WPF Controls

Content controls: Contain nested elements:
   Label, Button, ToolTip, PopUp, ScrollViewer

Headered content controls: Contain larger blocks of user interface:
   TabItem, GroupBox, Expander

Text controls: Allow users to enter input:
   TextBox, PasswordBox, RichTextBox

List controls: Collections of items in a list: ListBox, ComboBox classes.

Range-based controls: Controls with a Value property:
   Slider, ProgressBar

Date controls: Controls that allow users to select dates:
   Calendar, DatePicker

List controls: ListView, TreeView, DataGrid

Palette controls: Menu, ToolBar, StatusBar, Ribbon
Content Controls

**Diagram:**

- **FrameworkElement**
  - **Control**
    - **ContentControl**
      - **Label**
        - **ButtonBase**
          - **ToolTip**
      - **ScrollViewer**
        - **UserControl**
        - **Window**
      - **HeaderedContentControl**
        - **GroupBox**
        - **TabItem**
        - **Expander**
**Panel** class has a *Children* collection (property) to hold nested elements.

**ContentControl** class has a *Content* property for a single object.

Objects that don’t derive from UIElement: content control calls `ToString()` for displayed text

Objects that derive from UIElement: are displayed using `UIElement.OnRender()`

You can’t always get the effect you want by changing the content of a control. For example some button properties never change: button’s shaded background, its rounded border, and the mouse-over effect that makes it glow when you move the mouse pointer over it. You need to use a new control template.
**ToggleButton** has two states (pushed or unpushed) and stays in its pushed state until you click again to release; *sticky click* behavior.

**CheckBox** and **RadioBox** derive from **ToggleButton**
ToggleButton properties that are used by CheckBox

IsChecked is a nullable Boolean: true, false, or null.
true represents a checked box and false represents an unchecked
null value is an indeterminate state (shaded box): not set,
ambiguous state (from other state controls....)

IsThreeState, can set the ToggleButton into an indeterminate state.

ToggleButton has three events that fire when check box changes states:
Checked, Unchecked, and Indeterminate.

Consider consolidating logic into the Click event (ButtonBase).
The Click event fires whenever the button changes state.
RadioButton

RadioButton is a ToggleButton that adds a single property GroupName. Radio buttons are placed into groups (their container).

Place three RadioButton controls in a StackPanel, they form a group from which you can select just one of the three.

Placing a combination of radio buttons into two separate StackPanel controls, creates two independent groups.

GroupName property allows you to override this behavior.

Multiple groups in the same container

Single group that spans multiple containers

All radio buttons that belong together have the same group name.
GroupName example

<StackPanel>
  <GroupBox Margin="5">
    <StackPanel>
      <RadioButton>Group 1</RadioButton>
      <RadioButton>Group 1</RadioButton>
      <RadioButton>Group 1</RadioButton>
      <RadioButton Margin="0,10,0,0"
      GroupName="Group2">Group 2</RadioButton>
    </StackPanel>
  </GroupBox>
  <GroupBox Margin="5">
    <StackPanel>
      <RadioButton>Group 3</RadioButton>
      <RadioButton>Group 3</RadioButton>
      <RadioButton>Group 3</RadioButton>
      <RadioButton Margin="0,10,0,0"
      GroupName="Group2">Group 2</RadioButton>
    </StackPanel>
  </GroupBox>
</StackPanel>

Two containers holding 3 groups of radio buttons
ScrollViewer

ScrollViewer derives from ContentControl provides a virtual surface that lets users scroll around a much larger element.

ScrollViewer can hold anything, usually it wraps a layout container.

```xml
<ScrollViewer>
  <Grid Margin="3,3,10,3">
    <Grid.RowDefinitions>
      ...
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      ...
    </Grid.ColumnDefinitions>
    <Label Grid.Row="0" Grid.Column="0" Margin="3" VerticalAlignment="Center">Home:</Label>
    <TextBox Grid.Row="0" Grid.Column="1" Margin="3" Height="Auto" VerticalAlignment="Center"></TextBox>
    <Button Grid.Row="0" Grid.Column="2" Margin="3" Padding="2">Browse</Button>
    ...
  </Grid>
</ScrollViewer>
```

Resize to show all content ScrollViewer disappears
GroupBox

The GroupBox is the simplest of the three controls that derives from HeaderedContentControl.

```xml
<GroupBox Header="A GroupBox Test" Padding="5" Margin="5" VerticalAlignment="Top">
    <StackPanel>
        <RadioButton Margin="3">One</RadioButton>
        <RadioButton Margin="3">Two</RadioButton>
        <RadioButton Margin="3">Three</RadioButton>
        <Button Margin="3">Save</Button>
    </StackPanel>
</GroupBox>
```

GroupBox is often used to group related controls:
- radio buttons, checkboxes
- has titled border and rounded corners

GroupBox has no built-in functionality

RadioButton are grouped by placing them into any panel.
The TabItem represents a page in a TabControl. IsSelected property: tab is currently shown
TabStripPlacement property for tabs appear on the side of control

```xml
<TabControl Margin="5">
  <TabItem Header="Tab One">
    <StackPanel Margin="3">
      <CheckBox Margin="3">Setting One</CheckBox>
      <CheckBox Margin="3">Setting Two</CheckBox>
      <CheckBox Margin="3">Setting Three</CheckBox>
    </StackPanel>
  </TabItem>
  <TabItem Header="Tab Two">
    ...
  </TabItem>
</TabControl>
```
ListBox class is a variable-length list for item selection.

Can set SelectionMode property to Multiple or Extended.

Multiple mode, select or deselect any item by clicking it.

Extended mode, hold down Ctrl key to select additional items or Shift key to select a range of items.

In either type of multiple selection list, you use the SelectedItems collection instead of the SelectedItem property to get all the selected items.
WPF doesn't have a CheckedListBox control, but ...

```xml
<ListBox Name="lst SelectionChanged="lst_SelectionChanged"
    CheckBox.Click="lst_SelectionChanged">
    <CheckBox Margin="3">Option 1</CheckBox>
    <CheckBox Margin="3">Option 2</CheckBox>
</ListBox>

private void lst_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (lst.SelectedItem == null) return;
    txtSelection.Text = String.Format("You chose item at position {0}. \n\n" + "Checked state is {1}.", lst.SelectedIndex,
    ((CheckBox)lst.SelectedItem).IsChecked);
}
```
private void cmd_ExamineAllItems(object sender, RoutedEventArgs e)
{
    StringBuilder sb = new StringBuilder();
    foreach (CheckBox item in lst.Items)
    {
        if (item.IsChecked == true)
        {
            sb.Append(item.Content);
            sb.Append(" is checked.");
            sb.Append("\r\n");
        }
    }
    txtSelection.Text = sb.ToString();
}
ComboBox

ComboBox control holds a collection of ComboBoxItem objects

ComboBox control uses a drop-down list, single item selection

To select an item, by typing text IsEditable property must be true, and text-only ComboBoxItem or object that provides ToString(), for matching

```csharp
private void colorSet(object sender, SelectionChangedEventArgs e)
{
    int index = redComboBox.SelectedIndex;
    MessageBox.Show("color set to index "+ index);
}
```
<Window x:Class="ComboBoxDemo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Demo" Width="200" Height="200">
    <StackPanel Margin="0,0,2,0">
        <GroupBox x:Name="colorGroup" Header="Set Color" Margin="5"
Height="150">
            <StackPanel x:Name="colorSP" Height="47" Margin="0,0,0,-12"
Width="77" VerticalAlignment="Top"
HorizontalAlignment="Left">
                <ComboBox x:Name="redComboBox" SelectedIndex="1"
SelectionChanged="colorSet" Margin="0,10,0,0"
Background="Red">
                    <ComboBoxItem Background="Black" Content="0"/>
                    <ComboBoxItem Background="DarkRed" Content="139"/>
                    <ComboBoxItem Background="Red" Content="255"/>
                </ComboBox>
            </StackPanel>
        </GroupBox>
    </StackPanel>
</Window>
Text Controls

WPF has 3 text controls: **TextBox**, **RichTextBox**, and **PasswordBox**.

TextBox and RichTextBox controls derive from TextBoxBase.

RichTextBox has a FlowDocument content that can contain a complex combination of elements. Part of WPF Documents ...

TextBox has a string Text property, text selection, and spell checking (customizable dictionary) ...

If displaying text set to read-only mode.

The PasswordBox derives from Control

The PasswordBox has a string Password property content and uses SecureString.

SecureString is encrypted in memory, its key generated randomly and never written to disk.
Range based controls

WPF includes three range based controls. ScrollBar, ProgressBar, and Slider derive from the RangeBase class. These controls take a numeric value between a minimum and maximum value.

Properties of RangeBase

Value  The current value of the control. Default starts at 0. Value is a double. Has a ValueChanged event

Maximum  The upper limit

Minimum  The lower limit

SmallChange  Amount Value property is adjusted up or down for a small change; depends on the control ScrollBar and Slider, amount the value changes with arrow keys.

LargeChange  Amount the Value property is adjusted up or down for a large change; depends on the control For the ScrollBar and Slider value changes for Page Up and Page Down
Date Controls

WPF includes two date controls: the Calendar and the DatePicker.

Both support date selection.

Complex properties, check out as needed.
Element binding

The simplest data-binding occurs when the source object is a WPF element and the source property is a dependency property.

Source object can be: another WPF element, an ADO.NET data object (such as DataTable and DataRow), or a developer's data-only object.

Dependency properties have built-in support for change notification; change the value of the dependency property in the source object, the bound property in the target object is updated immediately.

Majority of data binding, binds elements to data objects.

   The display the information extracted from an external source (database or file)

Element-to-element binding can automate the way elements interact: when a user modifies a control, another element is updated without writing code. (XAML w/o code can run in a browser).

The name Path is used instead of Property because the Path might point to a property of a property; e.g., Grid.Row, Content.Children[0]
<Window x:Class="SliderFontBinding.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Slider Name="sliderFontSize" Margin="3" Minimum="1"
            Maximum="40" Value="10" TickFrequency="1"
            TickPlacement="TopLeft">
        </Slider>

        <TextBlock Margin="10,50,10,103" Text="Simple Text"
            Name="lblSampleText"
            FontSize="{Binding ElementName=sliderFontSize,
                Path=Value}">
        </TextBlock>
    </Grid>
</Window>

// MainWindow.xaml.cs code ...
// using statements here
namespace SliderFontBinding {
    public partial class MainWindow : Window {
        public MainWindow() {
            InitializeComponent();
        }
    }
}
Binding modes

OneWay  The target property is updated when the source property changes.

TwoWay  The target property is updated when the source property changes, and the source property is updated when the target property changes.

OneTime  The target property is set initially based on the source property value; an initialization. Later changes are ignored reduces overhead when you know the source property doesn't change.

OneWayToSource  Reverse of OneWay  Target updates source.

Default  The type of binding depends on the target property. TwoWay for usersettable properties or OneWay
Bindings usually set in XAML.

```csharp
// Bindings in code
Binding binding = new Binding();
binding.Source = sliderFontSize;
binding.Path = new PropertyPath("Value");
binding.Mode = BindingMode.TwoWay;
lblSampleText.SetBinding(TextBlock.FontSizeProperty, binding);

//Retrieve bindings
Binding binding = BindingOperations.GetBinding(lblSampleText, TextBlock.FontSize);
// Get the source element.
Slider boundObject = (Slider)expression.ResolvedSource;
// Get data from the source element, including its bound property.
string boundData = boundObject.FontSize;
```
Data access (database)

Database code is not embedded in the code-behind class for a window but encapsulated in a dedicated class.

**Open and close connections quickly:** Open the database connection in every method call, and close it before the method ends.

**Implement error handling:** Use error handling to make sure that connections are closed even if an exception occurs.

**Follow stateless design practices:** Accept all the information needed for a method in its parameters, and return all the retrieved data through the return value (multithreaded application or host your database component on a server).

**Store the connection string in one place:** Ideally, this is the application's configuration file.
A **DataSet** is an in-memory cache of data with properties:

- Tables (collection of **DataTable**)
- Relations (collection of **DataRelation**) properties.

Navigation from one **DataTable** to another within a **DataSet**.

**DataRow** property stores items in **DataTable** row.

**How-to bind data source to dataset tutorial.**

**Dimitris Nemtsov, DataGrid and SQLite database example**

```csharp
// Connect and return a DataTable w/values
public DataTable GetProducts() {
    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand("GetProducts", con);
    cmd.CommandType = CommandType.StoredProcedure;
    SqlDataAdapter adapter = new SqlDataAdapter(cmd);
    DataSet ds = new DataSet();
    adapter.Fill(ds, "Products");
    return ds.Tables[0];
}
```
ListView

ListView is designed for displaying different *views* of the same data.

ListView is good for a multicolumn view that displays several pieces of information about each data item.

The ListView derives from the ListBox class and extends with a View property.

Reusable views: The ListView separates all the view-specific details into one object; views that are data-independent

Multiple views: Can switch between multiple views with the same list. (Windows Explorer views on files and folders.)

Better organization: The view object wraps two styles: one for the root ListView control and one for the individual items in the list. Preserves their relationship.

For example a column-based ListView needs to keep its column headers and column data lined up.
The GridView derives from ViewBase and represents a list view with multiple columns.

Define columns by adding GridViewColumn objects to the GridView.Columns collection.

```xml
<ListView Margin="5" Name="lstProducts">
    <ListView.View>
        <GridView>
            <GridView.Columns>
                <GridViewColumn Header="Name" DisplayMemberBinding="{Binding Path=ModelName}" />
                <GridViewColumn Header="Model" DisplayMemberBinding="{Binding Path=ModelNumber}" />
                <GridViewColumn Header="Price" DisplayMemberBinding="{Binding Path=UnitCost, StringFormat=\{0:C\}}" />
            </GridView.Columns>
        </GridView>
    </ListView.View>
</ListView>
```

DataGrid is more customizable.
TreeView

TreeView is a specialized ItemsControl that hosts TreeViewItem objects. TreeViewItem is not a content control, it is a separate ItemsControl, with the ability to hold more TreeViewItem objects. This enables layered data display.

TreeView is customized with templates and TreeViewItem controls. Josh Smith’s example transforms the TreeView into an organizational chart.
<Window x:Class="WpfTutorialSamples.TreeView_control.TreeViewSample"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="TreeViewSample" Height="200" Width="250">
  <Grid Margin="10">
    <TreeView>
      <TreeViewItem Header="Level 1" IsExpanded="True">
        <TreeViewItem Header="Level 2.1" />
        <TreeViewItem Header="Level 2.2" IsExpanded="True">
          <TreeViewItem Header="Level 3.1" />
          <TreeViewItem Header="Level 3.2" />
        </TreeViewItem>
        <TreeViewItem Header="Level 2.3" />
      </TreeViewItem>
    </TreeView>
  </Grid>
</Window>
DataGrid

WPF’s full-featured data display tool. Displays data in columns and rows, has formatting (freeze columns, format rows and columns), and supports in-place editing.

Each row corresponds to a separate object, and each column corresponds to a property in that object.

Selection model: a row, multiple rows, or combination of cells.

Use the DataGrid as an all-in-one data editor for simple and complex data.

For nonstring properties, the DataGrid calls ToString(), {numbers, dates, and simple data types}.

For complex data object explicitly define columns: bind to a subproperty, use a value converter, or apply a template
DataGrid display properties

- **RowBackground**  
  brush for background of every row or

- **AlternatingRowBackground**  
  alternate rows (even white, odd gray

- **ColumnHeaderHeight**  
  Height of column header row (at top)

- **RowHeaderWidth**  
  Width of row headers column (at left).
  Shows the currently selected row (arrow),
  row is being edited (arrow in a circle).

- **ColumnWidth**  
  Default width of columns (can be resized)

- **RowHeight**  
  Height of every row. rows cannot be resized

- **GridLinesVisibility**  
  Show Horizontal, Vertical, None, or All.

- **VerticalGridLinesBrush** and **HorizontalGridLinesBrush**

- **HeadersVisibility**  
  Show Column, Row, All, or None

- **HorizontalScrollBarVisibility** and **VerticalScrollBarVisibility**  
  (Auto, Visible, Hidden).
DataGrid column display

DataGrid supports several types of columns (classes derived from DataGridColumn)

DataGridTextColumn: choice for most data types; editing as TextBox

DataGridCheckBoxColumn: shows a check box, Boolean (or nullable Boolean) values; editing as check box.

DataGridHyperlinkColumn: shows a clickable link, use with Frame or NavigationWindow to navigate an external website.

DataGridComboBox: looks like a DataGridTextColumn initially, editing as a drop-down ComboBox (constrain edits).

DataGridTemplateColumn: define a data template for displaying column values with templates (e.g., display image or a DatePicker)
DataGrid column template

<DataGridTemplateColumn>
  <DataGridTemplateColumn.CellTemplate>
    <DataTemplate>
      <Image Stretch="None" Source="{Binding Path=ProductImagePath, Converter={StaticResource ImagePathConverter}}"/>
    </DataTemplate>
  </DataGridTemplateColumn.CellTemplate>
</DataGridTemplateColumn>

Note: XAML string can't span lines like above.

This example assumes you’ve added the ImagePathConverter value converter to the UserControl. Resources collection:

<UserControl.Resources>
  <local:ImagePathConverter x:Key="ImagePathConverter"/>
</UserControl.Resources>
DataGrid examples