Visual Studio Do-It-Yourself シリーズ

第 5 回 データ バインディング

Microsoft®

Visual Studio Do-It-Yourself
第 5 回 データ バインディング

今回は、コントロール間のプロパティを連携するバインディングや、データベースをコントロールで利用するデータ バインディングについて学びます。

ここでは、次のことを学習します。

- コントロール間のバインディング
- XML データのバインディング
- LINQ を使用したバインディング

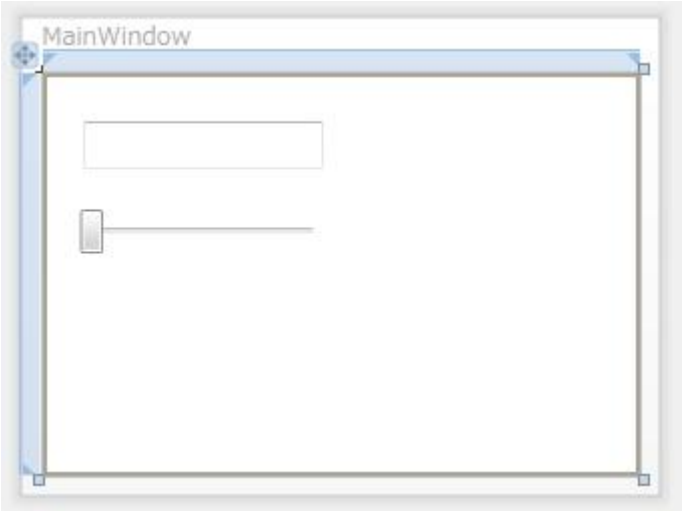
■バインディング（連結）とは

バインディング（データ バインディング）とは、コントロールとコントロール、あるいはコントロールとデータベースを結びつけて、連動させる仕組みのことです。WPF アプリケーションでは、こうしたバインディングを容易に組み込むことができ、プログラミングの必要性を少なくしています。

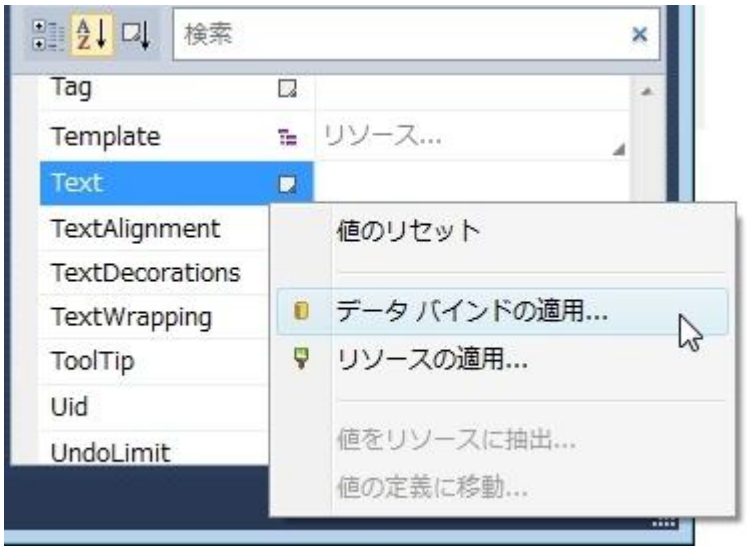
■コントロール間のバインディング

まず、コントロールのプロパティをバインディングしてみます。

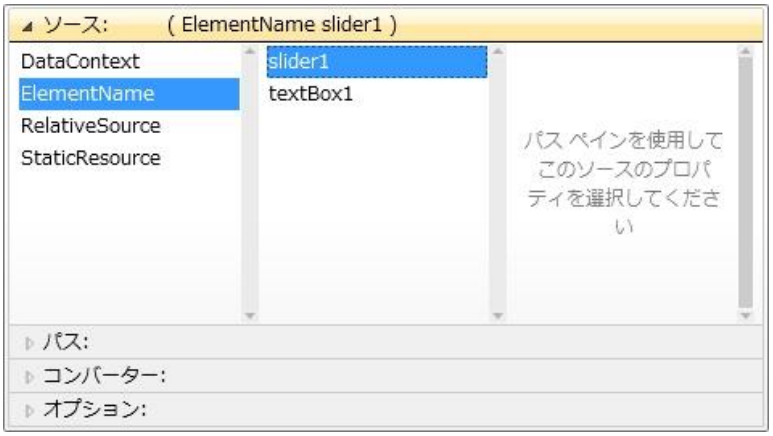
新しい WPF アプリケーションを作成して、TextBox と Slider コントロールを配置します。



TextBox を選択し、プロパティ ウィンドウで Text プロパティの脇に表示されている四角（詳細プロパティ）を右クリックして、[データ バインドの適用] を選びます。

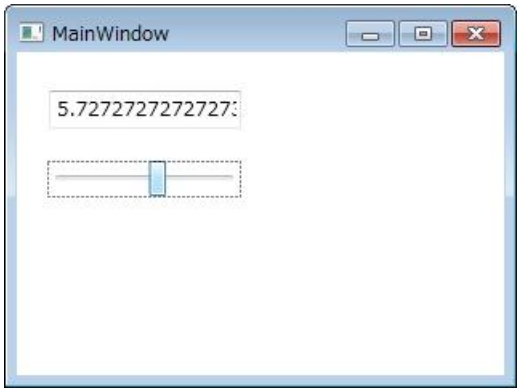


ここで、このプロパティにデータを供給する「ソース」を指定します。ここでは ElementName に Slider のコントロール名（slider1）を選びます。



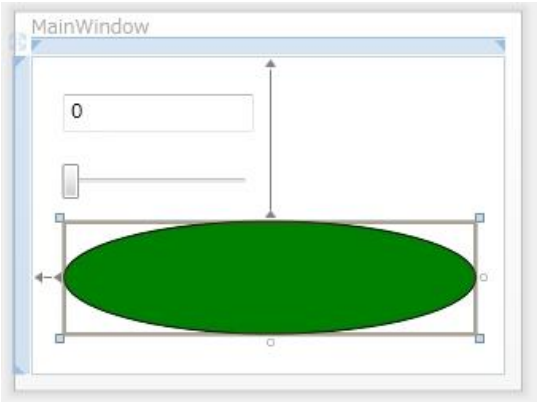
右側に「パス ペインを使用して、このソースのプロパティを選択してください」と表示されているとおり、下に表示されている「パス」をクリックして開き、プロパティの一覧から「Value」を選びます。

[Enter] を押すと設定は完了です。すでにバインディングが行われているので、ウィンドウ上の TextBox にはスライダーの現在の値（Value プロパティ）に合わせて「0」と表示されます。プログラムを実行して、スライダーを動かし、TextBox の内容が追従することを確認してください。



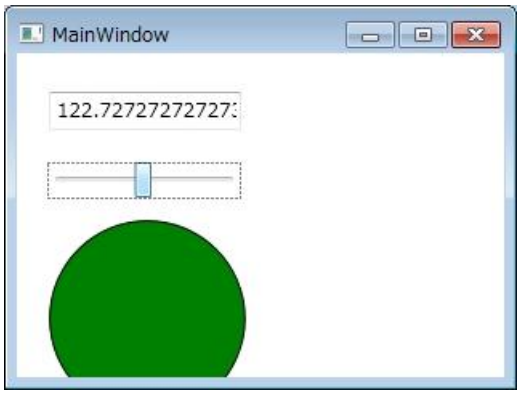
スライダーの値は 0～10（Minimum、Maximum の既定値）の範囲で変化しますが、Text プロパティは「双方向性」があるため、テキストボックスに値を入れると、スライダーのつまみの位置が変化します（テキストボックスの値を反映させるためには、[Tab] キーなどでフォーカスを外す必要があります）。また、値として変換できない場合は、赤枠でエラーが示されます。

バインディングは、さまざまな組み合わせに対応します。このウィンドウに Ellipse コントロールを配置し、プロパティ ウィンドウで Fill プロパティに適当な色を設定してください。



Slider コントロールの Maximum プロパティを少し大きな値（200 程度）に変更してください。また、Ellipse コントロールの Height と Width プロパティの両方について、さきほどと同じようにデータバインドを設定します。今回も、ElementName に slider1、パスに Value を指定します。この時点で、楕円は大きさがなくなり、見えなくなります。

このプログラムを実行すると、次のようにスライダーの移動に合わせて、テキストボックスと楕円の高さと幅がすべて追従するようになります（楕円は、高さと幅が同じなので円になります）。



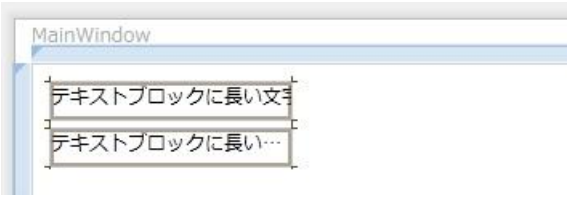
プロパティのバインディングを使えば、さまざまなものを結びつけることができます。また、数値プロパティを文字列プロパティにバインディングする場合などは、バインディング設定の「オプション」で書式を指定することもできます。Value のような整数値と TextAlignment のような列挙型の値のように、プロティどうしが型変換できないものは結び付けられません。

●コントロール自身に対するバインディング

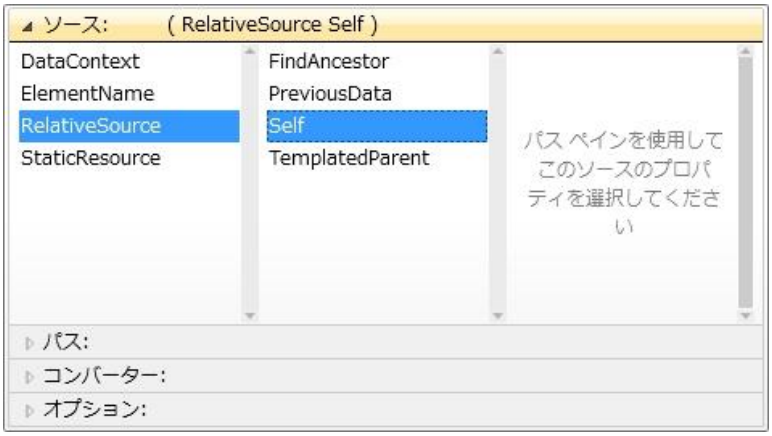
第 3 回のコントロールでは、TextBlock について説明しました。TextBlock には、指定された幅に文字列が収まらない場合に文字列の末尾に省略記号（…）を表示する TextTrimming というプロパティがあります。しかし、このプロパティを使うかどうかに関わらず、切り捨てられた文字列は表示されなくなります。

このような場合でも、ToolTip に Text プロパティと同じ内容を指定しておけば、マウスカーソルを TextBlock 上に移動させて文字列全体をツールチップ表示できます。ただし、Text と ToolTip に対して別々に文字列を設定するのは面倒ですし、一方を変更して他方を忘れてしまうおそれもあります。そこでバインディングを使い、ToolTip が常に Text プロパティと同じものになるよう設定します。

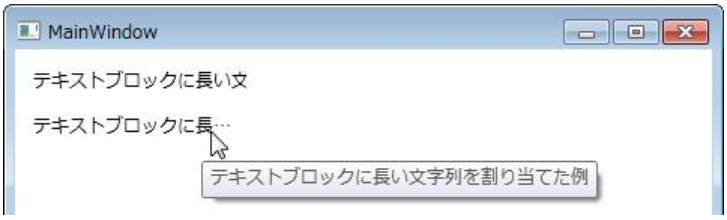
第 3 回の説明を参考に、ウィンドウに TextBlock を配置して長い文字列を割り当てます。



TextBox を選択し、プロパティ ウィンドウで ToolTip プロパティを選び [データ バインドの適用] を呼び出します。ここで、ソースとして「RelativeSource」の「Self」を設定します。これで自分自身がバインディングの対象コントロールになります。また、パスには「Text」を選びます。



プログラムを実行して、マウスカーソルを TextBlock 上に移動させると次に示すようなツールチップが表示されるようになります。



■XML データをバインディングする

バインディングできるものは、プロパティには限りません。たとえば、XML データをコントロールにバインディングすることもできます。

ここでは、MSDN オンラインの最新情報を提供する RSS を使います。この RSS 情報は、次のような形式の XML データです（一部省略）。

```
    …
<item>
    <title>Windows Azure Platform 開発をステップ バイ ステップで学ぶ…</title>
    <description>本ガイドは、Windows Azure Platform に…</description>
    <link>http://msdn.microsoft.com/ja-jp/windowsazure/…</link>
    <pubDate>Mon, 18 Oct 2010 09:00:00 GMT</pubDate>
    <guid isPermaLink="false">20771F6D-EAB9-E1E8-5E97-…</guid>
    <category domain="msdndomain:ContentType">技術情報</category>
    <enclosure url="http://www.microsoft.com/…/techarticles_rss30.jpg"
        length="1050" type="image/jpeg"/>
    <Thumbnail xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        url="http://www.microsoft.com/…/techarticles_rss30.jpg" height="30"
        width="30" xmlns="http://search.yahoo.com/mrss/"/>
</item>
    …
```

新しい WPF アプリケーションを作成し、コード ビハインド ファイル（MainWindow.xaml.cs）で次のように記述します。取り込んだ RSS データをウィンドウのデータ コンテキストとして使っています。

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

    XmlDataProvider provider = new XmlDataProvider();
    provider.Source = new Uri("http://www.microsoft.com/japan/msdn/rss.xml");
    provider.XPath = "/rss/channel/item";
    this.DataContext = provider;
  }
}
```

なお、XAML エディターで次のように記述しても同じ意味になりますが、この場合はデザイン時にも RSS データの取り込みが機能します。

```
<Window x:Class="WpfApplication5.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
  <Window.DataContext>
    <XmlDataProvider Source="http://www.microsoft.com/japan/msdn/rss.xml"
                     XPath="/rss/channel/item" />
  </Window.DataContext>
  <Grid>
  </Grid>
</Window>
```

次に、ウィンドウに ListBox を配置します。Width、Height、Margin、HorizontalAlignment、VerticalAlignment は、プロパティ ウィンドウの小さな四角（詳細プロパティ）を右クリックして、値をリセットしておきます。これで、リストボックスはウィンドウ全体に広がります。

これまでリストボックスに表示する項目は Items プロパティを使って追加していましたが、今回は ItemsSource プロパティを使います。プロパティ ウィンドウで Items プロパティの詳細プロパティで [データバインドの適用] を呼び出し、ソースとして「DataContext」、パスとして「なし」を選びます（たんに右側の「バインド中」と表示されている部分をクリックするだけでも同じです）。ウィンドウに割り当てたデータ コンテキストは、そこに配置されるコントロールにも伝搬するため、データ コンテキストをそのまま項目として使うという意味になります。

このプログラムを実行すると、次のように表示されます。

ここでは、RSS が提供するすべての項目がまとまって表示されてしまうため、わかりにくいものになっています。リストボックスには、項目のデータ構造にしたがって、どのように表示するかを決めるテンプレートという仕組みがあります。テンプレートの詳細は第 8 回で説明しますが、XAML を使って記述する必要があるため、XAML エディターを開いて次のように記述してください。

```
<ListBox Name="listBox1" ItemsSource="{Binding}">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel>
        <TextBlock FontSize="10" Text="{Binding XPath=category}" />
        <TextBlock FontSize="14" Text="{Binding XPath=title}" />
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

これは、個々のリストボックスの項目について、StackPanel レイアウトに 2 つの TextBlock を配置し、それぞれに XML 項目のパス（XPath）として category と title をバインドしています。これにより、それぞれの項目が指定したレイアウトで表示されることになります。

このプログラムを実行した様子を示します。

■LINQ を使用したデータ バインディング

最後に、LINQ（Language INtegrated Query）と呼ばれる機能を使ったバインディングについて説明します。LINQ は、言語の中のオブジェクトを対象にする場合も、データベースを対象にする場合も、同じように記述することができるもので、Visual Studio 2008 以降の C#や Visual Basic で採用されています。今回は、プログラム内のデータを対象にする LINQ to Objects を使いますが、データベースを対象にする LINQ to SQL などでもバインディングの基本は同じです。前述の XML データも LINQ を通じて扱うことができます。LINQ の詳細については、「LINQ」を参照してください。

新しい WPF アプリケーションを作成して、ウィンドウに ListBox コントロールを配置します。前述の手順で、リストボックスを最大化しておきます。コード ビハインド ファイル（MainWindow.xaml.cs）に切り替え、次のように記述します。

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

    var colornames = from color in typeof(Colors).GetProperties()
        select new { Name = color.Name, Color =  color.GetValue(null, null) };
    listbox1.ItemsSource = colornames;
  }
}
```

typeof(Colors).GetProperties()という文は、定義済みの色を定義している Colors クラスから、すべての色情報を列挙させて取り出すものです。こうして得られた結果から、色の名前（Name）と色情報そのもの（Color）をひとつの要素として、すべて列挙したものを colornames に代入しています。これを直接項目データとして割り当てています。

続いてユーザー インファーフェイスの定義です。今回はプログラムで ItemsSource を設定しているため、ListBox コントロール自身に対してバインディングを設定する必要はありません。リストボックスに表示するそれぞれの項目のためにレイアウトとバインディングの設定が必要です。このためには、XAML を記述する必要があります。

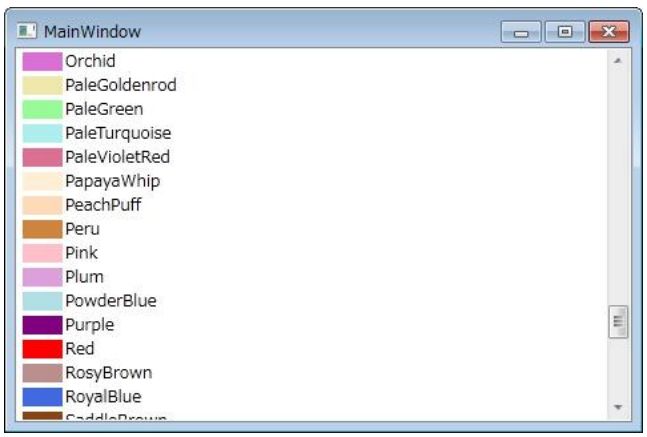
XAML エディターに切り替えて次のように記述します。

```
<ListBox Name="listbox1" >
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <Rectangle Width="30" Margin="2" >
          <Rectangle.Fill>
            <SolidColorBrush Color="{Binding Path=Color}" />
          </Rectangle.Fill>
        </Rectangle>
        <TextBlock Text="{Binding Path=Name}" />
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

ここでは、各項目のためのテンプレートを定義しています。StackPanel レイアウトを使い、最初の要素として Rectangle を、2 番目の要素として TextBlock を横に並べています。Rectangle の Fill プロパティには、SolidColorBrush を割り当て、Color プロパティには Path を使ってリストボックスの項目に割り当てられているデータの Color 項目をバインディングしています。ここで「Color="{Binding Path=Color}"」としないことに注意してください。Fill プロパティは Color 型ではなく Brush 型なので、ブラシオブジェクトを割り当てる必要があります。

※注意　実際には、「Fill="{Binding Path=Name}"」と色名（文字列型）を割り当てることで自動的にブラシに変換されます。

プログラムを実行すると、次のようにリストボックスの各項目には、色で塗られた長方形と、色名が表示されます。



■まとめ

今回は、バインディングについて学びました。バインディングを活用することで、ユーザー インターフェイス処理のためにプログラムを書く手間を大幅に減らすことができます。

次回は、リソースについて学びます。