C#.NET WPF

Introduction

Ressources de notre formation

- Pour les débutants…
  - Apress : Illustrated WPF by Daniel M. Solis
  - Couvre les concepts et mécanismes de base
- … jusqu'aux professionnels
  - Apress : Applied WPF 4 in Context de Raffaele Garofalo
  - Couvre les techniques actuelles d'architecture
- Référence MSDN (Framework 4.5)
  - http://msdn.microsoft.com/en-us/library/ms754130

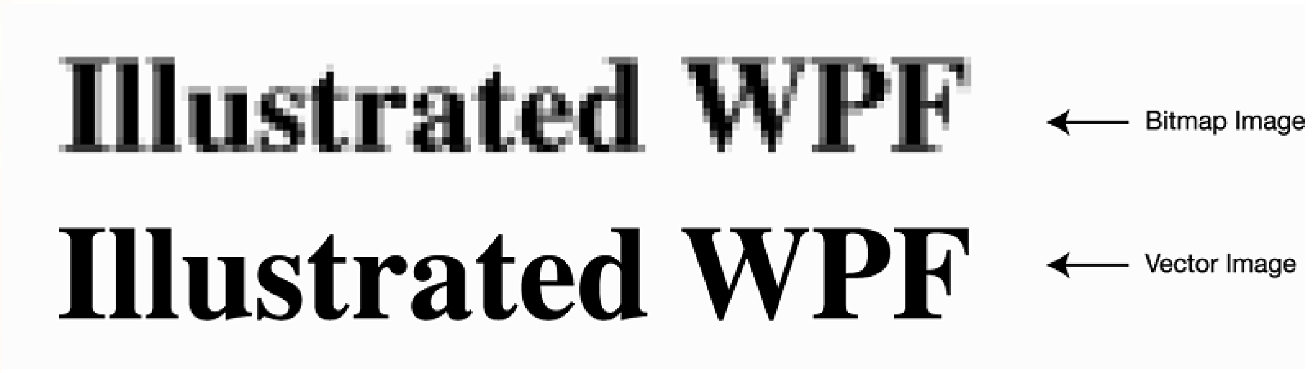
WPF ?

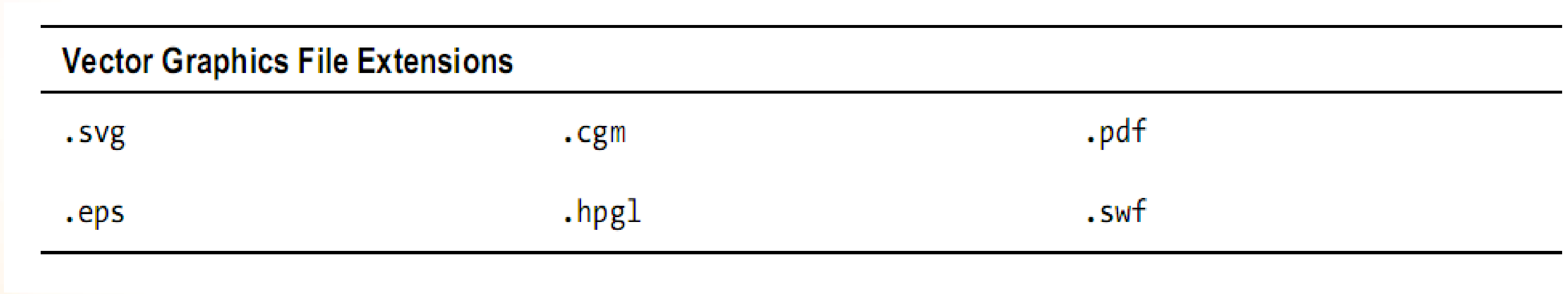
- **W**indows **P**resentation **F**oundation
  - A graphical subsystem
  - To create rich client applications for Windows systems
- User interface based on **XAML**
  - E**x**tensible **A**pplication **M**arkup **L**anguage
  - To decouple the UI code from the C# core development process

Introducing WPF

- Version 4.0
  - with the .NET Framework 4
- To build two different types of WPF applications
  - A stand-alone application .exe
    - for Windows
  - An in-browser application .xbap
    - for Internet Explorer
- Powerful UI controls
  - Office-style
- Vectorial UI technology

Bitmap vs Vector Image





Vector Graphics File Extensions		
.svg	.cgm	.pdf
.eps	.hpgl	.swf

Source : Illustrated WPF – Daniel M. Solis – Apress 2009

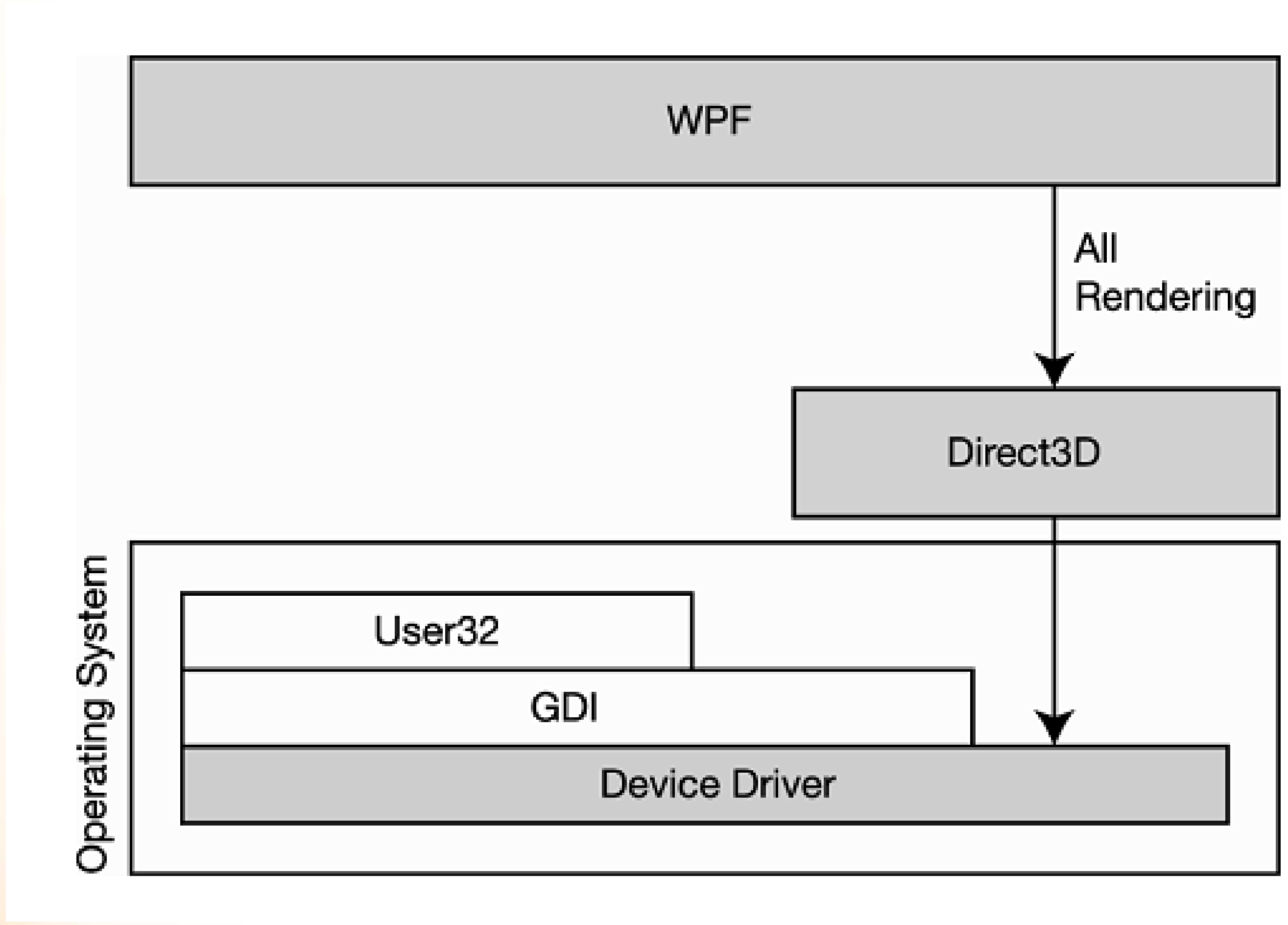
Les outils de travail Microsoft

- Visual Studio 2015
    - Development-oriented audience
- Expression Studio
    - Expression Blend
        - Design-oriented audience

Microsoft Expression Blend

- Sketchflow
  - To build dynamic mock-up applications for WPF, Silverlight and Windows Phone 7
- Import
  - Files from graphic software without losing the structure
- Behaviors
  - Animations to your UI
- Sample data, transitions, intellisense, templates

Rendering under WPF



C#.NET WPF

XAML

Introducing XAML

- Declarative Markup Language
  - Extension of type .xaml
  - Encoding in UTF-8
  - For WPF, Silverlight and Workflow Foundation
- Sample XAML Code

```
<StackPanel>
   <ListBox>
      <ListBoxItem Content="One" />
      <ListBoxItem Content="Two" IsSelected="True" />
      <ListBoxItem Content="Three" />
   </ListBox>
</StackPanel>
```

Namespaces and Root Elements

- The xmlns attribute indicates the default XAML namespace

```
<Window x:Class="WpfApplicationDemo1.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
    <Grid>
    </Grid>
</Window>
```

- In a WPF context, the root element of a XAML file will be Window, Application, UserControl, Page,…

Attributes Syntax and Content

- Any value is specified as a String value type

```
<Button Height="40" Width="120">
    <Button.Content>
        <StackPanel Orientation="Horizontal">
            <Image Source="home_go_32.png" Margin="0,0,5,0" />
            <TextBlock VerticalAlignment="Center" Text="HomePage"/>
        </StackPanel>
    </Button.Content>
</Button>
```

- In XAML, the content can be simple or complex, like a panel with nested UI controls

The Code Behind

- One file MainWindow.xaml
  - XAML
- One file MainWindow.xaml.cs
  - C#
  - Code-behind file

Events in XAML

- When you click a Button, the UI raises an event Click associated to that button

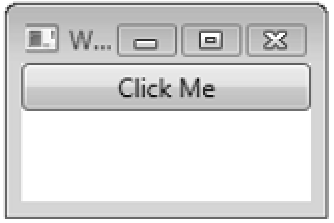
```
// XAML File
<Button Click=”Click_Event”>Hello World</Button>

// C# Code-behind
private void Click_Event(object sender, RoutedEventArgs e)
{
   Console.WriteLine(“Hello World”);
}
```

Window1.xaml

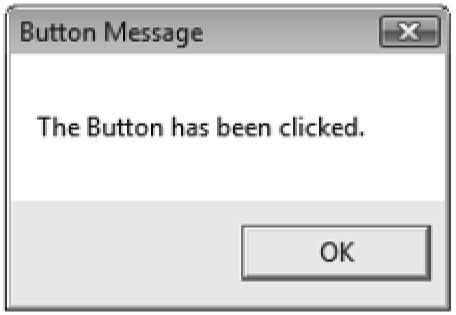
```
<Window x:Class="ButtonOnClick.Window1" ... Height="120" Width="150">
   <StackPanel>
      <Button Click="Button_Click">Click Me</Button>
   </StackPanel>
</Window>
```

Window1.xaml.cs

```
public partial class Window1 : Window
{
   public Window1() { InitializeComponent(); }

   private void Button_Click( object sender, RoutedEventArgs e )
   {
      MessageBox.Show( "The Button has been clicked.", "Button Message" );
   }
}
```



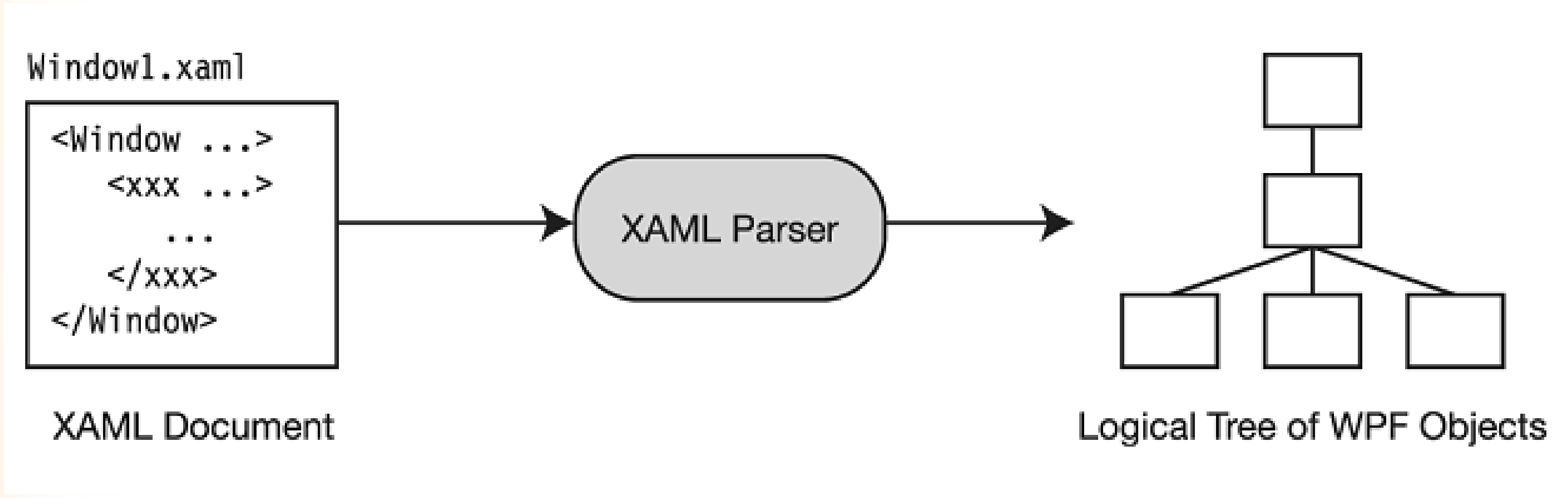
The Button in the Program Window



The Button has been clicked.

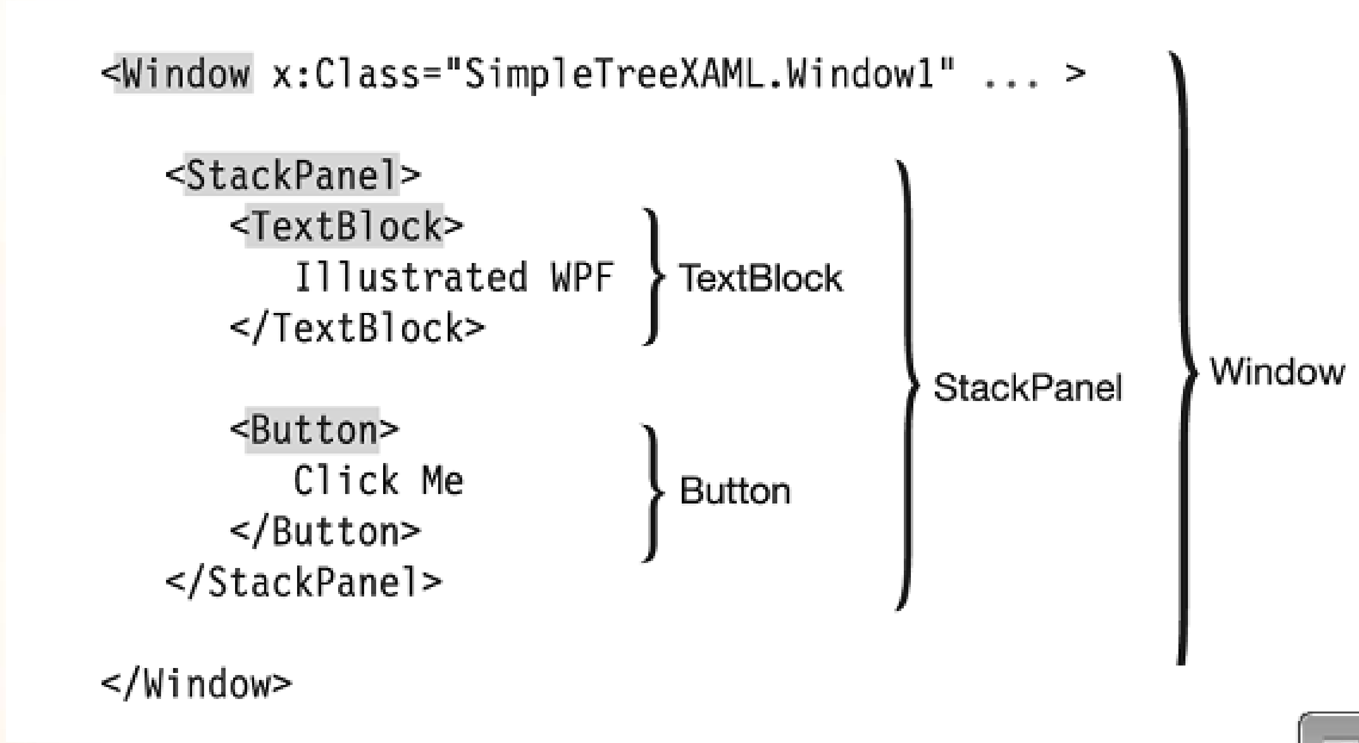
The MessageBox Produced by Clicking the Button

XAML Parser

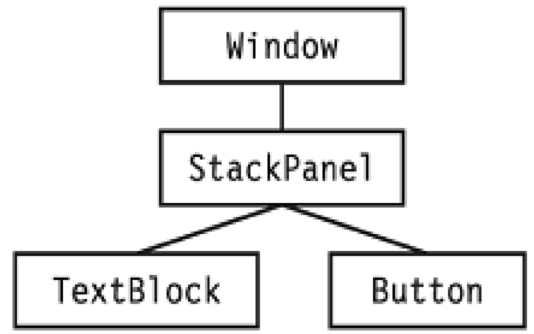


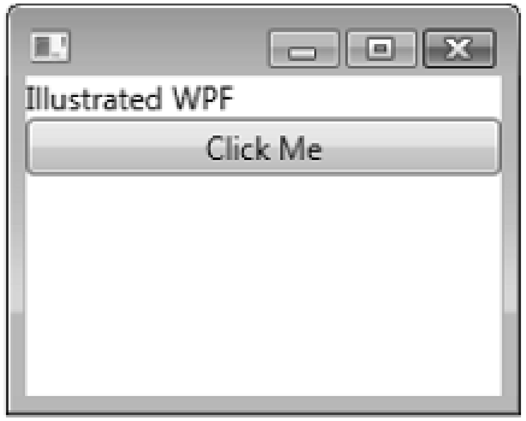
Window1.xaml
<Window ...>
<xxx ...>
...
</xxx>
</Window>
XAML Document
XAML Parser
Logical Tree of WPF Objects

Logical Tree of WPF Objects



```
<Window x:Class="SimpleTreeXAML.Window1" ... >

   <StackPanel>
      <TextBlock>
         Illustrated WPF
      </TextBlock>

      <Button>
         Click Me
      </Button>
   </StackPanel>

</Window>
```



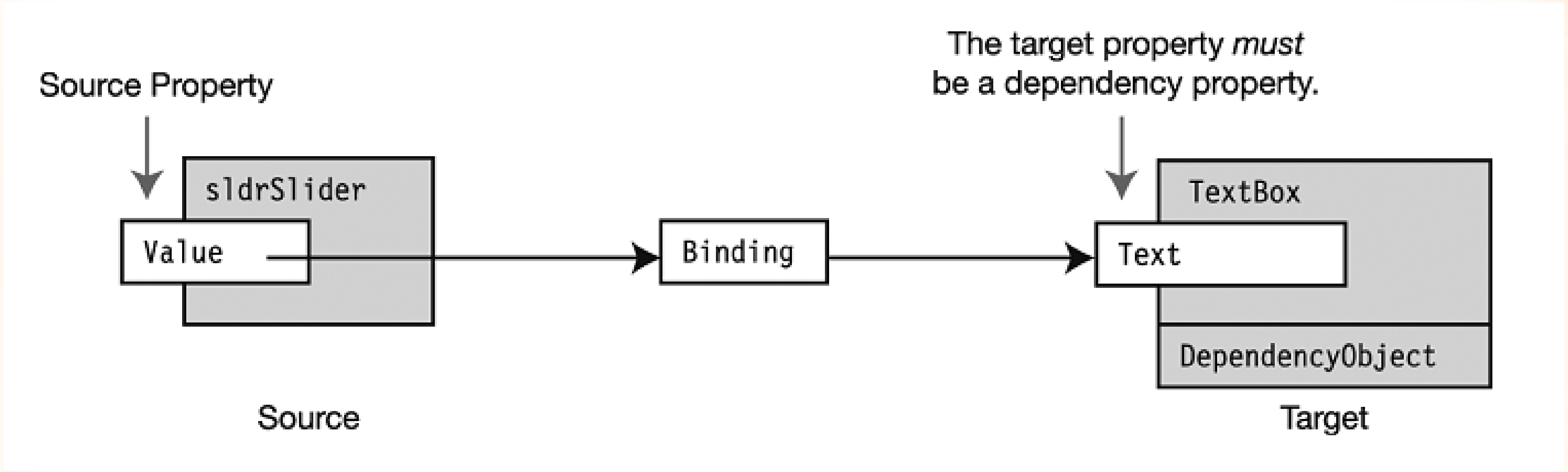


Source : Illustrated WPF – Daniel M. Solis – Apress 2009

C#.NET WPF

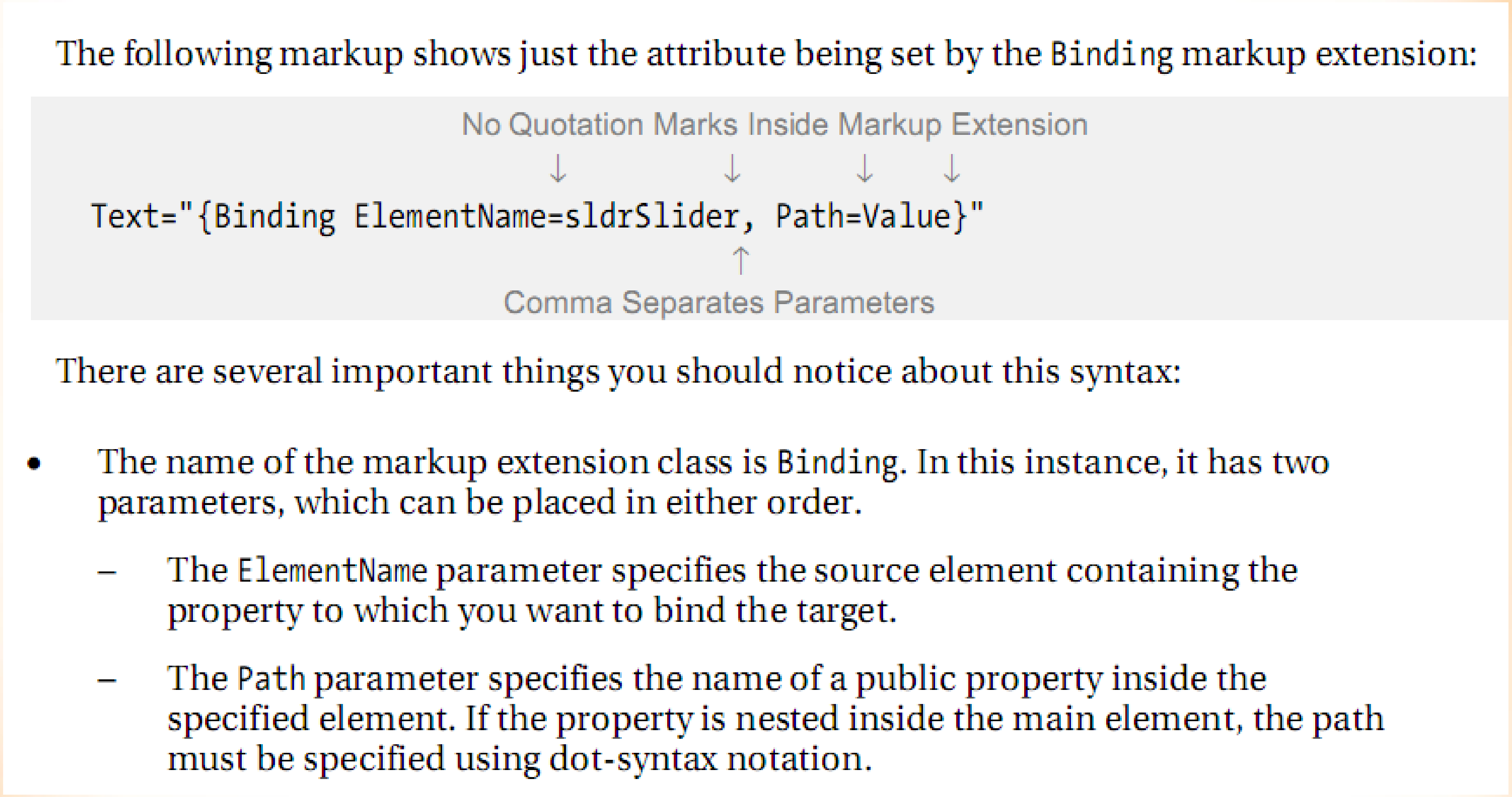
Binding

Source to Target



Source : Illustrated WPF – Daniel M. Solis – Apress 2009

Binding with a Markup Extension



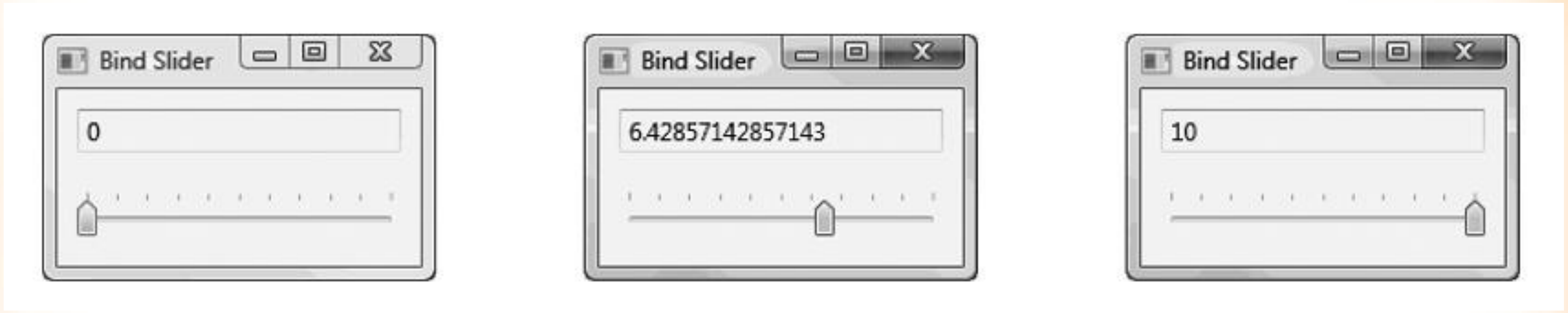
The following markup shows just the attribute being set by the Binding markup extension:

```
                  No Quotation Marks Inside Markup Extension
                  ↓              ↓            ↓      ↓
Text="{Binding ElementName=sldrSlider, Path=Value}"
                                     ↑
                          Comma Separates Parameters
```

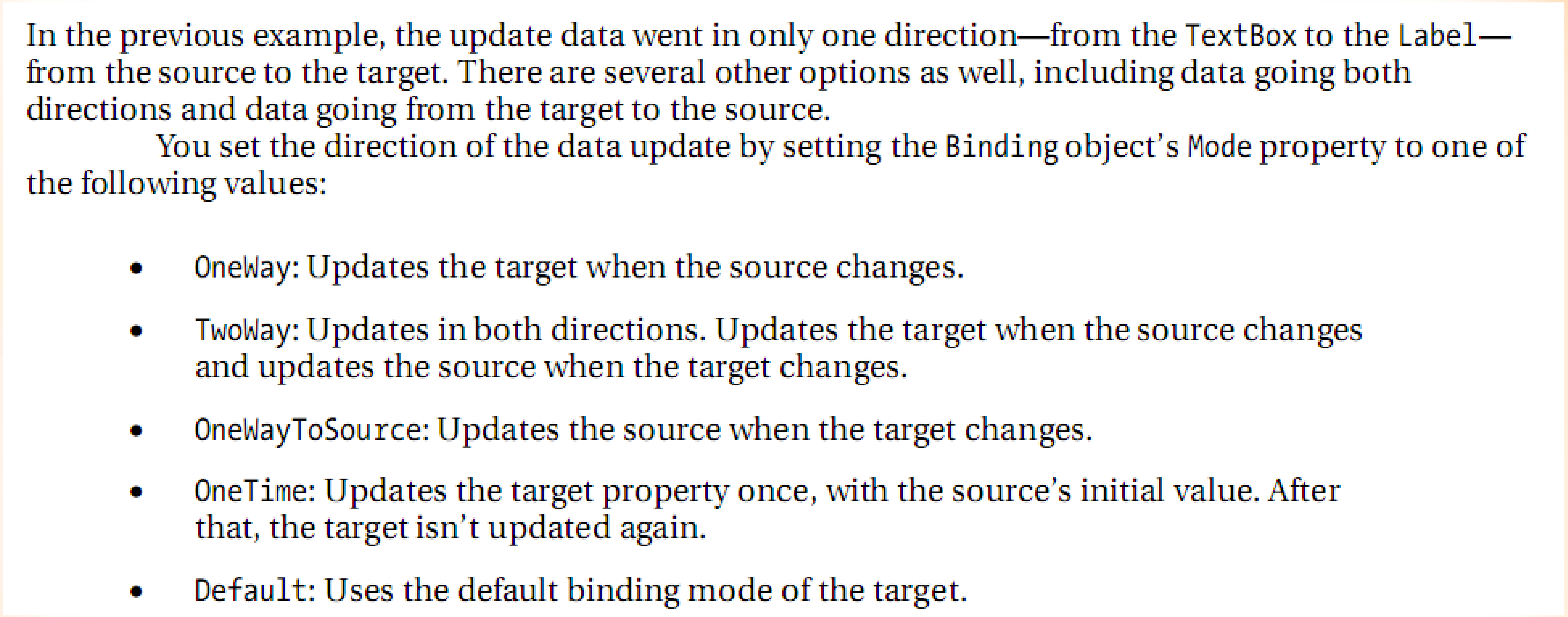
There are several important things you should notice about this syntax:

- The name of the markup extension class is Binding. In this instance, it has two parameters, which can be placed in either order.
  - The ElementName parameter specifies the source element containing the property to which you want to bind the target.
  - The Path parameter specifies the name of a public property inside the specified element. If the property is nested inside the main element, the path must be specified using dot-syntax notation.

Example



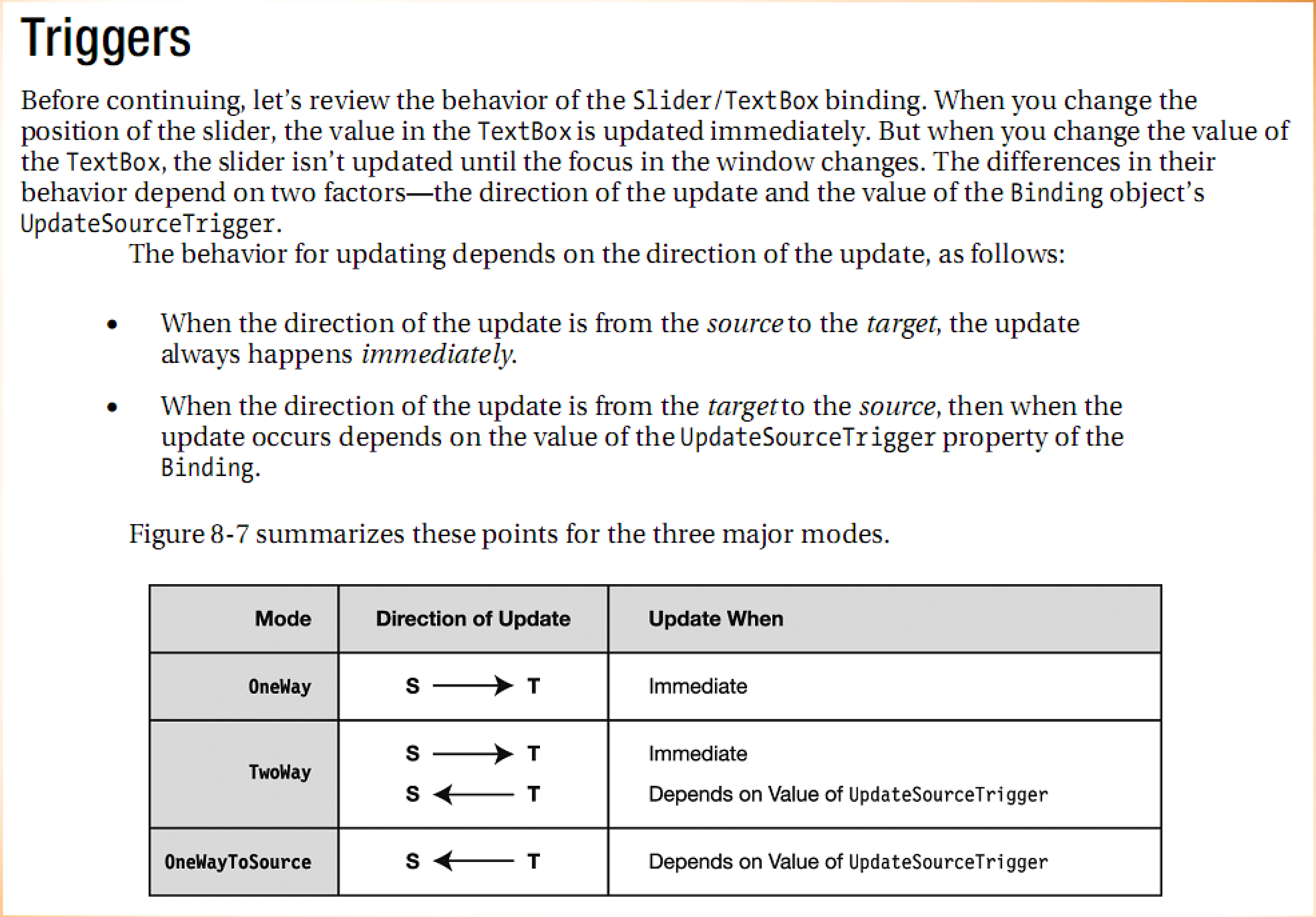
Binding Direction



In the previous example, the update data went in only one direction—from the `TextBox` to the `Label`—from the source to the target. There are several other options as well, including data going both directions and data going from the target to the source.

You set the direction of the data update by setting the `Binding` object's `Mode` property to one of the following values:

- `OneWay`: Updates the target when the source changes.
- `TwoWay`: Updates in both directions. Updates the target when the source changes and updates the source when the target changes.
- `OneWayToSource`: Updates the source when the target changes.
- `OneTime`: Updates the target property once, with the source's initial value. After that, the target isn't updated again.
- `Default`: Uses the default binding mode of the target.



Triggers

Before continuing, let's review the behavior of the `Slider`/`TextBox` binding. When you change the position of the slider, the value in the `TextBox` is updated immediately. But when you change the value of the `TextBox`, the slider isn't updated until the focus in the window changes. The differences in their behavior depend on two factors—the direction of the update and the value of the `Binding` object's `UpdateSourceTrigger`.

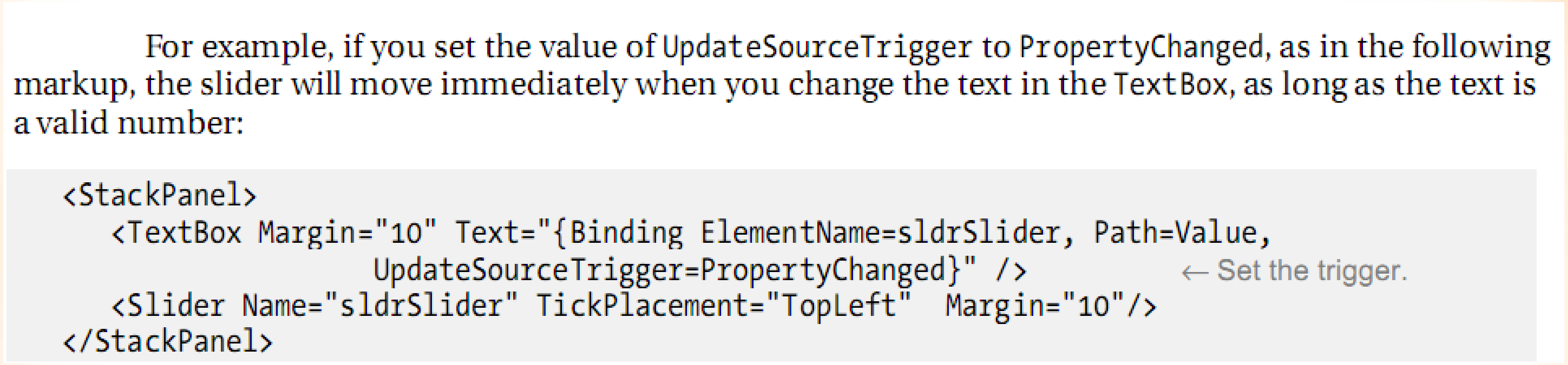
The behavior for updating depends on the direction of the update, as follows:

- When the direction of the update is from the *source* to the *target*, the update always happens *immediately*.
- When the direction of the update is from the *target* to the *source*, then when the update occurs depends on the value of the `UpdateSourceTrigger` property of the `Binding`.

Figure 8-7 summarizes these points for the three major modes.

Mode	Direction of Update	Update When
`OneWay`	$S \longrightarrow T$	Immediate
`TwoWay`	$S \longrightarrow T$	Immediate
	$S \longleftarrow T$	Depends on Value of `UpdateSourceTrigger`
`OneWayToSource`	$S \longleftarrow T$	Depends on Value of `UpdateSourceTrigger`

Target to Source



For example, if you set the value of UpdateSourceTrigger to PropertyChanged, as in the following markup, the slider will move immediately when you change the text in the TextBox, as long as the text is a valid number:

```
<StackPanel>
   <TextBox Margin="10" Text="{Binding ElementName=sldrSlider, Path=Value,
                    UpdateSourceTrigger=PropertyChanged}" />          ← Set the trigger.
   <Slider Name="sldrSlider" TickPlacement="TopLeft"  Margin="10"/>
</StackPanel>
```

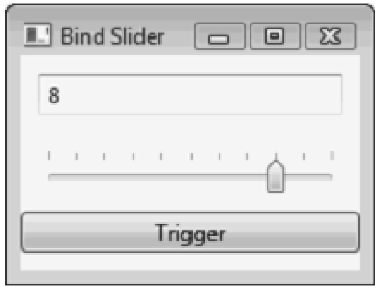


Figure 8-8. *Update the source explicitly when the Trigger button is clicked.*

To do that, you set the UpdateSourceTrigger to Explicit as shown in the following markup:

```
<StackPanel>
   <TextBox Margin="10" Name="tbValue"
            Text="{Binding ElementName=sldrSlider, Path=Value,
                   UpdateSourceTrigger=Explicit}" />       ← Set the trigger.
   <Slider Name="sldrSlider" TickPlacement="TopLeft"  Margin="10"/>
   <Button Click="Button_Click">Trigger</Button>
</StackPanel>
```

In the code-behind, you need to create an event handler for the button, to trigger the explicit update. You accomplish this by getting the BindingExpression for the target property and calling its UpdateSource method:

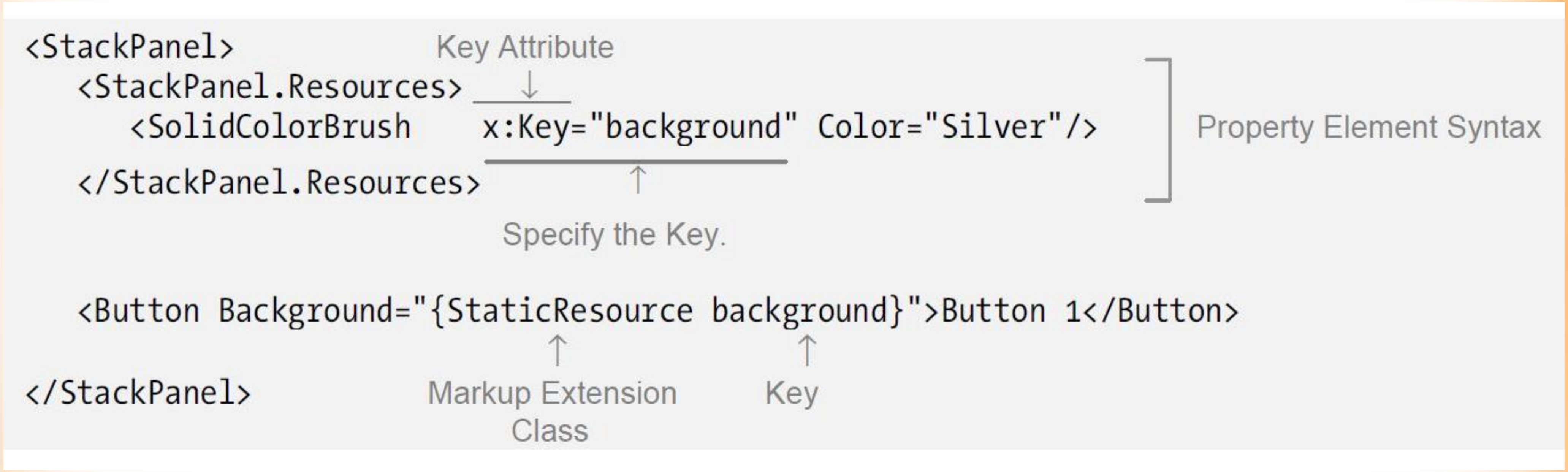
```
public partial class Window1 : Window
{
   public Window1() { InitializeComponent(); }

   private void Button_Click( object sender, RoutedEventArgs e )
   {
      BindingExpression be =
            tbValue.GetBindingExpression( TextBox.TextProperty );
      be.UpdateSource();
   }
}
```

C#.NET WPF

Resources

Static Resource

- Demo : WpfApplicationExample.sln



```
<StackPanel>
   <StackPanel.Resources>
      <SolidColorBrush    x:Key="background" Color="Silver"/>
   </StackPanel.Resources>

   <Button Background="{StaticResource background}">Button 1</Button>

</StackPanel>
```

Dynamic Resource

- Déclaration
  - … x:Key="gradBrush" …
- Utilisation(s)
  - Background="{DynamicResource gradBrush}"
- Modification(s)
  - this.Resources["gradBrush"] = Brushes.Silver;

C#.NET WPF

Styles

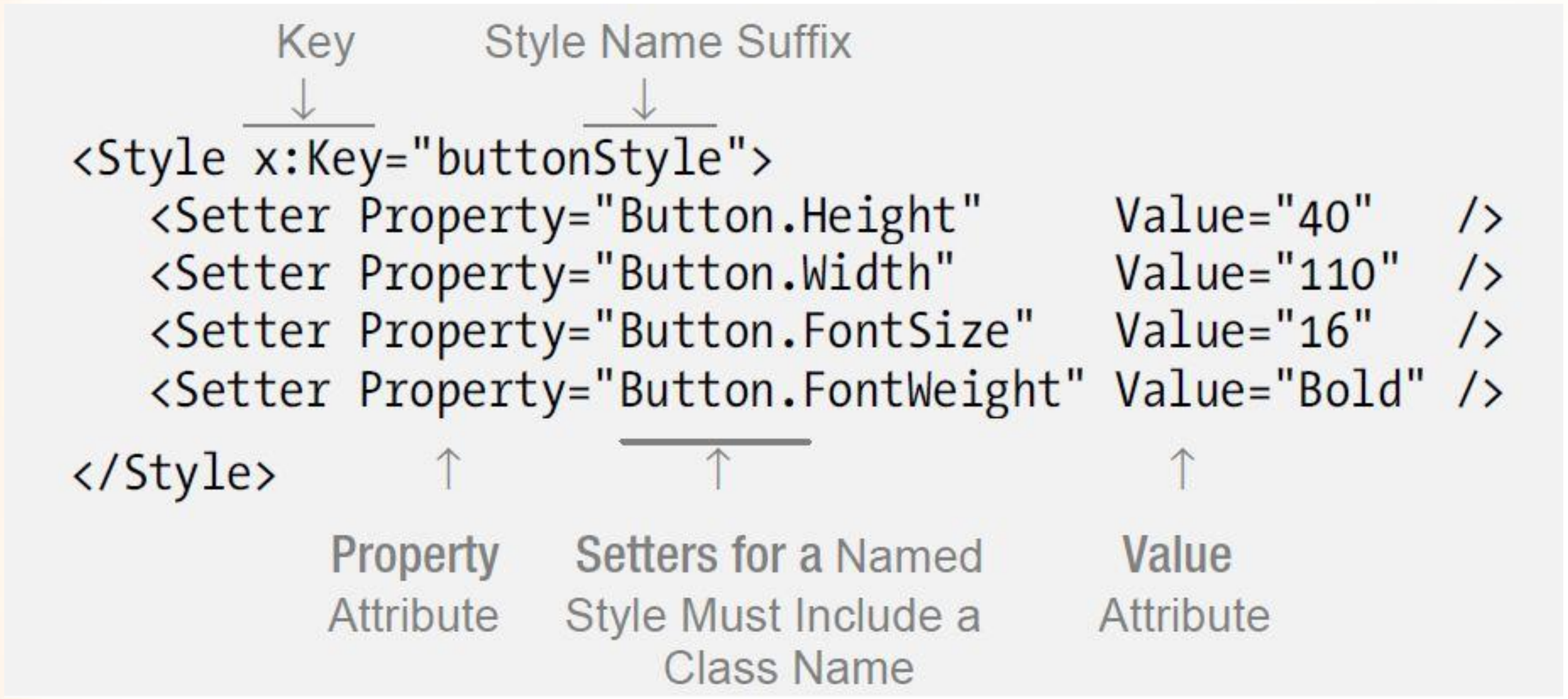
Styles

- Apply a group of property settings to a number of different elements
- Styles are declared as resources

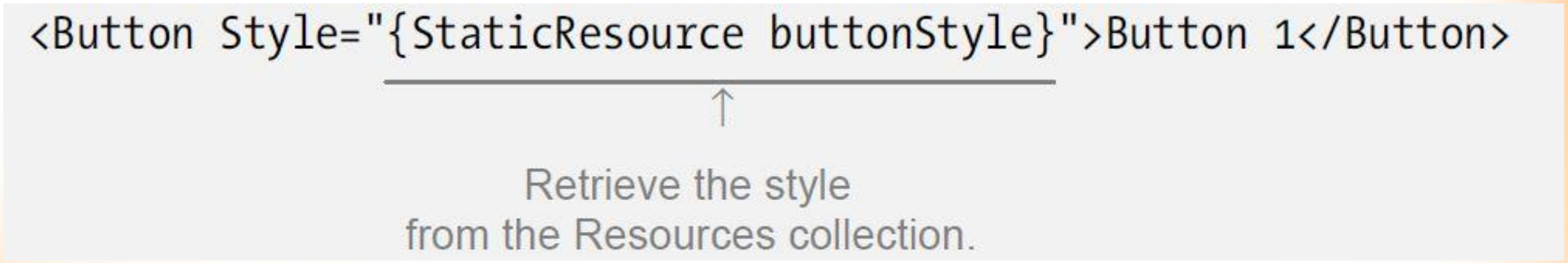
```
<Window.Resources>
  <Style ...>
    ...
  </Style>
</Window.Resources>
```

- Named Styles and Targeted Styles

Named Styles



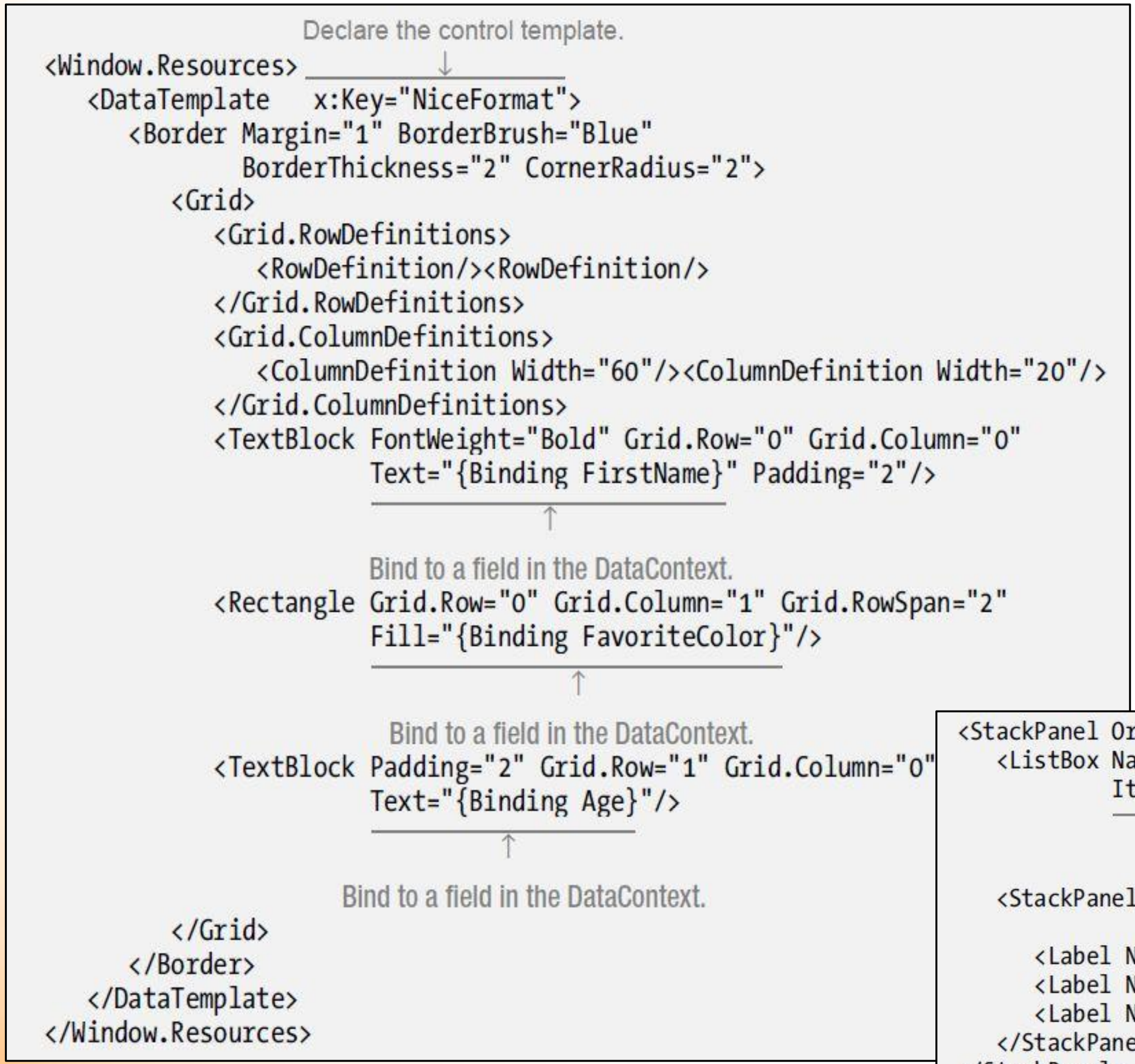
```
<Style x:Key="buttonStyle">
   <Setter Property="Button.Height"     Value="40"   />
   <Setter Property="Button.Width"      Value="110"  />
   <Setter Property="Button.FontSize"   Value="16"   />
   <Setter Property="Button.FontWeight" Value="Bold" />
</Style>
```



```
<Button Style="{StaticResource buttonStyle}">Button 1</Button>
```

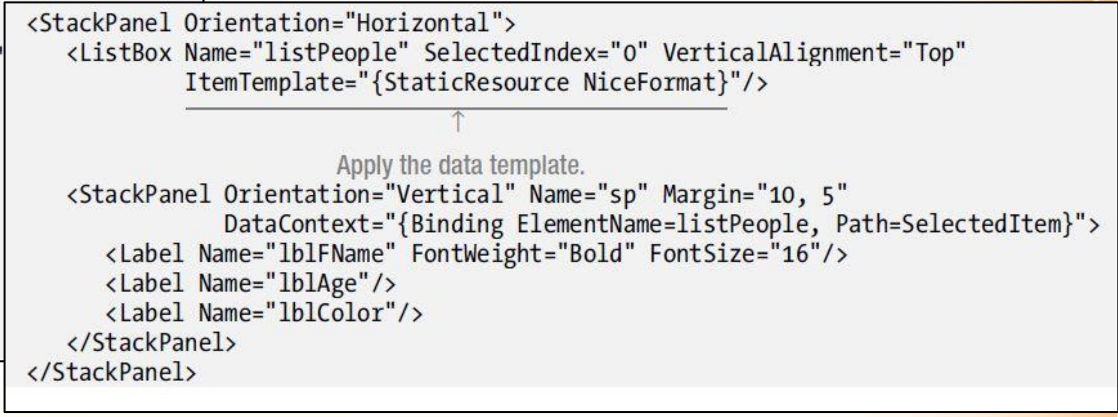
Source : Illustrated WPF – Daniel M. Solis – Apress 2009

Data Templates



```
<Window.Resources>
   <DataTemplate   x:Key="NiceFormat">
      <Border Margin="1" BorderBrush="Blue"
              BorderThickness="2" CornerRadius="2">
         <Grid>
            <Grid.RowDefinitions>
               <RowDefinition/><RowDefinition/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
               <ColumnDefinition Width="60"/><ColumnDefinition Width="20"/>
            </Grid.ColumnDefinitions>
            <TextBlock FontWeight="Bold" Grid.Row="0" Grid.Column="0"
                       Text="{Binding FirstName}" Padding="2"/>

            <Rectangle Grid.Row="0" Grid.Column="1" Grid.RowSpan="2"
                       Fill="{Binding FavoriteColor}"/>

            <TextBlock Padding="2" Grid.Row="1" Grid.Column="0"
                       Text="{Binding Age}"/>

         </Grid>
      </Border>
   </DataTemplate>
</Window.Resources>
```

Declare the control template.
Bind to a field in the DataContext.
Bind to a field in the DataContext.
Bind to a field in the DataContext.



```
<StackPanel Orientation="Horizontal">
   <ListBox Name="listPeople" SelectedIndex="0" VerticalAlignment="Top"
            ItemTemplate="{StaticResource NiceFormat}"/>

   <StackPanel Orientation="Vertical" Name="sp" Margin="10, 5"
               DataContext="{Binding ElementName=listPeople, Path=SelectedItem}">
      <Label Name="lblFName" FontWeight="Bold" FontSize="16"/>
      <Label Name="lblAge"/>
      <Label Name="lblColor"/>
   </StackPanel>
</StackPanel>
```

Apply the data template.

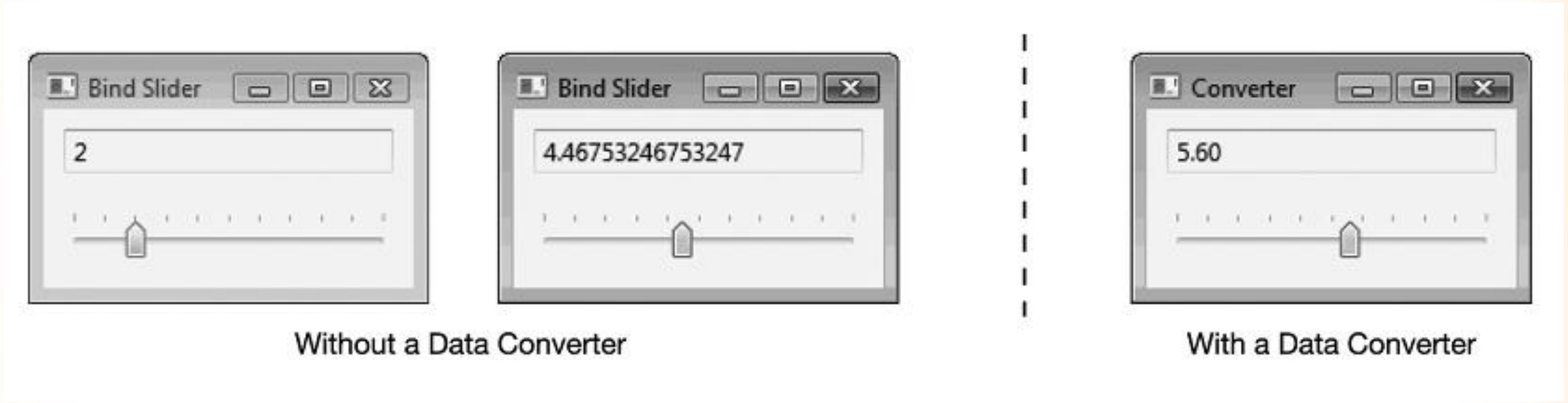
Source : Illustrated WPF – Daniel M. Solis – Apress 2009

C#.NET WPF

Data Converters

Data Converters

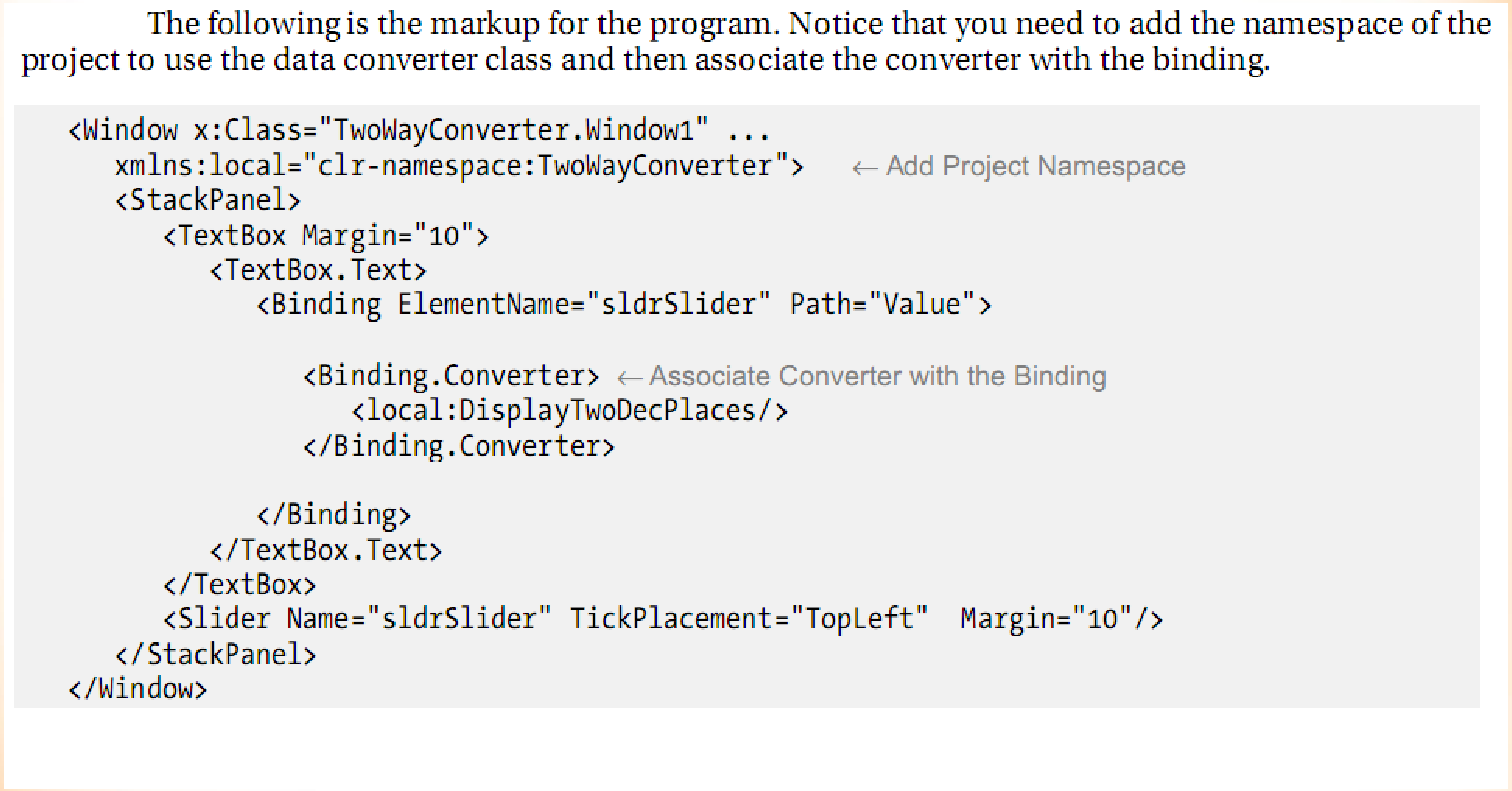
- To manipulate the data between the source and the target



As an example, the following markup and code implement a TextBox/Slider window where the TextBox always shows two decimal places, as shown in the third window of Figure 8-9. The following data converter class, called DisplayTwoDecPlaces, is in a separate file called DisplayTwoDecPlaces.cs. The namespace of my project, in this particular case, is TwoWayConverter.

```
using System;
using System.Windows.Data;

namespace TwoWayConverter
{
   [ValueConversion( typeof( double ), typeof( string ) )]
   public class DisplayTwoDecPlaces : IValueConverter
   {
      public object Convert( object value, Type targetType,
         object parameter, System.Globalization.CultureInfo culture )
      {
         double dValue = (double) value;
         return dValue.ToString( "F2" );
      }

      public object ConvertBack( object value, Type targetType,
         object parameter,System.Globalization.CultureInfo culture )
      {
         double dValue;
         double.TryParse( (string) value, out dValue );
         return dValue;
      }
   }
}
```

Converter with the Binding



The following is the markup for the program. Notice that you need to add the namespace of the project to use the data converter class and then associate the converter with the binding.

```
<Window x:Class="TwoWayConverter.Window1" ...
   xmlns:local="clr-namespace:TwoWayConverter">   ← Add Project Namespace
   <StackPanel>
      <TextBox Margin="10">
         <TextBox.Text>
            <Binding ElementName="sldrSlider" Path="Value">

               <Binding.Converter> ← Associate Converter with the Binding
                  <local:DisplayTwoDecPlaces/>
               </Binding.Converter>

            </Binding>
         </TextBox.Text>
      </TextBox>
      <Slider Name="sldrSlider" TickPlacement="TopLeft"  Margin="10"/>
   </StackPanel>
</Window>
```

C#.NET WPF

Datacontext

Datacontext & binding

- Provide data for binding from a class, …
- inheritance

Datacontext & binding - sample

Xaml.cs :

```
EtudiantsData etuds = new EtudiantsData();
this.datagrid.DataContext =etuds;
```

Datacontext & binding - sample

EtudiantsData.cs :

```
public class EtudiantsData
{
        private IList<Etudiant> etudList;

        public ListEtud {
                get { return etudList;}
        }
}

public class Etudiant {
      private string _nom;
      public Nom {
              get { return _nom;}
}
```

Datacontext & binding - sample

.xaml :

```
<ComboBox x:Name=«comboEtud»  ..... ItemsSource="{Binding}/>
….
<Label x:Name=“nomEtud" Content="{Binding
ElementName=comboEtud ,Path=SelectedItem.Nom}” />
```