Visual Studio Do-It-Yourself シリーズ

第 3 回 コントロール

Microsoft®

Visual Studio Do-It-Yourself
第 3 回 コントロール

今回は、Windows アプリケーションのユーザー インターフェイスを作成する際に使う基本的なコントロールについて学びます。

ここでは、次のことを学習します。

- 主要なコントロールとその機能
- XAML を使ったコントロールの配置
- レイアウトとコントロールの関係

今回は、コントロールの学習を通じて、次のようなアプリケーションを作成します。

■コントロールの概要

第 2 回で学んだレイアウト パネルを使うことで、ユーザー インターフェイスの全体的な構成を決めることはできますが、レイアウトは配置場所を決めるだけです。アプリケーションが、実際にどのような入力や表示をするかはその他のコントロールを使う必要があります。

今回は、Visual Studio 2010 で Windows アプリケーション（WPF アプリケーション）を作成する場合に利用できる標準的なコントロールについて解説します。Windows アプリケーションを開発するためのコントロールについて、次の表に示します。なお、この表に掲載したコントロール以外にも、さまざまなコントロールがあります。一部のコントロールや機能は、Visual Studio の WPF デザイナーでは使うことができません。これらは XAML エディターで編集する必要がありますが、ここでは特に触れません。

コントロール	機能
Border	別の要素の周囲に境界線、背景、またはその両方を描画します
Button	クリックするとイベントを発生します
Calendar	月単位のカレンダーを表示します
CheckBox	オン/オフ、Yes/No のような切り替えを選択するチェック ボックス
ComboBox	入力ボックスとドロップダウンリストボックスを組み合わせたもの
DataGrid	データを表形式で表示します
DatePicker	日付を入力または選択するためのコントロール
DocumentViewer	XPS ドキュメントなど固定ドキュメントを表示します
Ellipse	楕円を表示します
Expander	開閉可能な領域を作ります
GridSplitter	Grid レイアウトに配置して、分割位置を移動するために使います
GroupBox	ヘッダーと境界線を表示してコントロールをグループ分け表示します
Image	画像を表示します
Label	文字列を表示し、アンダースコア（_）を使ってアクセスキーを指定できます
ListBox	複数の項目を表示したり、選択したりします
ListView	複数の項目をアイコン形式で並べて表示したり、選択したりします
MediaElement	オーディオやビデオを再生します
Menu	メニューを表示して、メニュー項目を選んでイベントを発生します
PasswordBox	マスク付きでパスワードを入力するためのコントロールです
ProgressBar	進行状況を表示するためのコントロール
RadioButton	複数の選択肢から一つを選ぶためのコントロール
Rectangle	長方形を表示します
RichTextBox	書式付きのテキストを表示したり、入力したりするコントロール
ScrollBar	スクロールバーを表示し、つまみを使って位置を選択できるようにします
ScrollViewer	スクロールバーを使ってコントロールより広い範囲を使えるようにします
Separator	主にメニューやリストボックスの項目として使い、区切り線を表示します
Slider	指定された範囲内の値をつまみを使って変更します
StatusBar	通常、ウィンドウ下部に情報を表示するために使います
TabControl	タブを使って表示内容を切り替えるためのコントロール
TextBlock	テキストを表示します
TextBox	文字列を入力します
ToolBar	アイコン付ボタンなどを配置するツールバー
ToolBarTray	ToolBar を配置するためのコントロール
TreeView	ツリー形式で項目を表示します
Viewbox	この上に配置したコントロールを拡大表示します
WebBrowser	ブラウザーのように HTML ドキュメントを表示します
WindowsFormsHost	Windows フォーム用のコントロールを使えるようにします

■主要なコントロール

よく使われるコントロールをピックアップして解説します。ここで解説するのは、それぞれの主要な機能のみです。機能の詳細は、オンライン ヘルプなどを参照してください。

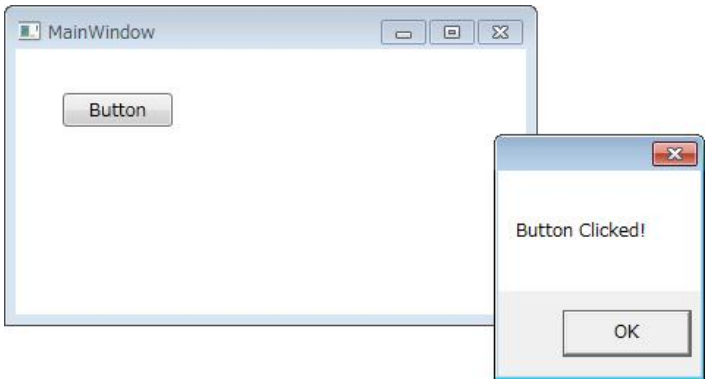
●Button

家電製品の機能ボタンや、玄関の呼び出しボタンのように、「押して何かの機能を呼び出す」ためのコントロールです。

ボタンに表示する内容は Content プロパティで指定します。ボタンをダブルクリックすると、ボタンを押したときの動作をプログラムで記述できます。新しいウィンドウに Button コントロールを配置して、ダブルクリックし、次のようなイベントハンドラーを割り当てることができます。

```
private void button1_Click(object sender, RoutedEventArgs e)
{
  MessageBox.Show("Button Clicked!");
}
```

これを実行してボタンを押すと、「Button Clicked!」というメッセージ ダイアログが表示されます。



ボタンは、マウスでクリックするだけでなく、入力フォーカス（文字入力を受け付ける状態）があるときにスペースや [Enter] キーを使って押すこともできます。また、ボタン以外のコントロールに入力フォーカスがあるときも、ボタンの IsDefault や IsCancel プロパティをチェックしておくと、それぞれ [Enter] や [Esc] を押したときに、対応するボタンが押されたことになります。

イベント ハンドラーは、プロパティ ウィンドウでも割り当てられます。プロパティ ウィンドウには、[プロパティ] というタブの隣に [イベント] というタブがあり、これをクリックすれば Button コントロールで使えるイベントが列挙されます。Click イベントの右側をダブルクリックすると、さきほどと同じようにイベント ハンドラーを記述できます。

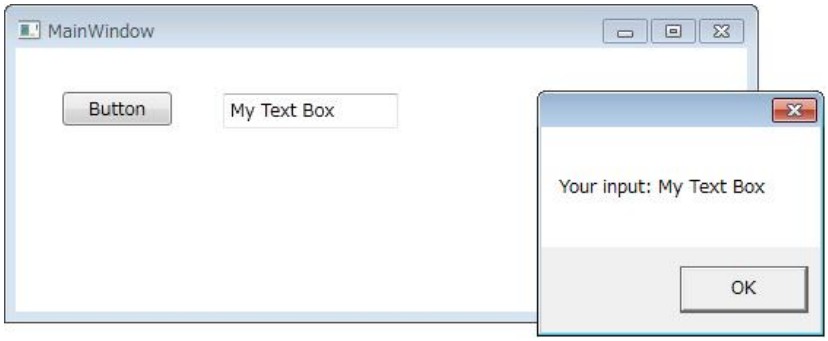
●TextBox

文字列を入力するためのコントロールです。

あらかじめ文字列を表示する場合は、プロパティ ウィンドウの Text プロパティを使って設定しておきます。入力または編集した文字列は、Text プロパティを参照すれば取り出せます。前述のアプリケーションに TextBox コントロールを追加して、ボタンのイベント ハンドラーを次のように修正します。

```
private void button1_Click(object sender, RoutedEventArgs e)
{
  MessageBox.Show("Your input: " + textBox1.Text);
}
```

これを実行して、TextBox コントロールに文字列を入力し、ボタンを押すと、入力した文字列を含むメッセージボックスが表示されます。

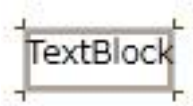


TextBox は、1 行だけでなく複数行のテキストを編集するためにも使えます。このとき、AcceptsReturn や AcceptsTab をチェックしておくと、編集中に [Enter] や [Tab] キーを受け入れることができます。TextWrapping プロパティを Wrap にすると、領域の右端を超えるときに折り返して表示されます。

複数行に渡って入力された行は、Lines プロパティ（string[]型）を使うと 1 行ごとに読み取ることができます。

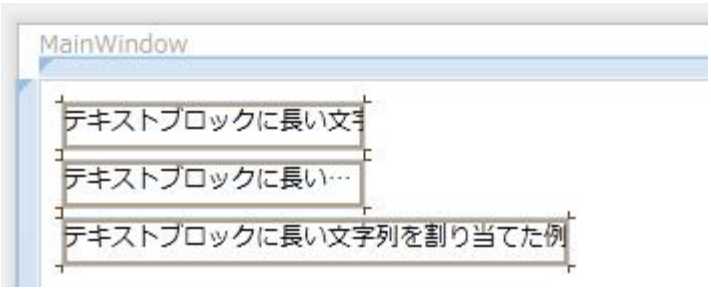
●TextBlock

文字列を表示するためのコントロールです（下図は選択状態のもので、通常、枠はありません）。



文字列を表示するためのコントロールには、後述する Label もありますが、TextBlock の方が軽量で基本的なコントロールです。TextBlock に表示する文字列は Text プロパティに割り当てることができ、左右方向の位置合わせには TextAlignment プロパティを使います。また、TextWrapping を Wrap に指定すれば、指定された幅を超えた文字列を折り返して表示することもできます（Label には、この機能はありません）。

TextBlock の特徴的な機能として、TextTrimming プロパティがあります。これは指定された幅に収まりきらないテキストを「…」で省略表示するというものです。次の図は、3 つの TextBlock に同じ文字列を Text プロパティに割り当て、その他の条件を変えて配置したものです。



最初の 2 つは Width プロパティを固定値（ここでは 150）に設定し、2 つめは TextTrimming プロパティを CharacterEllipsis に設定しています。3 つ目は Width プロパティを Auto にしています。幅の決まっている場所に文字列を表示したい場合は、TextTrimming を使うと文字列が表示しきれていないことが一目でわかります。文字列全体を確認させるためには ToolTip プロパティに Text と同じ文字列を割り当てておくとよいでしょう。これは、第 5 回で説明するデータ バインディングを使うと便利です。

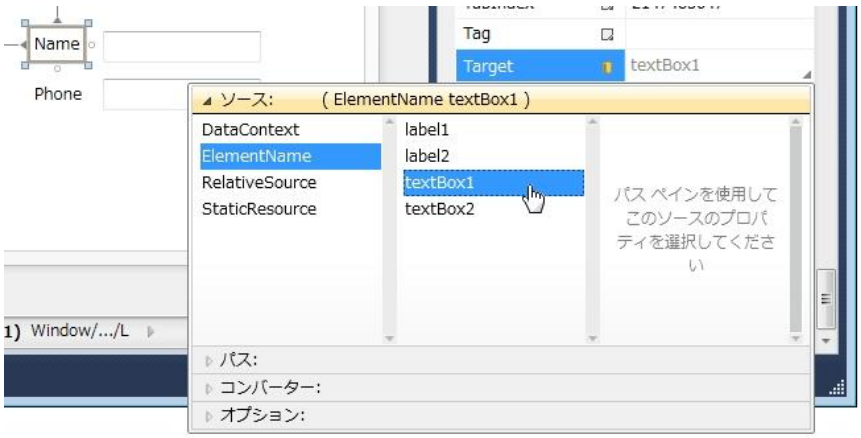
●Label

文字列を表示するためのコントロールです（下図は選択状態のもので、通常、枠はありません）。

表示する文字列は Content プロパティに設定します。前述の TextBlock は文字列を表示するだけのコントロールですが、Label では枠の色や太さ（BorderBrush、BorderThickness）、枠内での水平と垂直方向での表示位置（HorizontalContentAlignment と VerticalContentAlignment）などを設定できます。あまり使われませんが Focusable プロパティをチェックしておけば、入力フォーカスを持つこともできます。

また、Content プロパティにおいて英字の前にアンダースコア（_）を置くことで、アクセスキー（[Alt]+[英字]）を指定できます。通常、Label 自身に入力フォーカスを持たせることはなく、アクセスキーが押されたときの入力フォーカスは Target プロパティで指定します。Target プロパティの割り当てには、第 5 回で説明するデータ バインディングという仕組みを使いますが、ここでは手順のみを説明します。

まず、Label と対象となるコントロールを配置します。Label コントロールの Content プロパティに指定する文字列は、どこかにアンダースコア（_）と英字の組み合わせを入れておきます。たとえば、「_File」とすれば「File」というラベルに、「ファイル(_F)」とすれば「ファイル(F)」というラベルになります。この Label コントロールのプロパティ ウィンドウで Target プロパティを探します。ここで右側の「バインド中...」という部分をクリックすると、バインディングを指定するドロップダウンが表示されます。「ソース」エリアで、ElementName と対象となるコントロール名を選びます。



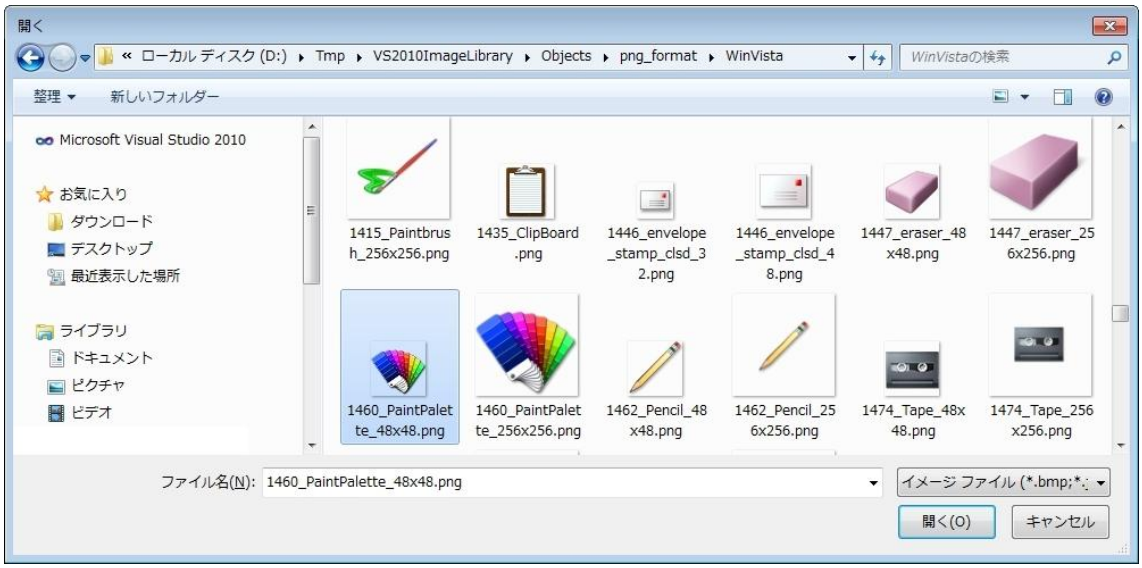
アプリケーションを実行すると、最初は下線が表示されていませんが、[Alt] を押すと表示されます。ここで、[Alt]+[F] のように入力すれば、対象として指定したコントロールに入力フォーカスが移ります。

●Image

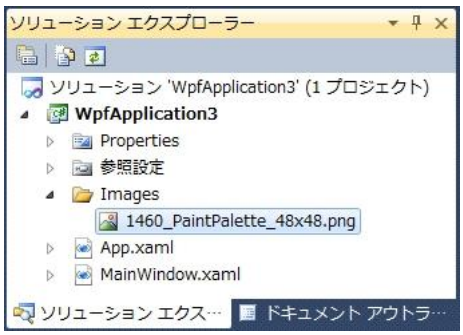
イメージ（画像）を表示するためのコントロールです。

表示するイメージは、Source プロパティで指定します。イメージ データは、プロジェクトに取り込まれたものでも、外部のイメージ ファイルでもかまいませんが、ここでは既存のイメージファイルをプロジェクトに取り込んで表示する例を紹介します。

ツールボックスで Image を選んでウィンドウに配置します。プロパティ ウィンドウで、Source プロパティの右側の [...] ボタンをクリックし、「イメージの選択」ダイアログを表示します。



画像を選んで [開く] ボタンを押すと、その画像がリソースとしてプロジェクトに取り込まれます。



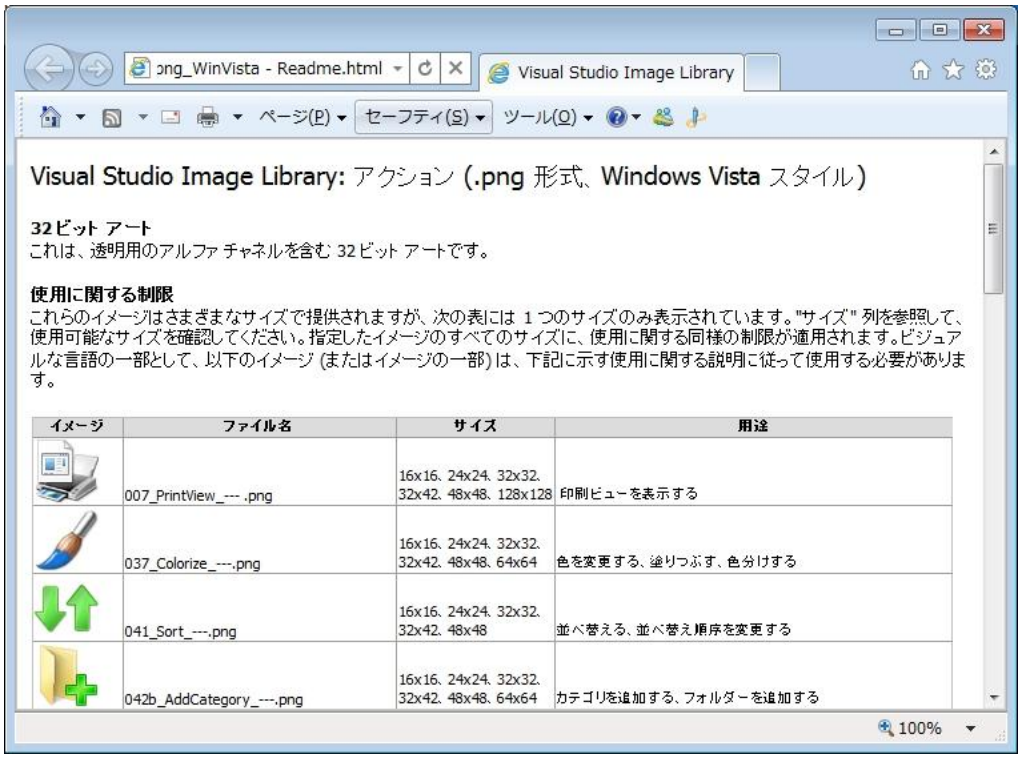
取り込むイメージはプロジェクトの直下ではなく、Images フォルダーに格納されます。Image コントロールを使わない場合でも、あらかじめ Images フォルダーを作って取り込むようにしてください。

取り込んだイメージは、配置されている Image コントロールの大きさに合わせて拡大・縮小されます。これは、Stretch プロパティが Fill になっているためです。Stretch に指定する値は次の中から選べます。

Stretch プロパティ	意味
None	画像を拡大・縮小しない
Fill	Image コントロールの大きさに合わせて拡大・縮小する
Uniform	縦横の比率を変えずに、画像がコントロールに収まるよう拡大・縮小する
UniformToFill	縦横の比率を変えずに、画像の縦または横が収まるように拡大・縮小する

Image コントロール自身の大きさを表示する画像に合わせる場合は、Width と Height プロパティを削除します。これは Auto を設定するのと同じです。

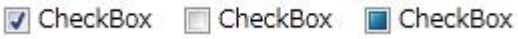
Image コントロールは、BMP、GIF、ICO、JPG、PNG、WDP（HDP）、TIFF 形式のファイルを扱えます。また、Visual Studio 2010 には、アプリケーションでよく使われる汎用的な画像をイメージ ライブラリとして提供しています（これは、Express Edition には含まれていません）。イメージ ライブラリは、¥Program Files¥Microsoft Visual Studio 10.0¥Common7¥VS2010ImageLibrary¥1041 フォルダーに VS2010ImageLibrary.zip という圧縮ファイルとして用意されているので、必要な場合には適当なフォルダーに展開して使います。Visual Studio や Office で使われているイメージも含まれており、それぞれのフォルダーには解説用の html ファイルが置かれています（下図参照）。



●CheckBox

オンとオフ、Yes と No、True と False のように、2 つの値を切り替えるためのコントロールです。

チェックボックスに表示する内容は Content プロパティに指定します。チェックされているかどうかは、IsChecked プロパティで判断しますが、これは True と False 以外に null も設定できます。これは、たとえば「はい」「いいえ」「該当しない」、「Yes」「No」「無効」といった 3 値の切り替えが必要な場合に使います。マウスクリックで 3 値を切り替えるためには、IsThreeState プロパティをチェックしておきます。

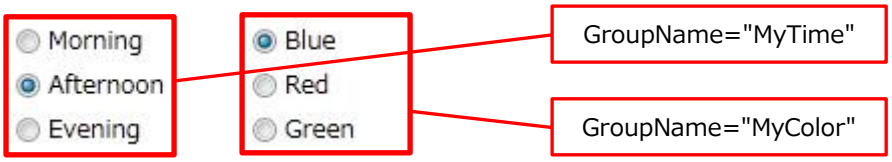


イベント ハンドラーでは、IsChecked プロパティは bool 型ではなく bool?型（Visual Basic では Boolean?型）になることに注意してください。

●RadioButton

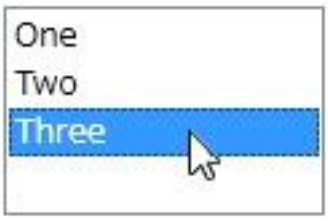
複数の選択肢の中から、1 つを選ぶときに使うコントロールです。

ラジオボタンもチェックボックスと同じように表示する内容を Content プロパティで指定します。ラジオボタンでは、マウスでクリックしたものが有効（チェックされた状態）になり、それ以外のラジオボタンは無効（チェックされていない状態）になります。ラジオボタンには GroupName というプロパティがあり、同じカテゴリーのラジオボタンに同じ名前を指定することで、1 つのウィンドウの中で目的別のラジオボタンを使い分けることができます。GroupName を指定されていないラジオボタンは、ウィンドウの中で共通のものとみなされます。

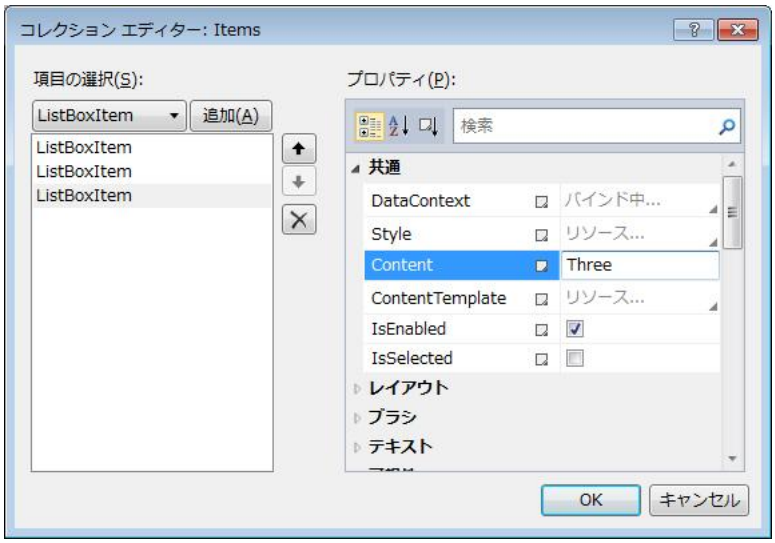


●ListBox

長方形の領域の中に表示されている複数の項目の中から、1 つの項目を選ぶためのコントロールです。



リストボックスに追加する項目は Items プロパティで設定します。プロパティ ウィンドウで Items を探し、右側の [...] ボタンをクリックすると、下図のようなコレクション エディターが表示されます。

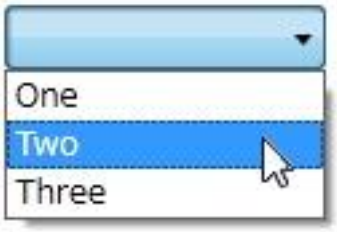


ここで [追加] ボタンをクリックして、項目（ListBoxItem）を追加していきます。右側は項目ごとのプロパティを編集する場所となっており、「共通」カテゴリーにある Content プロパティで表示する文字列を指定できます。たとえば、3 つの項目を追加して、それぞれの Content プロパティと Foreground プロパティをともに「Blue」「Red」「Green」とすると、以下の図のようなリストボックスになります。

項目リスト（Items）は、プログラムで追加したり、削除したりできます。また、リストボックスのどの項目が選択されているかということは SelectedIndex（位置、未選択の時は-1）や SelectedItem などで参照できます。SelectionMode を Multiple や Extended にすると複数の項目を選択できるようになります。

●ComboBox

長方形の領域の中に表示されている複数の項目の中から、1 つの項目を選ぶためのコントロールです。



コンボボックスは、前述のリストボックスと同じように Items プロパティを使って項目を設定しておきますが、この項目はウィンドウ上には表示されていません。コントロールの右側にある下向きの▼をクリックしたときにドロップダウン形式で項目リストが表示され、そこから選ぶことができます。

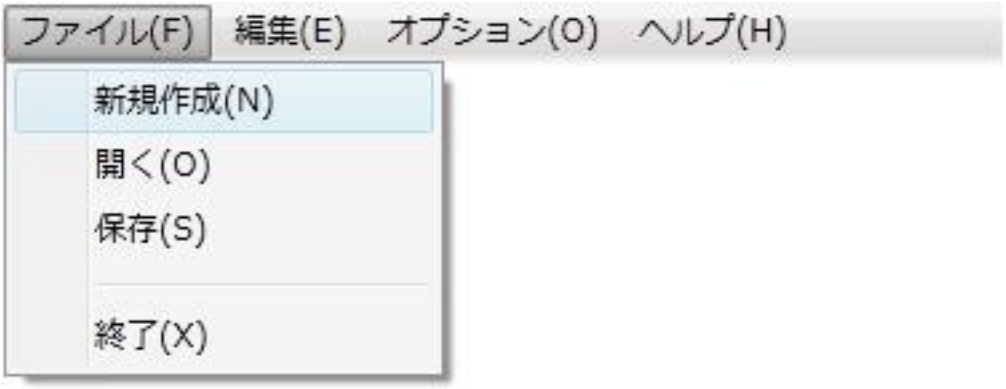
コンボボックスは、選択肢の中から 1 つの項目を選ぶという点で、ラジオボタンに似ています。ラジオボタンに比べて、プログラムで項目リストの変更することが容易にできること、ラジオボタンよりも狭い場所しか使わないこと、項目を選ぶために、いったんクリックしてドロップダウンする必要があるといった違いがあります。このため、性別（男・女）や元号（明治・大正・昭和・平成）のように選択肢が少ない場合にはラジオボタンを、生まれた月（1～12）や日（1～31）、都道府県（47 個）のように選択肢が多い場合には ComboBox を使うとよいでしょう。

項目を追加する方法はリストボックスと同じですが、コンボボックスは文字列入力を受け付ける機能もあります。IsEditable プロパティをチェックすると、ボタンのように表示されていた部分が入力ボックスのようになり、実行時にはドロップダウン リストから選ぶだけでなく、新たな文字列を入力できます。次の図は、前述のリストボックスと同じ項目を追加して、IsEditable をチェックして実行した様子です。

IsEditable をチェックして、さらに IsReadOnly プロパティをチェックすると、入力ボックスは読み取り専用になり、編集できなくなります。この場合でも、ドロップダウン リストから選択した項目について、範囲選択してクリップボードにコピーすることができます。

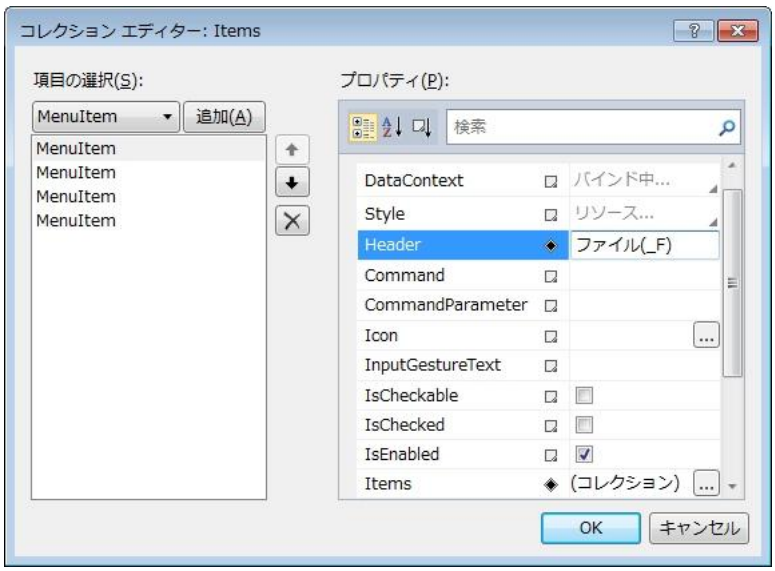
●Menu

ウィンドウにメニュー バーを追加するためのコントロールです。



メニューバーは通常、ウィンドウのタイトルバーのすぐ下に配置します。これには第 2 回で学んだレイアウト パネルを使うのが便利です。たとえば、Grid で行分割し、最初の行に配置することができます。あるいは DockPanel に配置して、DockPanel.Dock="Top"属性を指定することもできます。

メニュー項目は、Items プロパティで設定します。プロパティ ウィンドウで Items を探し、右側の [...] ボタンをクリックすると、次のようなコレクション エディターが表示されます。



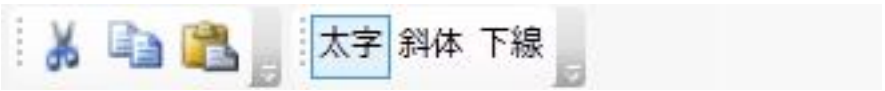
ここで、MenuItem を追加すると、トップレベルのメニュー項目が追加されます。それぞれの項目に表示する文字列は右側の Header プロパティで指定します。各メニュー項目に対応するプルダウンは、それぞれの MenuItem 項目に対する Items プロパティで指定します。このときは [項目の選択] で Separator を選ぶことで「区切り線」を追加できます。さらに Items プロパティを使って、サブメニューを追加できます。具体的なメニューの追加手順については後述します。

■XAML を使ってコントロールを配置する

コントロールによっては、WPF デザイナーだけでは使いにくいものがありますが、この場合は XAML という記述言語を使います。ここでは、このようなコントロールの例として ToolBar と StatusBar を取り上げます。

●ToolBar

ウィンドウにツール バーを追加するためのコントロールです。



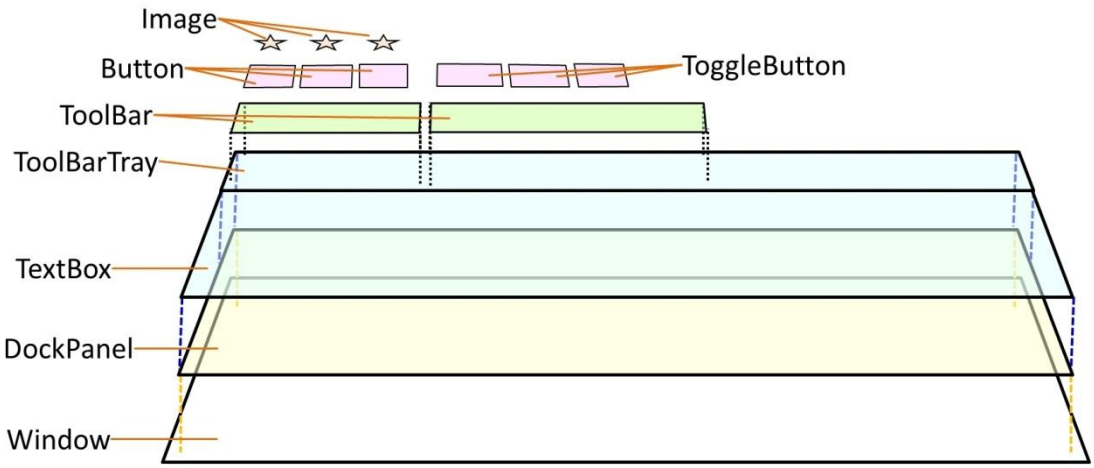
ツールバーは、主にアイコン（イメージ）を使った比較的小さなボタンを並べたもので、メニューよりもすばやく機能を呼び出せます。ツールバーは、ToolBar コントロール上にボタンやコンボボックスなどのコントロールを配置することで実装しますが、この操作は WPF デザイナーではサポートされていません。

ToolBar コントロールは単独でも使えますが、通常は複数の ToolBar を ToolBarTray コントロール上に配置します。ツールバーに配置するボタンには前述の Button コントロールを使えます。ToggleButton（切り替え式のボタン）というチェックボックスのように機能するボタンもよく使われます。Button コントロールには、文字列だけでなくイメージも表示できますが、これも XAML を使って記述する必要があります。

新しい WPF アプリケーションを作成して、（WPF デザイナーではなく）XAML エディターを開きます。ここで、<Grid>～</Grid>を以下の<DockPanel>～</DockPanel>で置き換えます。

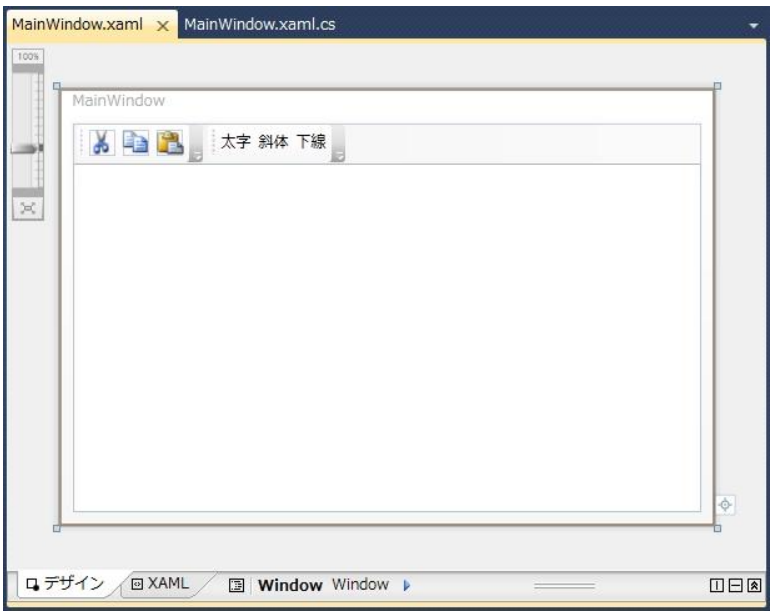
```
<Window…
  <DockPanel>
    <ToolBarTray DockPanel.Dock="Top">
      <ToolBar>
        <Button Name="btnCut"><Image Source="images/CutHS.png" /></Button>
        <Button Name="btnCopy"><Image Source="images/CopyHS.png" /></Button>
        <Button Name="btnPaste"><Image Source="images/PasteHS.png" /></Button>
      </ToolBar>
      <ToolBar>
        <ToggleButton Name="btnBold">太字</ToggleButton>
        <ToggleButton Name="btnItalic">斜体</ToggleButton>
        <ToggleButton Name="btnUnderline">下線</ToggleButton>
      </ToolBar>
    </ToolBarTray>
    <TextBox DockPanel.Dock="Top" TextWrapping="Wrap">
    </TextBox>
  </DockPanel>
</Window>
```

XAML（eXtensible Application Markup Language）は、XML（eXtensible Markup Language）を元にした記述言語で、<タグ名>～</タグ名> でユーザー インターフェイスの要素を定義します。これまで WPF デザイナーで編集していたものも、その背後では XAML コードに変換されていました。この例では、DockPanel の中に ToolBarTray と Grid が配置され、さらに ToolBarTray の中に 2 つの ToolBar が配置され、最初の ToolBar には 3 つの Button が配置され、それぞれに Image コントロールが含まれています。2 番目の ToolBar には 3 つの ToggleButton が配置されています。図で示すと次のようになります。



この例では、前述の Image コントロールで説明したイメージ ライブラリを使っています。具体的には、VS2010ImageLibrary.zip の "Actions¥png_format¥Office and VS" フォルダーにある CutHS.png、CopyHS.png、PasteHS.png を Images フォルダーに取り込んでいます（イメージ ライブラリは、Express Edition には含まれていません）。

WPF デザイナー（[デザイン] タブ）に切り替えると、次のように表示されます。



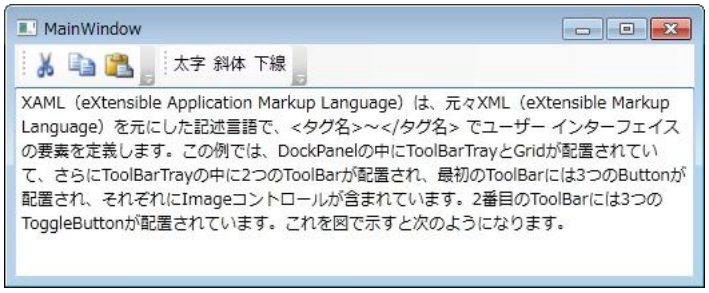
XAML を使って配置したコントロールも、WPF デザイナーに切り替えれば、コントロールをダブルクリックしてイベント ハンドラーを定義したり、プロパティ ウィンドウでプロパティを設定したりできます。

イメージを載せた Button コントロールを選ぶときには注意してください。そのままクリックすると、Button ではなく Image コントロールを選んでしまうかもしれません。どのコントロールを選んでいるかは、プロパティ ウィンドウのタイトルの下で判断できます。「Image <名前なし>」のように表示されていれば、Image コントロールが、「Button btnCut」のように表示されていれば Button コントロールが選ばれています。Image を選んでいるときでも、[Esc] キーを押せば、それを配置しているコントロール、つまり Button コントロールに選択が移動します。同じように、[Esc] を押すたびに ToolBar、ToolBarTray、DockPanel、Window と下の階層にあるコントロールに選択が移動していきます。

コントロールを確実に選択するためには、ドキュメント アウトラインを使うこともできます。ドキュメントアウトラインは、下図のようにコントロールの配置されている様子が階層化されて表示されており、ここで目的のコントロールを選ぶと、プロパティ ウィンドウに、そのコントロールの情報が表示されます。

ここで、btnCut（ハサミのイメージを置いてあるボタン）を選んでください。プロパティ ウィンドウで、このボタンの Command プロパティを探し、右側の下向き▼を押すと、ここで使えるコマンドの一覧がドロップダウン リストとして表示されます。ここでは「Cut」を選びます。同じように、btnCopy の Command プロパティには「Copy」を、btnPaste の Command プロパティには「Paste」を選んでください（コマンドについては第 4 回で説明します）。

このプロジェクトを実行して、文字列を入力してください。左側の 3 つのボタンによって、「切り取り」「コピー」「貼り付け」といった操作が機能することがわかります（右側の 3 つのボタンは機能しません）。



●StatusBar

ウィンドウにステータス バーを追加するためのコントロールです。

StatusBar コントロールは、プロパティ ウィンドウで Items プロパティの右側にある [...] ボタンを押すことで、StatusBarItem という項目表示用のコントロールを追加できます。また、XAML を使うことで ToolBar と同じく StatusBar にもさまざまなコントロールを配置できます。

前述のツールバーで作成したアプリケーションで、<ToolBarTray>の記述の後、<TextBox>の記述の前に、次のように<StatusBar>の記述を追加します。

```
    ……
    </ToolBarTray>
    <StatusBar DockPanel.Dock="Bottom">
      <StatusBarItem Name="statusDate" Content="(date)" />
      <Separator />
      <ComboBox Name="comboZoom" Width="70">
        <ComboBoxItem Content="75%" />
        <ComboBoxItem Content="100%" />
        <ComboBoxItem Content="150%" />
      </ComboBox>
      <Separator />
      <StatusBarItem Name="statusMessage" Content="message" />
    </StatusBar>
    <TextBox DockPanel.Dock="Top" TextWrapping="Wrap"></TextBox>
  </DockPanel>
</Window>
```

これにより、上図のようなステータスバーを追加できます。いったんウィンドウに追加すると、WPF デザイナーで配置したコントロールのプロパティを操作できます。ツールバーやステータスバーに配置したコントロールも、レイアウト パネルに配置する場合と同じようにプロパティを設定したり、イベント ハンドラーを割り当てたりできます。

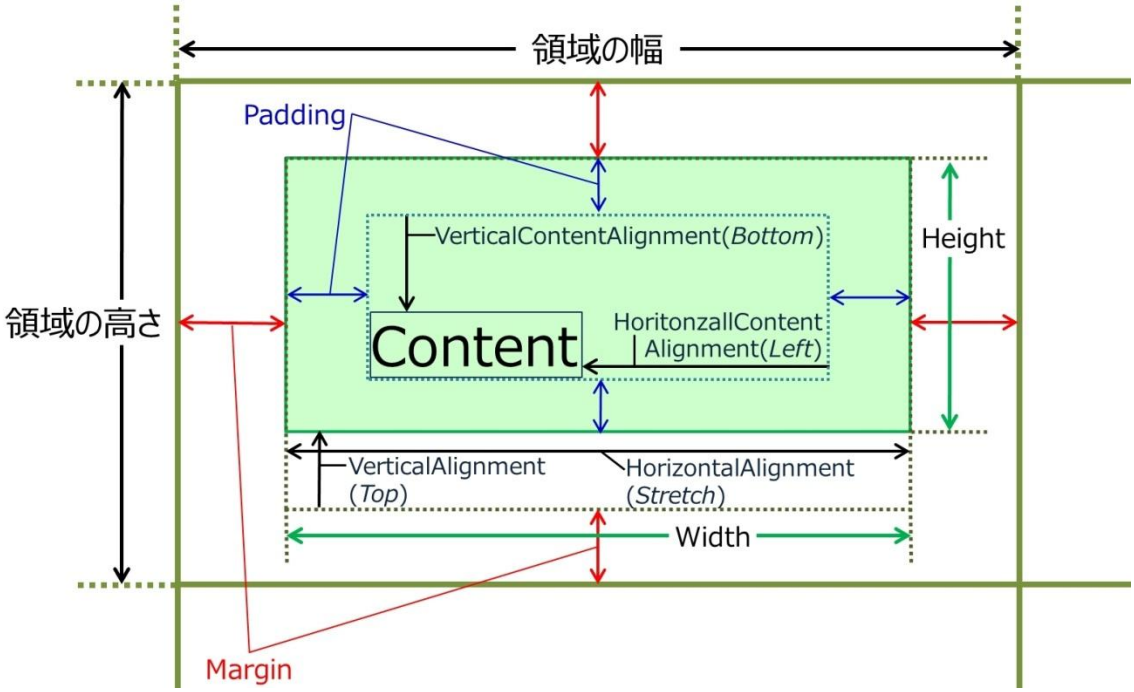
■レイアウトとコントロール

第 2 回で学んだレイアウトとコントロールの配置には、さまざまな設定があります。多くのコントロールは、配置される位置や大きさを調整するために以下のプロパティを持っています。

プロパティ	意味
Width	コントロールの幅。数値または Auto。既定値は Auto。
Height	コントロールの高さ。数値または Auto。既定値は Auto。
Margin	領域とコントロールの間隔。単独の数値で一括指定するか、左、上、右、下の順に個別設定。既定値は、すべて 0。
Padding	コントロールと表示コンテンツの間隔。単独の数値で一括指定するか、左、上、右、下を個別設定。既定値は、すべて 0。
HorizontalAlignment	領域中でコントロールを左右どちらに調整するか。Left、Center、Right、Stretch のいずれか。既定値は Stretch。
VerticalAlignment	領域中でコントロールを上下どちらに調整するか。Top、Center、Bottom、Stretch のいずれか。既定値は Stretch。
HorizontalContentAlignment	コントロール中でコンテンツを左右どちらに調整するか。Left、Center、Right、Stretch のいずれか。既定値は Left。
VerticalContentAlignment	コントロール中でコンテンツを上下どちらに調整するか。Top、Center、Bottom、Stretch のいずれか。既定値は Top。

※既定値は標準的なコントロールの場合です。

領域とコントロール、およびこれらのプロパティの関係を以下の図に示します。



この図で薄い緑色で塗られている部分が領域の中に配置するコントロール、さらに「Content」で示されている部分がコントロール上に表示する内容です。Button や CheckBox では、表示する内容を Content プロパティで設定します。前述の ToolBar の例で示した通り、XAML で記述することで、Content には文字列だけでなく、他のコントロールを配置することもできます（レイアウト パネルを含みます）。

これらのプロパティが影響を受けやすいのは Grid レイアウトです。Grid では、列の幅や行の高さのそれぞれについて固定・可変（加重平均）・自動調整という設定があるためです。Canvas レイアウトでは、領域の大きさが存在しないため、HorizontalAlignment や VerticalAlignment が意味を持ちません。StackPanel や WrapPanel では並べる方向（縦または横、WrapPanel は両方）の高さや幅は必要な最小値が使われます。

通常、これらの設定が問題になることはありませんが、ウィンドウのサイズ変更に合わせて領域のサイズも自動的に調整されるように設定している場合には注意が必要です。たとえば、Grid レイアウトを使うときに、TextBox の Width プロパティと、この TextBox を配置している列の幅の両方を Auto に設定すると、文字を入力するたびに TextBox の幅が広がってしまいます。

■まとめ

今回は、ユーザー インターフェイスを作成するための、さまざまなコントロールについて学習しました。実際のアプリケーションでは、ユーザー インターフェイスだけでなく、どのように動作させるかといったロジックをプログラミングする必要もあります。

次回は、ユーザー インターフェイスとロジックを連動させるためのイベントについて学んでいきます。