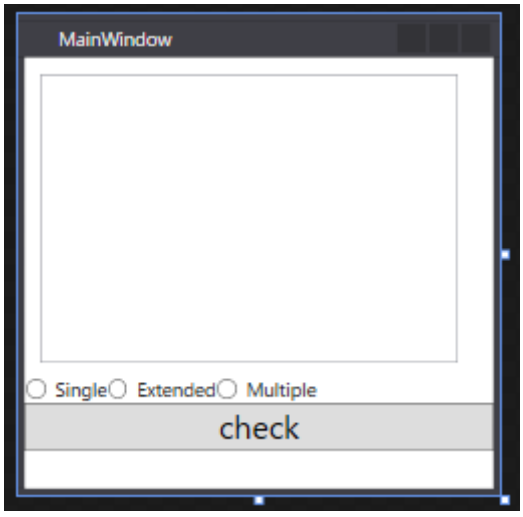
Extended

キーボードの Control キーや，Shift キーを押している最中にクリックをすると，複数選択が可能となります。使い方は Excel の行選択をするときと同じで，Control を押しながらであれば，クリックした行を選択でき，Shift を押している場合は，最初に選択した行から，その次に選択した行までを，まとめて選択することができます。

Multiple

キーボードのキーを押さなくても，複数選択が可能となります。クリックした行が選択状態になり，もう一度クリックすると，選択が解除されます。

サンプルコード



MainWindow.xaml

```
<Window x:Class="WPF021.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF021"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel>
            <ListBox x:Name="MyListBox"
                     HorizontalAlignment="Left"
                     VerticalAlignment="Top"
                     Margin="10"
                     Width="260"
                     Height="180">
```

```
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <StackPanel Orientation="Horizontal">
                            <Image Source="{Binding FileName}"
                                   Width="50"
                                   Height="50"
                                   Margin="10"
                                   />
                            <TextBlock Text="{Binding Name}"
                                       Margin="10"
                                       />
                        </StackPanel>
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>
            <StackPanel Orientation="Horizontal">
                <RadioButton x:Name="SingleRadioButton" Content="Single"
Checked="RadioButton_Checked"/>
                <RadioButton x:Name="ExtendedRadioButton" Content="Extended"
Checked="RadioButton_Checked"/>
                <RadioButton x:Name="MultipleRadioButton" Content="Multiple"
Checked="RadioButton_Checked"/>
            </StackPanel>
            <Button FontSize="20"
                    Content="check"
                    Click="Button_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

- ListBox.ItemTemplate と DataTemplate を使用して，ListBox にデータバインディングするレイアウトを作成しています。
- 1 行分のレイアウトは画像と名前を表示するようにしています。
- ListBox の下に，ラジオボタンを 3 つ設置し，SelectionMode を選択できるようにしています。
- RadioButton の下にボタンを設置し，クリックイベントにブレークポイントを置くことで，ListBox の選択状態のときの中身を確認するために使用します。

MainWindow.xaml.cs

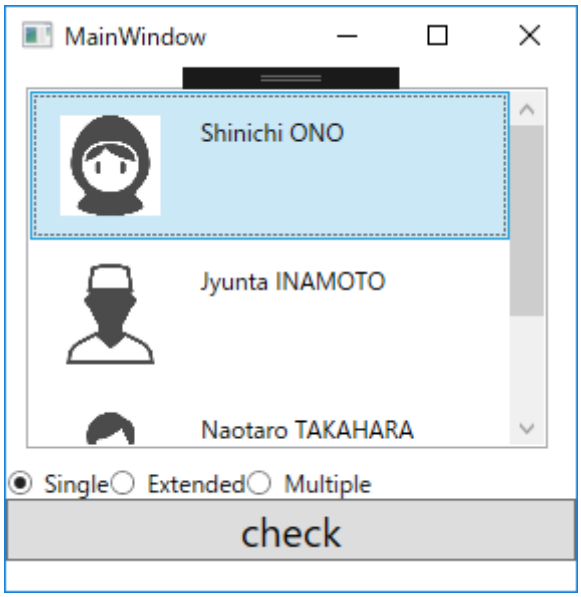
```
using System.Collections.ObjectModel;
using System.Windows;
using System.Windows.Controls;

namespace WPF021
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
```

```
        private ObservableCollection<Dto> _dtos = new ObservableCollection<Dto>();

        public MainWindow()
        {
            InitializeComponent();

            _dtos.Add(new Dto("Images/A.jpeg", "Shinichi ONO"));
            _dtos.Add(new Dto("Images/B.jpeg", "Jyunta INAMOTO"));
            _dtos.Add(new Dto("Images/C.jpeg", "Naotaro TAKAHARA"));
            MyListBox.ItemsSource = _dtos;
            SingleRadioButton.IsChecked = true;
        }

        private void RadioButton_Checked(object sender, RoutedEventArgs e)
        {
            if(SingleRadioButton.IsChecked.Value)
            {
                MyListBox.SelectionMode = SelectionMode.Single;
            }
            else if (ExtendedRadioButton.IsChecked.Value)
            {
                MyListBox.SelectionMode = SelectionMode.Extended;
            }
            else
            {
                MyListBox.SelectionMode = SelectionMode.Multiple;
            }
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {

        }
    }

    public sealed class Dto
    {
        public Dto(string fileName, string name)
        {
            FileName = fileName;
            Name = name;
        }

        public string FileName { get; set; }
        public string Name { get; set; }
    }
}
```

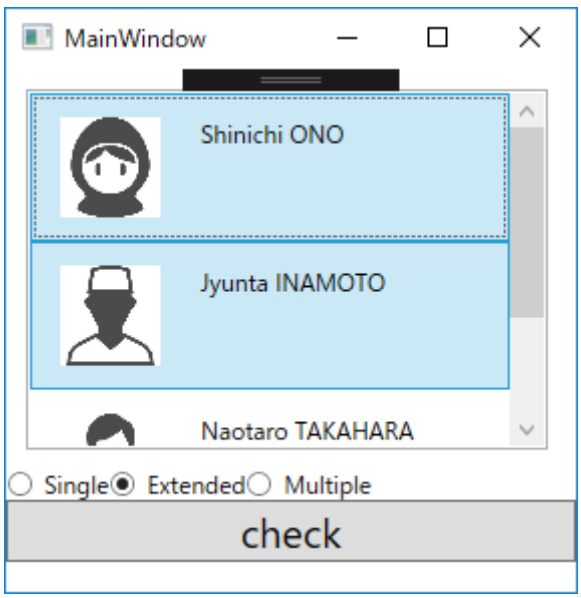
- 一番下に，別のクラスとして Dto クラスを作成しています。内容は画面のファイル名と名称の 2 項目です。このクラスのリストをデータバインドします。
- Private フィールドに Dto のリストを生成しています。
- Dto のリストに 3 件のデータを作成しています。
- ListBox の ItemsSource に Dto のリストを設定しています。

- ラジオボタンの初期値は Single を選択しています。
- ラジオボタンのチェック状態変更イベントに応じて，ListBox の SelectionMode を変更しています。

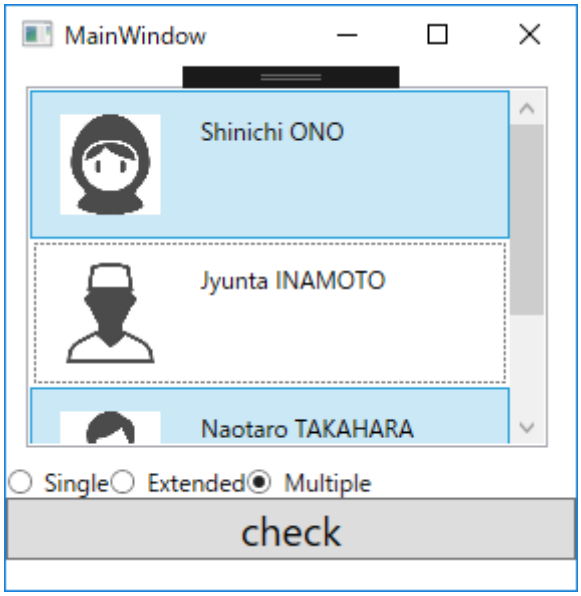
実行結果



- Single モードのときは 1 行しか選択できません。

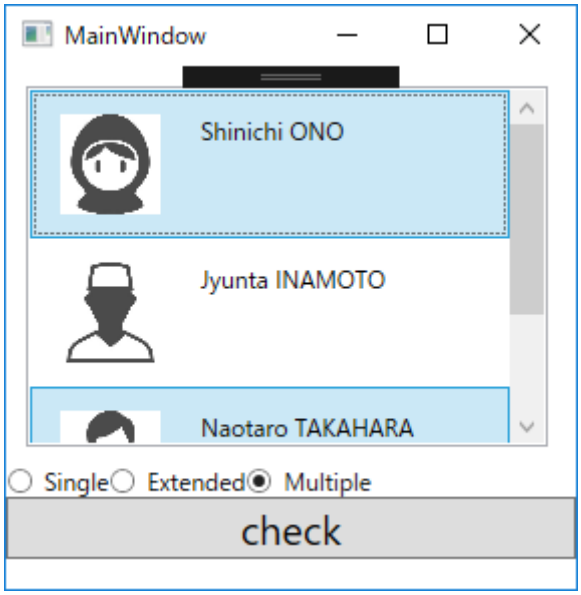


- Extended モードのときは，Control か Shift を押して複数選択が可能です。

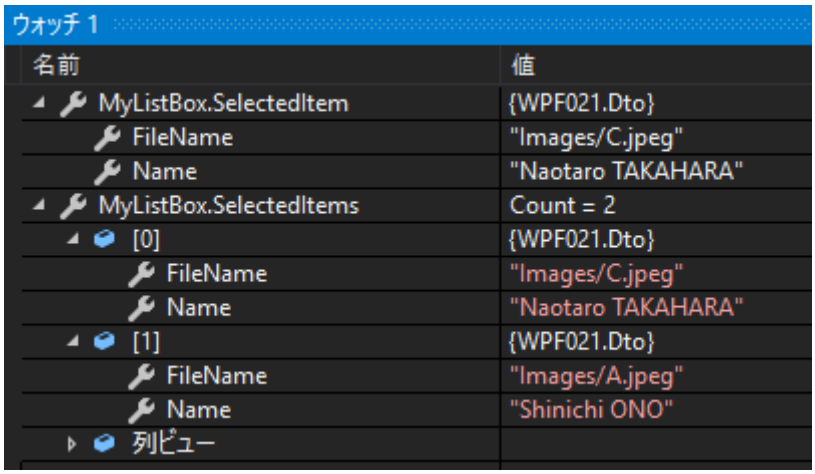


Multiple モードのときは，キーボードを押さずにクリックのみで，複数選択が可能です。選択されている行をクリックすると，選択が解除されます。

選択中の SelectionItems の中身



- 「check」ボタンのクリックイベントにブレークポイントを置きます。
- 3 行目を選択した後，1 行目を選択した状態で「check」ボタンを押下します。
- ブレークポイントでデバッグが停止されている状態で，ListBox の SelectedItem と SelectionItems の中身を確認してみましょう。

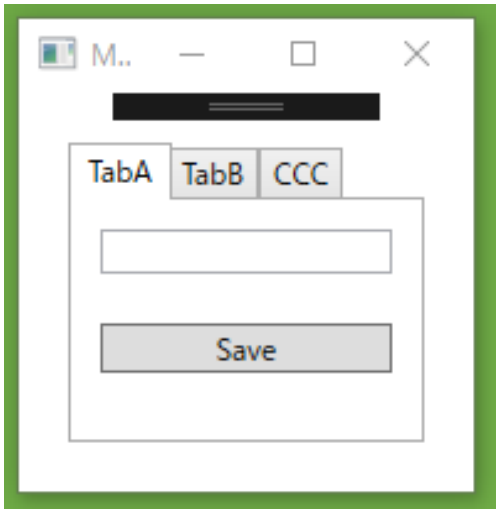


- SelectedItem には 3 行目の値が入っています。最初に選択された行が格納されていることがわかります。
- SelectionItems の中身には 3 行目，1 行目の順番でデータが格納されています。行を選択した順番で格納されていることがわかります。

#22　TabControl（タブコントロール）

TabControl（タブコントロール）とは？

TabControl とは，ヘッダー付きのページを複数作れるコントロールです。画面のエリアに対して，配置する必要のあるコントロールが多い場合や，グループ分けして入力させたい場合に使用します。



TabControl の書き方

TabControl は，TabItem の数だけページが作られます。各ページのタイトルは TabItem の Header に設定します。

書き方の例
<TabControl> 　　<TabItem Header="TabA"> 　　</TabItem> </TabControl>

主なプロパティ

ソースコード上から表示されているページを切り替える場合等は「SelectedIndex」を変更します。ページの数だけインデックスがあり，1 ページ目はインデックス「0」から始まります。そのため，2 ページ目を選択する場合は「1」と指定します。

サンプルコード

MainWindow.xaml

```
<Window x:Class="WPF022.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF022"
        mc:Ignorable="d"
        Title="MainWindow" Height="200" Width="200">
    <Grid>
        <TabControl x:Name="MyTabControl"
                    Margin="20">
            <TabItem Header="AAA">
                <StackPanel>
                    <TextBox Margin="10"/>
                    <Button Content="Save"
                            Margin="10"/>
                </StackPanel>
            </TabItem>
            <TabItem Header="BBB">
                <StackPanel>
                    <TextBox Margin="10"/>
                    <Button Content="Search"
                            Margin="10"/>
                </StackPanel>
            </TabItem>
        </TabControl>
    </Grid>
</Window>
```

- TabControl に TabItem を 2 つ設定しています。
- タイトルはそれぞれ「AAA」と「BBB」になります。
- TabItem の中に StackPanel などを設置し，任意のコントロールを配置します

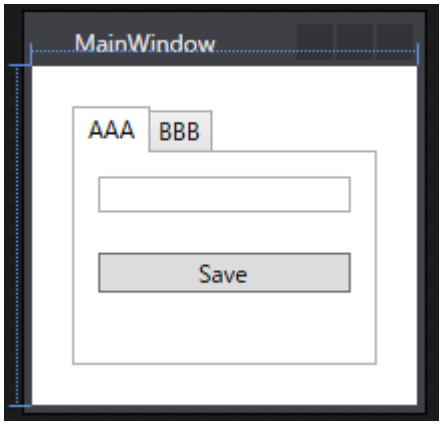
MainWindow.xaml.cs

```
using System.Windows;

namespace WPF022
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            MyTabControl.SelectedIndex = 1;
        }
```

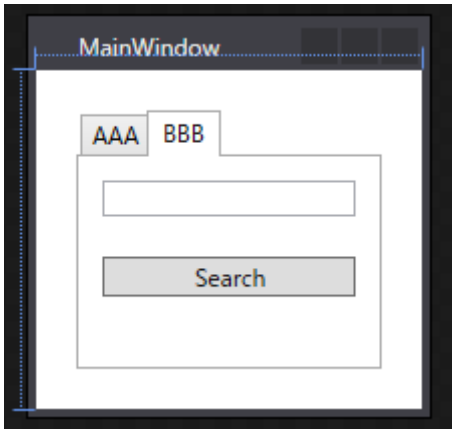
```
    }
}
```

- MyTabControl の SelectedIndex を指定し，タブコントロールの表示されるページを変更しています。

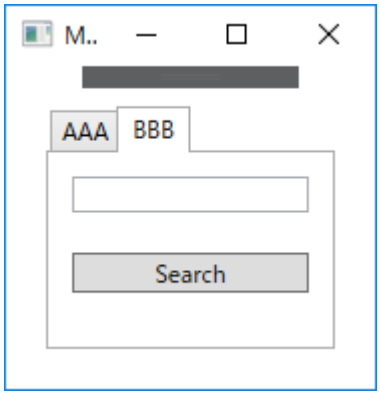
Xaml のレイアウト「AAA」選択時



Xaml のレイアウト「BBB」選択時



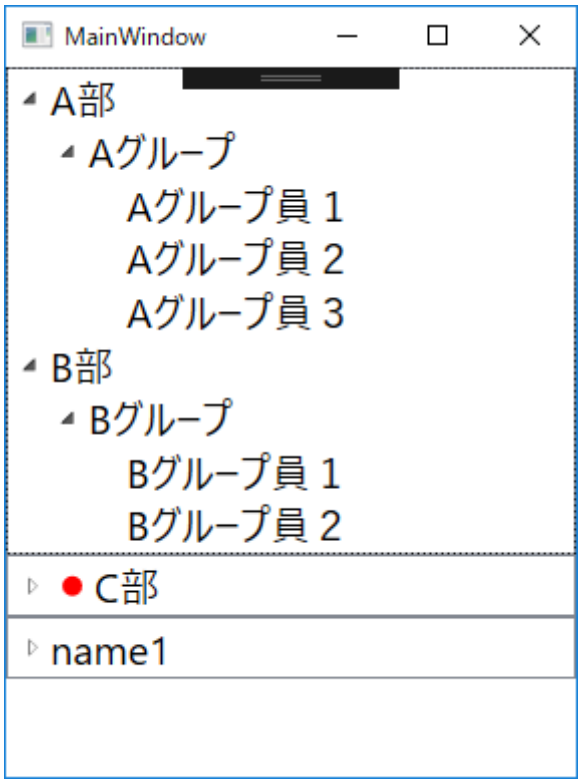
実行結果



#23　TreeView

TreeView とは？

TreeView とは，階層構造を表すことのできるコントロールです。ちょうど Windows のファイルエクスプローラーのような感じです。フォルダーの下にフォルダーがあって，その下にファイルがあるみたいな感じで，階層構造になっていますよね。そういう親のしたに子がいて，さらにその子にも子階層があるというような構造をツリー構造（木構造）といい，それを見た目に表すことのできるコントロールという事になります。



TreeView の書き方

TreeView の書き方には，あらかじめ表示する内容が決まっている場合等に利用できる「静的」なやり方と，データバインドを使用した「動的」な設定の仕方があります。

静的な書き方

Xaml で TreeView の中に TreeViewItem を作ることで，一つの階層を作ることができます。その TreeViewItem にもさらに TreeViewItem を作成することができ，その繰り返しにより，親子関係を定義します。階層に表示される文言は Header に設定します。

TreeViewItem を用いた階層の例

```
<TreeView FontSize="20">
    <TreeViewItem Header="A 部">
        <TreeViewItem Header="A グループ">
            <TreeViewItem Header="グループ員１"/>
            <TreeViewItem Header="グループ員２"/>
            <TreeViewItem Header="グループ員３"/>
        </TreeViewItem>
    </TreeViewItem>
</TreeView>
```

- TreeViewItem の中に TreeViewItem を作ることで，親子階層になる
- TreeViewItem の Header が階層のタイトルになる

実行例

TreeViewItem.Header を利用することで，文言の部分をマーク付きの文言にするなどの加工が簡単に行えます。

TreeViewItem.Header を使用した例

```
<TreeView FontSize="20">
    <TreeViewItem>
        <TreeViewItem.Header>
            <StackPanel Orientation="Horizontal">
                <Ellipse Height="10"
                         Width="10"
                         Margin="6"
```

```
                        Fill="Red"/>
                <TextBlock Text="B 部"/>
            </StackPanel>
        </TreeViewItem.Header>
        <TreeViewItem Header="グループ員１"/>
    </TreeViewItem>
</TreeView>
```

- TreeViewItem に直接 Header を指定せず，TreeViewItem.Header を定義
- TreeViewItem.Header に StackPanel を定義し，任意のコントロールを配置
- 赤い丸の右にテキストブロックで文言を表示

実行例

データバインディングを使った動的な設定例

MainWindow.xaml.cs

```
/// <summary>
/// MainWindow.xaml の相互作用ロジック
/// </summary>
public partial class MainWindow : Window
{
    private ObservableCollection<Dto> _dtos = new ObservableCollection<Dto>();
    public MainWindow()
    {
        InitializeComponent();

        var dto1 = new Dto("Name1");
        dto1.Dtos.Add(new Dto("Name1-1"));
        dto1.Dtos.Add(new Dto("Name1-2"));
        _dtos.Add(dto1);
```

```
        CTreeView.ItemsSource = _dtos;
    }
}

public sealed class Dto
{
    public Dto(string name)
    {
        Name = name;
    }

    public string Name { get; set; }
    public List<Dto> Dtos { get; set; } = new List<Dto>();
}
```

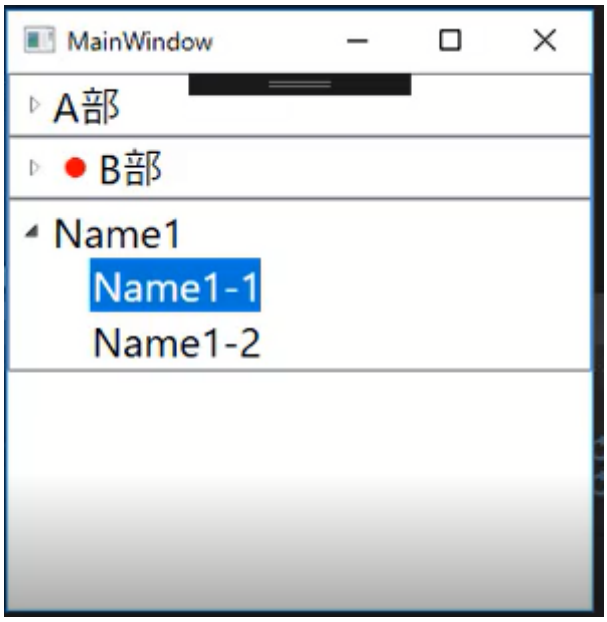
- まずデータバインディングを行うための型となるクラスを作成します。ここでは Dto としています。
- Dto は階層名を表示するための Name プロパティと，自分自身の子階層を表す Dtos のプロパティがあります。
- Dtos のプロパティは，親階層と同じ Dto クラスの List になっています。
- Private フィールドに Dto のリストを生成しています（_dtos)。これが TreeView の ItemsSource となります。
- dto1 を生成しています。これは親階層の Name を「Name1」とし，その子階層を「Name1-1」「Name1-2」としています。
- dto1 を_dtos に Add し，それを CTreeView の ItemsSource にセットしています。

MainWindow.xaml

```
<TreeView x:Name="CTreeView" FontSize="20">
    <TreeView.ItemTemplate>
        <HierarchicalDataTemplate DataType="local:Dto"
                                  ItemsSource="{Binding Dtos}">
            <TextBlock Text="{Binding Name}"/>
        </HierarchicalDataTemplate>
    </TreeView.ItemTemplate>
</TreeView>
```

- TreeView の名前を CTreeView にしています。
- TreeView.ItemTemplate と HierarchicalDataTemplate のエリアを作成
- HierarchicalDataTemplate の DataType に階層の型となるクラス名「Dto」を指定します
- ItemsSource には Dto の子階層のプロパティ名を指定します「Dtos」
- 階層名として表示するエリアを TextBlock で表示するために Name を Binding します

実行結果


MainWindow
A部
B部
Name1
Name1-1
Name1-2

サンプルコード全体

MainWindow.xaml

```
<Window x:Class="WPF023.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF023"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel>
            <TreeView FontSize="20">
                <TreeViewItem Header="A 部">
                    <TreeViewItem Header="A グループ">
                        <TreeViewItem Header="グループ員１"/>
                        <TreeViewItem Header="グループ員２"/>
                        <TreeViewItem Header="グループ員３"/>
                    </TreeViewItem>
                </TreeViewItem>
            </TreeView>

            <TreeView FontSize="20">
                <TreeViewItem>
                    <TreeViewItem.Header>
                        <StackPanel Orientation="Horizontal">
                            <Ellipse Height="10"
                                     Width="10"
                                     Margin="6"
                                     Fill="Red"/>
                            <TextBlock Text="B 部"/>
                        </StackPanel>
                    </TreeViewItem.Header>
                    <TreeViewItem Header="グループ員１"/>
                </TreeViewItem>
            </TreeView>

            <TreeView x:Name="CTreeView" FontSize="20">
                <TreeView.ItemTemplate>
                    <HierarchicalDataTemplate DataType="local:Dto"
                                              ItemsSource="{Binding Dtos}">
                        <TextBlock Text="{Binding Name}"/>
                    </HierarchicalDataTemplate>
                </TreeView.ItemTemplate>
            </TreeView>
        </StackPanel>
    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
```

```
using System.Windows;

namespace WPF023
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow : Window
    {
        private ObservableCollection<Dto> _dtos = new ObservableCollection<Dto>();
        public MainWindow()
        {
            InitializeComponent();

            var dto1 = new Dto("Name1");
            dto1.Dtos.Add(new Dto("Name1-1"));
            dto1.Dtos.Add(new Dto("Name1-2"));
            _dtos.Add(dto1);

            CTreeView.ItemsSource = _dtos;
        }
    }

    public sealed class Dto
    {
        public Dto(string name)
        {
            Name = name;
        }

        public string Name { get; set; }
        public List<Dto> Dtos { get; set; } = new List<Dto>();
    }
}
```

#24　Text について（TextBlock,TextBox）

ここでは TextBlock や TextBox などで文字列を表示するときの文字の加工や，改行のモードについて解説していきます。

文字の加工

FontSize

文字の大きさは FontSize で設定します。

FontWeight

文字を太文字にする場合等は FontWeight にて設定します。

FontStyle

文字を普通に表示するのか，斜めに表示するのかを設定します。

TextTrimming

文字がエリア内に表示できないときに，文字の語尾に「...」を表示することができます。

CharacterEllipsis を選択すると，表示できるぎりぎりまで表示して「...」が表示されます。
WordEllipsis を選択すると，単語全体が表示できる場所まで表示して「...」が表示されます。
単語の途中で切りたくない場合はこちらを選択します。

改行指定

TextBlock で意図的に改行コードを入れる場合は Run と LineBreak で調整します。
次の例では AAA と BBB の間で改行されます。

```
<TextBlock FontSize="20"
           Width="120">
    <Run Text="AAA"/>
    <LineBreak/>
    <Run Text="BBB"/>
</TextBlock>
```

改行の方法

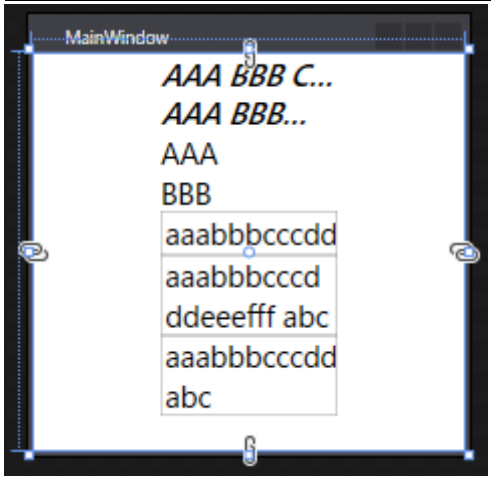
TextWrapping で，改行の方法が指定できます。
Wrap を指定すると単語の切れ目に関係なく，エリアの幅に合わせて改行されます。
WrapWithOverflow を選択すると，単語の途中では改行されません。そのため，単語がすべて表示されない場合があります。

サンプルコード

MainWindow.xaml

```
<Window x:Class="WPF024.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF024"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <StackPanel>
            <TextBlock Text="AAA BBB CCC DDD EEE FFF"
                       FontSize="20"
                       Width="120"
                       FontWeight="Bold"
                       FontStyle="Italic"
                       TextTrimming="CharacterEllipsis"
                       />

            <TextBlock Text="AAA BBB CCC DDD EEE FFF"
```

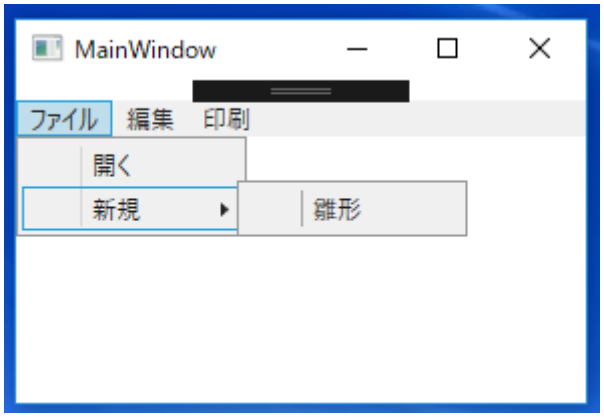
```
                        FontSize="20"
                        Width="120"
                        FontWeight="Bold"
                        FontStyle="Italic"
                        TextTrimming="WordEllipsis"
                        />

            <TextBlock FontSize="20"
                       Width="120">
                <Run Text="AAA"/>
                <LineBreak/>
                <Run Text="BBB"/>
            </TextBlock>

            <TextBox FontSize="20"
                     Width="120"
                     Text="aaabbbcccdddeeefff abc"
                     TextWrapping="NoWrap"/>

            <TextBox FontSize="20"
                     Width="120"
                     Text="aaabbbcccdddeeefff abc"
                     TextWrapping="Wrap"/>

            <TextBox FontSize="20"
                     Width="120"
                     Text="aaabbbcccdddeeefff abc"
                     TextWrapping="WrapWithOverflow"/>
        </StackPanel>
    </Grid>
</Window>
```



#25　Menu

Menu とは？

Menu とは，画面の上部とかによくある「ファイル」とは「開く」とかいったものを選択する時に使うものです。Excel や Word などでも画面の上部に必ずメニューが付いていますよね。その Menu の設定の仕方を解説していきます。



Menu の書き方

MenuItem

Menu を設定する場合は「MenuItem」というものを指定します。一つの MenuItem が「ファイル」などの一つのメニューになります。

Header

Header にメニューに表示する「ファイル」等の文言を指定しましょう。

MainWindow.xaml

```
<Menu>
    <MenuItem Header="編集"/>
</Menu>
```

子階層を作る方法

子階層を作るときは MemuItem の中に，さらに MenuItem を設定することで実現できます。その子階層に，さらに MenuItem を指定していくことも可能です。

MainWindow.xaml

```
<Menu>
    <MenuItem Header="ファイル">
        <MenuItem Header="開く"/>
        <MenuItem Header="新規">
            <MenuItem Header="雛形"/>
        </MenuItem>
    </MenuItem>
</Menu>
```

クリックイベント

すべての MenuItem に対して Click イベントを設定できるので，そこに，クリックされたときの処理を記述します。

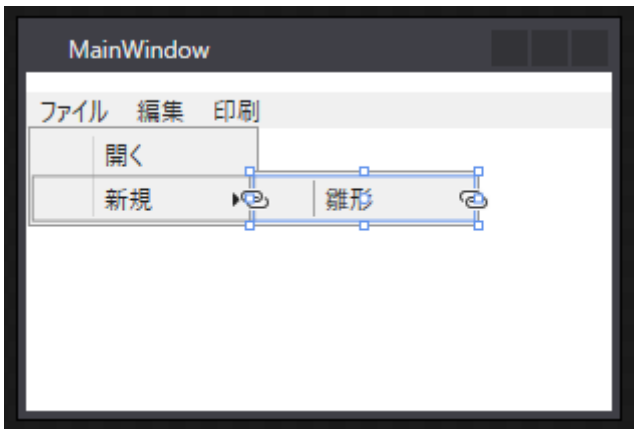
MainWindow.xaml

```
<Menu>
    <MenuItem Header="印刷" Click="MenuItem_Click"/>
</Menu>
```

サンプルコード全体

MainWindow.xaml

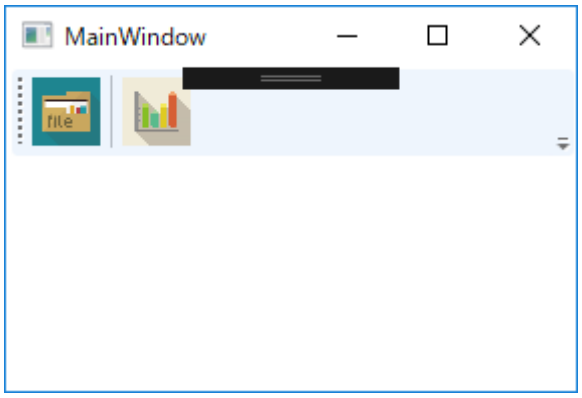
```
<Window x:Class="WPF025.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF025"
        mc:Ignorable="d"
        Title="MainWindow" Height="200" Width="300">
    <Grid>
        <StackPanel Margin="0 10 0 0">
            <Menu>
                <MenuItem Header="ファイル">
                    <MenuItem Header="開く"/>
                    <MenuItem Header="新規">
                        <MenuItem Header="雛形"/>
                    </MenuItem>
                </MenuItem>
                <MenuItem Header="編集"/>
                <MenuItem Header="印刷" Click="MenuItem_Click"/>
            </Menu>
        </StackPanel>
    </Grid>
</Window>
```


MainWindow
ファイル
編集
印刷
開く
新規
雛形

#26　ToolBar

ToolBar とは？

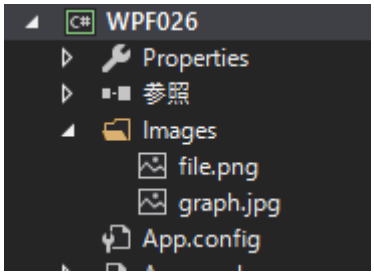
ToolBar とは，画面の上部などにアイコンをならべて，「コピー」とか「保存」とかを行うことのできるボタンの集まりです。Excel とか Word とか，その他いろんなアプリについいてますよね。今回は，ツールバーの作り方を解説します。



画像の準備

事前に画像の準備をしてください。画像は何でも構いません。ネットなどで「アイコン　フリー画像」などで検索すれば，無料で使える画像がいくらでもあります。

まず WPF のプロジェクトを作って，その下にフォルダーを作ります。今回は「Images」という名前にしておきます。その中にツールバーで使う画像を入れておきましょう。



ツールバーの書き方

ツールバーは ToolBar のエリアの中に Button を置いて，ボタンにイメージを設定します。Image Source には，先ほど準備したファイルのパスを指定します。

MainWindow.xaml
<ToolBar Height="45"

```
         VerticalAlignment="Top">
    <Button>
        <Image Source="Images¥file.png"/>
    </Button>
</ToolBar>
```

クリックイベント

クリックイベントは，Button のクリックイベントを実装しておきます。

MainWindow.xaml

```
<ToolBar Height="45"
         VerticalAlignment="Top">
    <Button Click="Button_Click">
        <Image Source="Images¥graph.jpg"/>
    </Button>
</ToolBar>
```

Separator

ツールボタンをグループ単位で仕切りたい場合は Separator で区切ることができます。これを入れると，縦の線を引っ張ることができます。

MainWindow.xaml

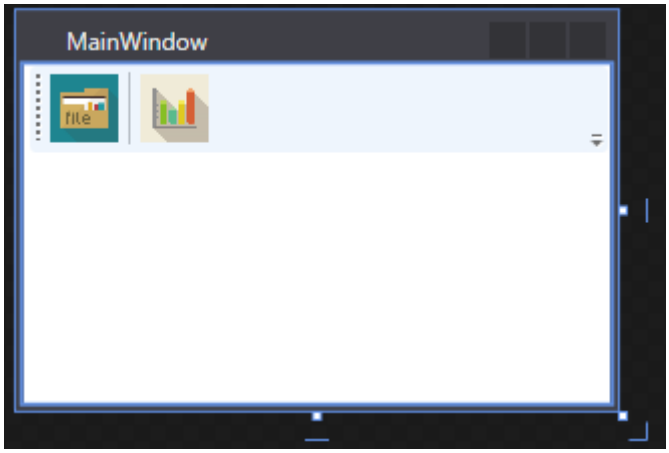
```
<Button>
    <Image Source="Images¥file.png"/>
</Button>
<Separator/>
<Button Click="Button_Click">
    <Image Source="Images¥graph.jpg"/>
</Button>
```

サンプルソース全体

MainWindow.xaml

```
<Window x:Class="WPF026.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF026"
        mc:Ignorable="d"
        Title="MainWindow" Height="200" Width="300">
    <Grid>
        <ToolBar Height="45"
                 VerticalAlignment="Top">
            <Button>
                <Image Source="Images¥file.png"/>
            </Button>
            <Separator/>
```

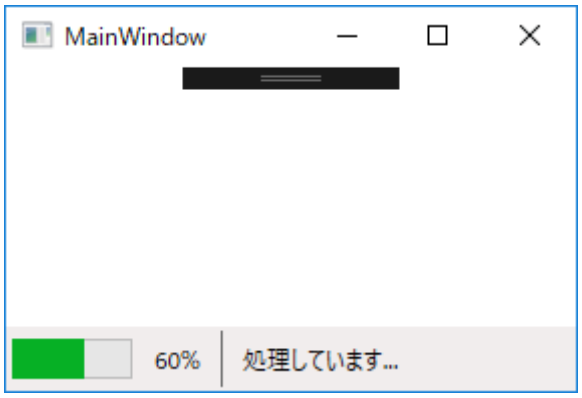
```
            <Button Click="Button_Click">
                <Image Source="Images¥graph.jpg"/>
            </Button>
        </ToolBar>
    </Grid>
</Window>
```



#27　StatusBar

StatusBar とは？

StatusBar とは，画面の一番下で，処理の状況やガイダンスを表示するためによく使われるコントロールです。「処理中です…」や「保存しました」などが表示されたり，処理の進捗状況をプログレスバーで表示したりします。



StatusBar の書き方

ステータスバーは「StatusBar」の中に，任意のコントロールを並べることで実現できます。

MainWindow.xaml

```
<StatusBar VerticalAlignment="Bottom">
    <Label Content="処理しています..."/>
</StatusBar>
```

Separator

ステータスバーの中の任意のコントロール同士の間を，縦線で区切って見た目をわかりやすくする場合は Separator を設置します。

MainWindow.xaml

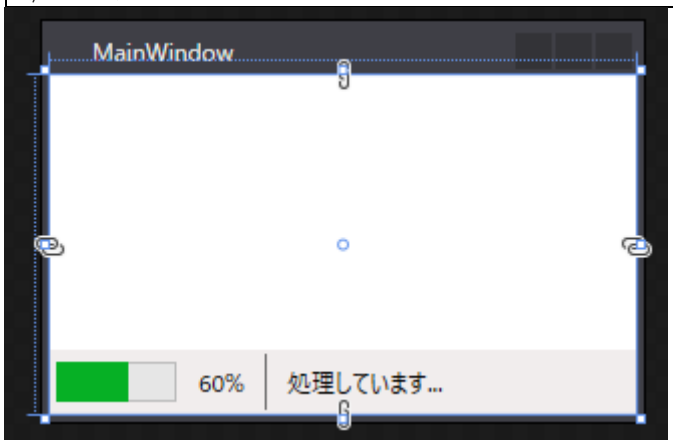
```
<Label Content="60%"/>
<Separator/>
<Label Content="処理しています..."/>
```

サンプルコード全体

MainWindow.xaml

```
<Window x:Class="WPF027.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

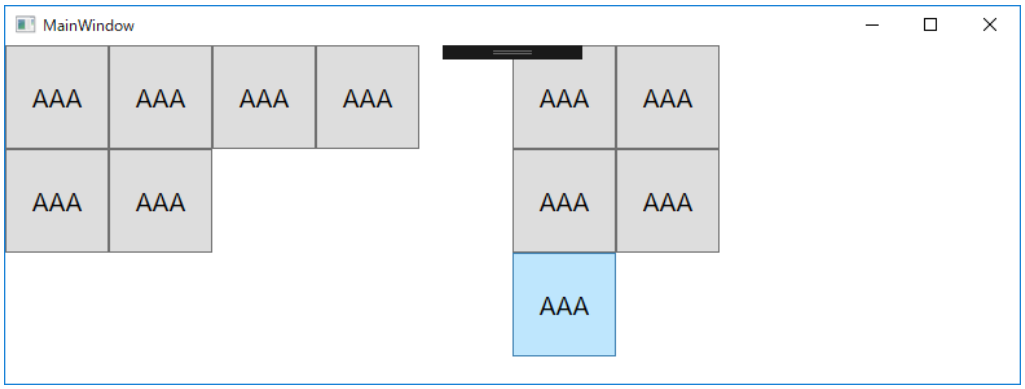
```
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF027"
        mc:Ignorable="d"
        Title="MainWindow" Height="200" Width="300">
    <Grid>
        <StatusBar VerticalAlignment="Bottom">
            <ProgressBar Width="60"
                         Height="20"
                         Value="60"
                         />
            <Label Content="60%"/>
            <Separator/>
            <Label Content="処理しています..."/>
        </StatusBar>

    </Grid>
</Window>
```



#28　WrapPanel

WrapPanel とは？

WrapPanel はコントロールを置くためのパネルで，縦向きか横向きで順番にコントロールを整列して配置してくれます。コントロールを並べていく際に，コントロールがエリアに入りきらなくなったら，折り返しながらコントロールを並べるのが特徴です。



WrapPanel の書き方

WrapPanel は，コントロールを配置していく方向と，一つのコントロールに割り当てる縦と横の幅を指定します。

Orientation

コントロールを縦に並べるときは Vertical, 横方向に並べていく場合は Horizontal を指定します。

コントロール 1 つ分の縦幅を ItemHeight，横幅を ItemWidth で指定します。

横に並べる例

MainWindow.xaml

```
<WrapPanel Orientation="Horizontal"
           ItemHeight="80"
           ItemWidth="80">
    <Button Content="AAA" FontSize="20"/>
    <Button Content="AAA" FontSize="20"/>
    <Button Content="AAA" FontSize="20"/>
    <Button Content="AAA" FontSize="20"/>
    <Button Content="AAA" FontSize="20"/>
```

```
    <Button Content="AAA" FontSize="20"/>
</WrapPanel>
```

縦に並べる例

MainWindow.xaml

```
<WrapPanel Orientation="Vertical"
           ItemHeight="80"
           ItemWidth="80">
    <Button Content="AAA" FontSize="20"/>
    <Button Content="AAA" FontSize="20"/>
    <Button Content="AAA" FontSize="20"/>
    <Button Content="AAA" FontSize="20"/>
    <Button Content="AAA" FontSize="20"/>
</WrapPanel>
```

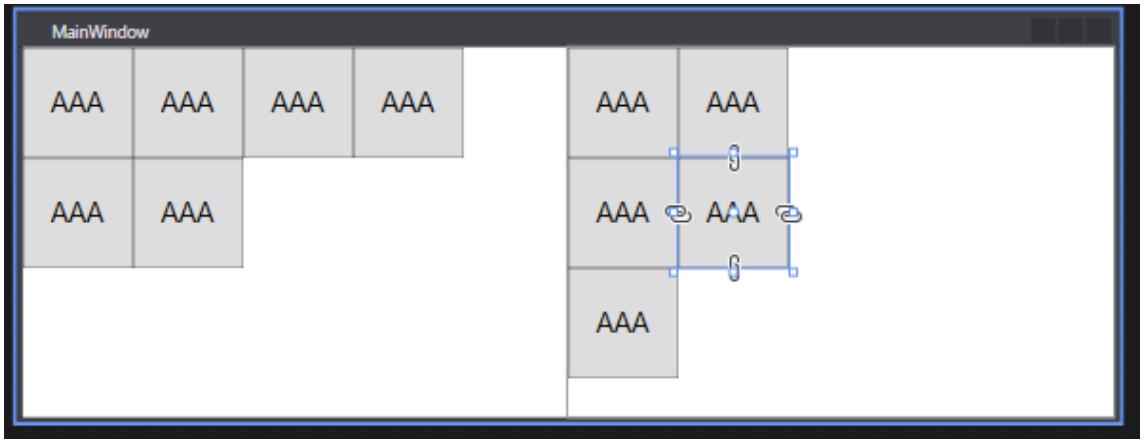
違いは Orientation のみです。

サンプルコード全体

MainWindow.xaml

```
<Window x:Class="WPF028.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF028"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="800">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

        <WrapPanel Orientation="Horizontal"
                   ItemHeight="80"
                   ItemWidth="80">
            <Button Content="AAA" FontSize="20"/>
            <Button Content="AAA" FontSize="20"/>
            <Button Content="AAA" FontSize="20"/>
            <Button Content="AAA" FontSize="20"/>
            <Button Content="AAA" FontSize="20"/>
            <Button Content="AAA" FontSize="20"/>
        </WrapPanel>

        <WrapPanel Grid.Column = "1"
                   Orientation="Vertical"
                   ItemHeight="80"
                   ItemWidth="80">
            <Button Content="AAA" FontSize="20"/>
```

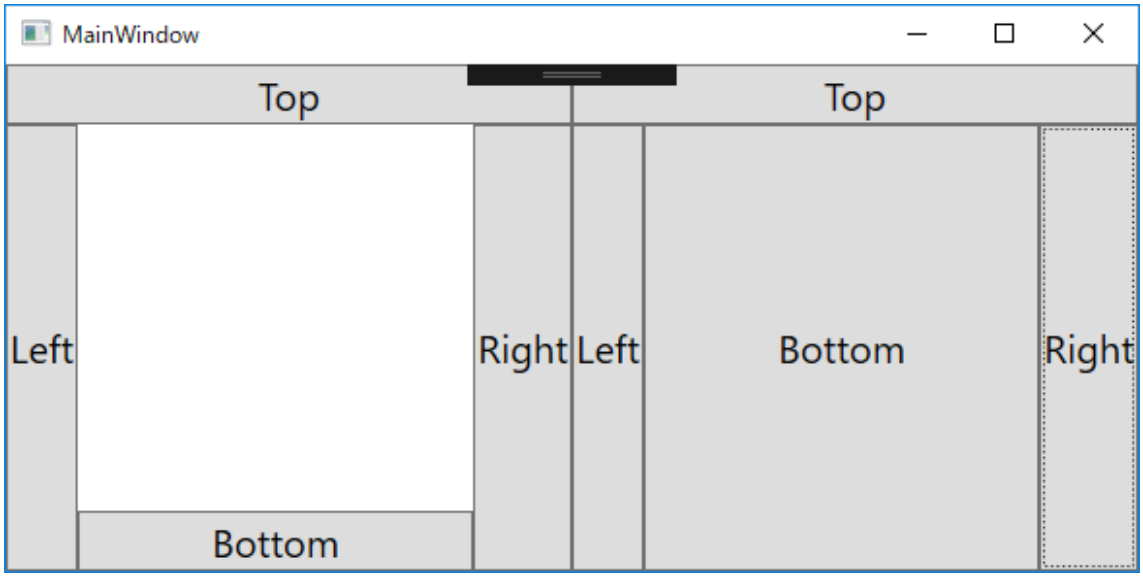
```
            <Button Content="AAA" FontSize="20"/>
            <Button Content="AAA" FontSize="20"/>
            <Button Content="AAA" FontSize="20"/>
            <Button Content="AAA" FontSize="20"/>
        </WrapPanel>
    </Grid>
</Window>
```



#29　DockPanel

DockPanel とは？

枠内の上下左右に整列して配置させるためのコントロールです。



DockPanel の使い方

DockPanel.Dock

DockPanel は DockPanel.Dock を指定して，上下左右のどこに張り付けるかを決めます。

- Top　上に張り付きます
- Left　左に張り付きます
- Right　右に張り付きます
- Bottom　下にはりつきます

LastChildFill

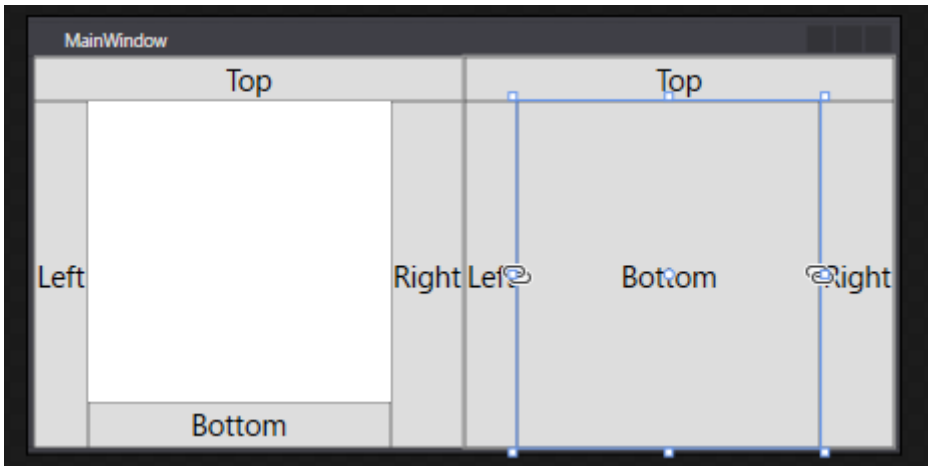
LastChildFill は枠内の最後の要素が，残りのスペースを埋めるかどうかを決定します。
先の画面イメージの左側は，Bottom が下に張り付いていますが，右側は，Bottom が残りのスペースをすべて埋める形で張り付いているのがわかると思います。この場合，右側の例が LastChildFill を True にしたときの例となります。最後のコントロールが，残りのスペースを埋める形で張り付きます。デフォルトは True なので残りのスペースを埋めようとします。
だから，DockPanel を作って，コントロールを 1 つしか貼り付けない場合は，Top と指定

していても，Top ではなく全体に伸びるような感じで張り付きます。コントロール 1 つで Top に張り付けたい場合などは，LastChildFill を False にしておきましょう。それ以外の場合でも，残りのスペースを埋める必要がない場合は，明示的に False にしておきましょう。

サンプルコード全体

MainWindow.xaml

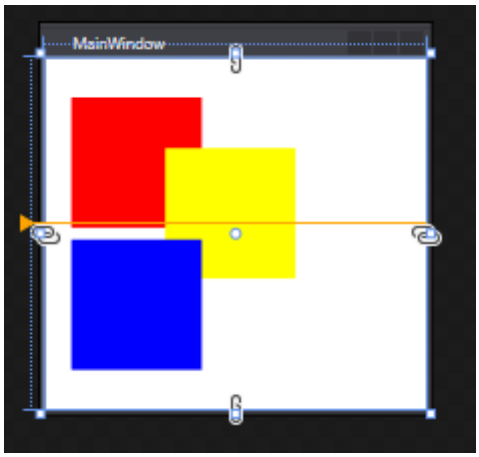
```
<Window x:Class="WPF029.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF029"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="600">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

        <DockPanel LastChildFill="False">
            <Button DockPanel.Dock="Top" Content="Top" FontSize="20"/>
            <Button DockPanel.Dock="Left" Content="Left" FontSize="20"/>
            <Button DockPanel.Dock="Right" Content="Right" FontSize="20"/>
            <Button DockPanel.Dock="Bottom" Content="Bottom" FontSize="20"/>
        </DockPanel>

        <DockPanel Grid.Column="1" LastChildFill="True">
            <Button DockPanel.Dock="Top" Content="Top" FontSize="20"/>
            <Button DockPanel.Dock="Left" Content="Left" FontSize="20"/>
            <Button DockPanel.Dock="Right" Content="Right" FontSize="20"/>
            <Button DockPanel.Dock="Bottom" Content="Bottom" FontSize="20"/>
        </DockPanel>
    </Grid>
</Window>
```



#30　Canvas

Canvas とは？

Canvas とは，コントロールを座標指定で設置していくときに使うものです。「左から 20 ピクセル」「上から 30 ピクセル」の位置に Button を設置する…などと使うことができます。



Canvas の使い方

Canvas を使う場合は，Canvas の中に任意のコントロールを設置して，Canvas.Left，Canvas.Top などの値を指定することで，座標を指定します。
座標の指定は Canvas.Left，Canvas.Top 以外にも Canvas.Right と Canvas.Bottom があります。
通常は Canvas.Left と Canvas.Top を使用することで，左から何ピクセル，上から何ピクセルという形で指定します。右から指定したい場合は Canvas.Right，下から指定したい場合は Canvas.Bottom を使用しても問題ありません。

MainWindow.xaml

```
<Canvas>
    <Rectangle Canvas.Left="20"
               Canvas.Top="20"
               Width="100"
               Height="100"
               Fill="Red"/>
</Canvas>
```

Canvas の中でコントロールが重なり合うとき

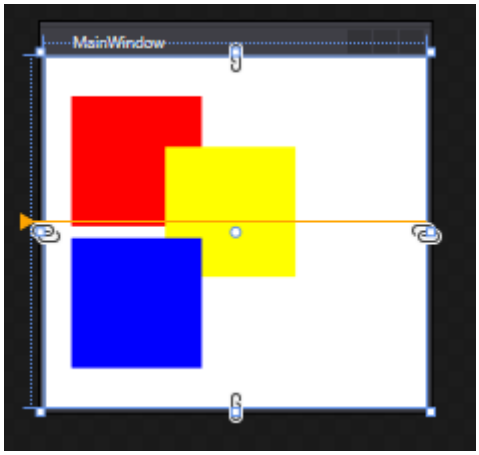
Canvas の中でコントロールが重なりあう場合は，コントロールごとに，Panel.ZIndex を指定することで，前面や背面の制御をすることができます。Panel.ZIndex の値が大きければ

大きいほど，前面に表示されます。

MainWindow.xaml

```
<Rectangle Canvas.Left="20"
           Canvas.Top="20"
           Width="100"
           Height="100"
           Fill="Red"
           Panel.ZIndex="0"/>
<Rectangle Canvas.Left="20"
           Canvas.Bottom="30"
           Width="100"
           Height="100"
           Fill="Blue"
           Panel.ZIndex="2"/>
<Rectangle Canvas.Right="100"
           Canvas.Bottom="100"
           Width="100"
           Height="100"
           Fill="Yellow"
           Panel.ZIndex="1"/>
```

この場合は「Blue」「Yellow」「Red」の順で表示されます。

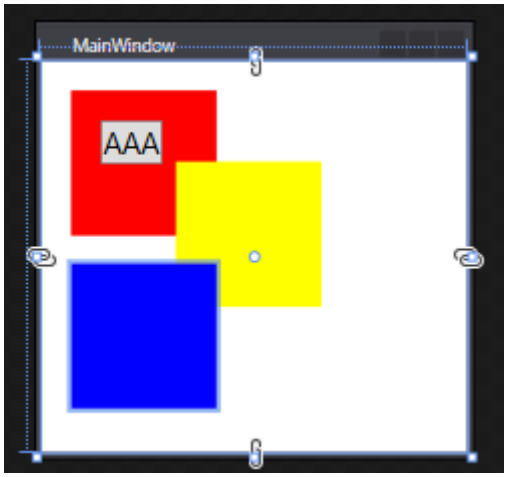


Rectangle 以外にも，普通に Button を設置することももちろんできます。

MainWindow.xaml

```
<Rectangle Canvas.Left="20"
           Canvas.Top="20"
           Width="100"
           Height="100"
           Fill="Red"
           Panel.ZIndex="0"/>
<Rectangle Canvas.Left="20"
           Canvas.Bottom="30"
           Width="100"
           Height="100"
           Fill="Blue"
           Panel.ZIndex="2"/>
<Rectangle Canvas.Right="100"
```

```
            Canvas.Bottom="100"
            Width="100"
            Height="100"
            Fill="Yellow"
            Panel.ZIndex="1"/>
<Button Canvas.Left="40" Canvas.Top="40"
        Content="AAA" FontSize="20"/>
```



サンプルコード全体

MainWindow.xaml

```
<Window x:Class="WPF030.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF030"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300">
    <Grid>
        <Canvas>
            <Rectangle Canvas.Left="20"
                       Canvas.Top="20"
                       Width="100"
                       Height="100"
                       Fill="Red"
                       Panel.ZIndex="0"/>
            <Rectangle Canvas.Left="20"
                       Canvas.Bottom="30"
                       Width="100"
                       Height="100"
                       Fill="Blue"
                       Panel.ZIndex="2"/>
            <Rectangle Canvas.Right="100"
                       Canvas.Bottom="100"
                       Width="100"
                       Height="100"
                       Fill="Yellow"
```

```
                        Panel.ZIndex="1"/>
            <Button Canvas.Left="40" Canvas.Top="40"
                    Content="AAA" FontSize="20"/>
        </Canvas>
    </Grid>
</Window>
```