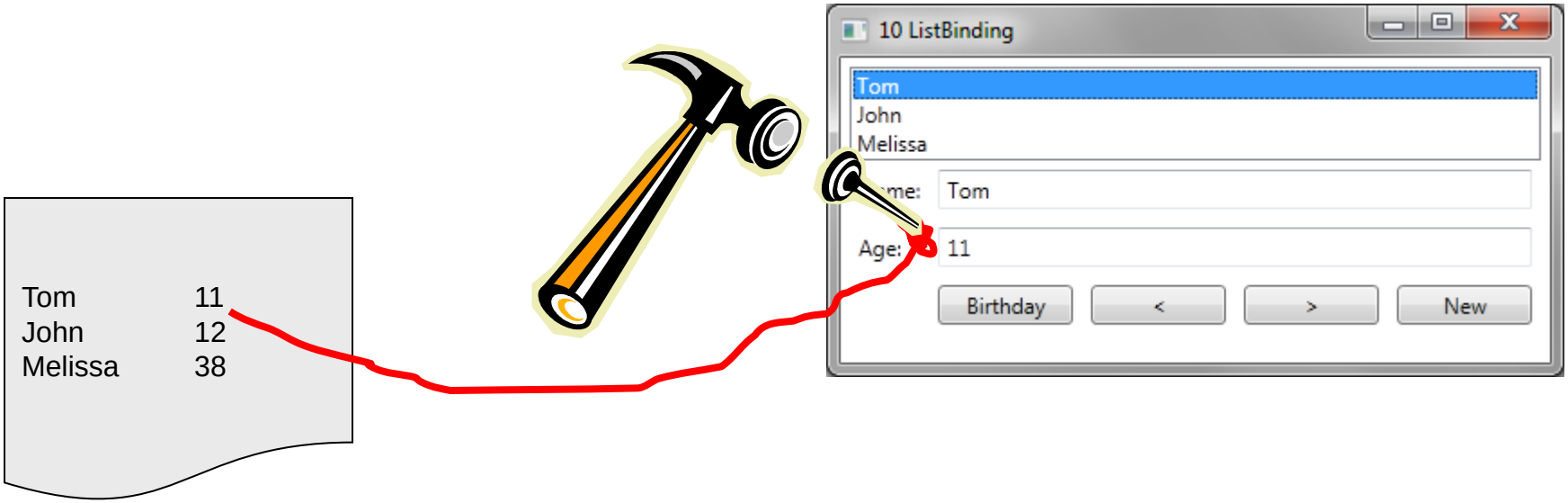
Data Bindings

Binding data to controls



Agenda

- Introduction
- What is data binding?
- Binding Example in C#
- Basic Data Binding Concepts
- Element Binding
- Binding to List Data

Data Principles

- Most applications are built to display or create some type of data.
- Whether that data is a document, database, or drawing, Windows applications are fundamentally about displaying, creating, and editing data.
- A data model describes the contract between a source of data and the consumer of that data (Controls).
- Because all of WPF's data operations are based on the fundamental .NET data model, WPF controls can retrieve data from any CLR object.
- In WPF a separation between data and UI is possible (and highly recommended) but not required.

DataBinding
Demo 00

A beginners Application

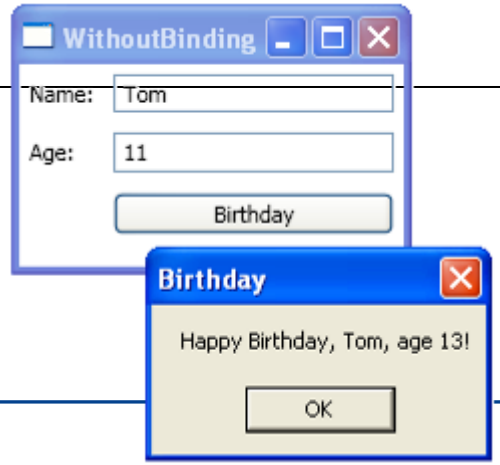
```
public class Person {
  string name;
  public string Name
  {
    get { return this.name; }
    set { this.name = value; }
  }

  int age;
  public int Age
  {
    get { return this.age; }
    set { this.age = value; }
  }

  public Person() { }
  public Person(string name, in
  {
    this.name = name;
    this.age = age;
  }
}
```

```
public partial class Window1 : Window {
  Person person = new Person("Tom", 11);
  public Window1() {
    InitializeComponent();
    // Fill initial person fields
    this.nameTextBox.Text = person.Name;
    this.ageTextBox.Text = person.Age.ToString();
    this.birthdayButton.Click += birthdayButton_Cli
  }

  void birthdayButton_Click(object s, RoutedEventAr
    ++person.Age;
    MessageBox.Show(string.Format(
                    "Happy Birthday, {0}, age {1}!"
                    person.Name,
                    person.Age),
                    "Birthday");
    }
  }
```



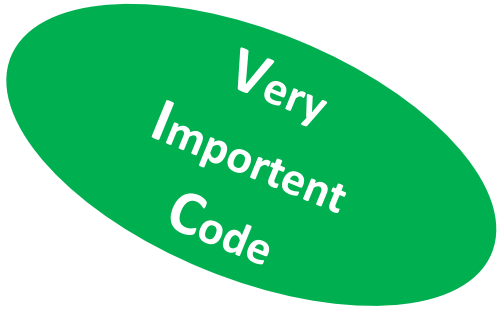
No synchronization of data in the model and in the GUI!

Object Changes

- A robust way for the UI to track object changes is for the object to raise an event when a property changes.
- The right way for an object to do this is with an implementation of the **`INotifyPropertyChanged`** interface.

Enhanced Application – Part 1

```
public class Person : INotifyPropertyChanged
{
   public event PropertyChangedEventHandler PropertyChanged;
   protected void Notify([CallerMemberName]string propName = null) {
       if( this.PropertyChanged != null ) {
          PropertyChanged(this, new PropertyChangedEventArgs(propName));
       }
   }
   string name;
   public string Name {
      get { return this.name; }
      set {
         if( this.name == value ) { return; }
         this.name = value;
         Notify("Name");}
   }
   int age;
   public int Age {
      get { return this.age; }
      set {
         if( this.age == value ) { return; }
         this.age = value;
         Notify();}
    }
    public Person() { }
}
```

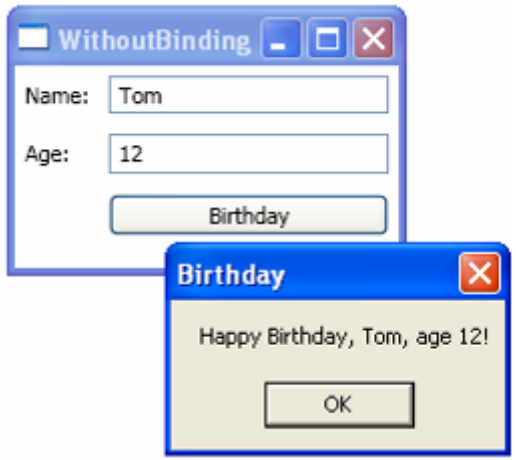


Enhanced Application – Part 2

```
public Window1() {
  InitializeComponent();
  // Fill initial person fields
  this.nameTextBox.Text = person.Name;
  this.ageTextBox.Text = person.Age.ToString();
  // Watch for changes in Tom's properties
  person.PropertyChanged += person_PropertyChanged;
  // Watch for changes in the controls
  this.nameTextBox.TextChanged += nameTextBox_TextChanged;
  this.ageTextBox.TextChanged += ageTextBox_TextChanged;
  // Handle the birthday button click event
  this.birthdayButton.Click += birthdayButton_Click;
}
void person_PropertyChanged(object sender, PropertyChangedEventArgs e) {
  switch( e.PropertyName ) {
    case "Name":
      this.nameTextBox.Text = person.Name;
      break;
    case "Age":
      this.ageTextBox.Text = person.Age.ToString();
      break;
  }
}
void nameTextBox_TextChanged(object sender, TextChangedEventArgs e) {
  person.Name = nameTextBox.Text;
}
```

Enhanced Application - Result

- It works
  - But the coding is very tedious!
  - And it doesn't scale very well.

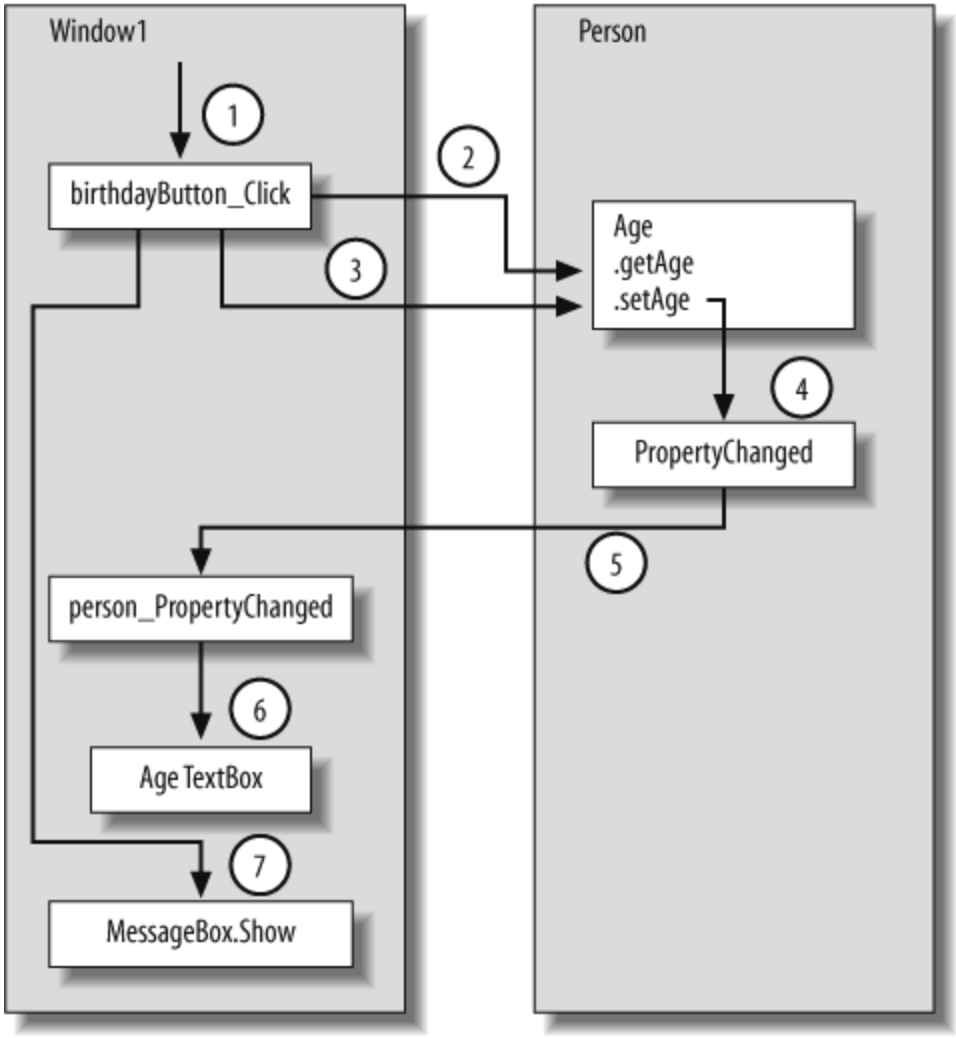



```
void birthdayButton_Click(object sender, RoutedEventArgs e)
      {
         ++person.Age;
         MessageBox.Show(
           string.Format(
             "Happy Birthday, {0}, age {1}!",
             person.Name,
             person.Age),
           "Birthday");
      }
```

DataBinding Demo 01

Keeping the UI up-to-date with changes in the object

1. User clicks on button, which causes Click event to be raised.
2. Click handler gets the age (11) from the Person object.
3. Click handler sets the age (12) on the Person object.
4. Person Age property setter raises the PropertyChanged event.
5. PropertyChanged event is routed to event handler in the UI code.
6. UI code updates the age TextBox from "11" to "12."
7. Button click event handler displays a message box showing the new age ("12").



WHAT IS DATA BINDING?

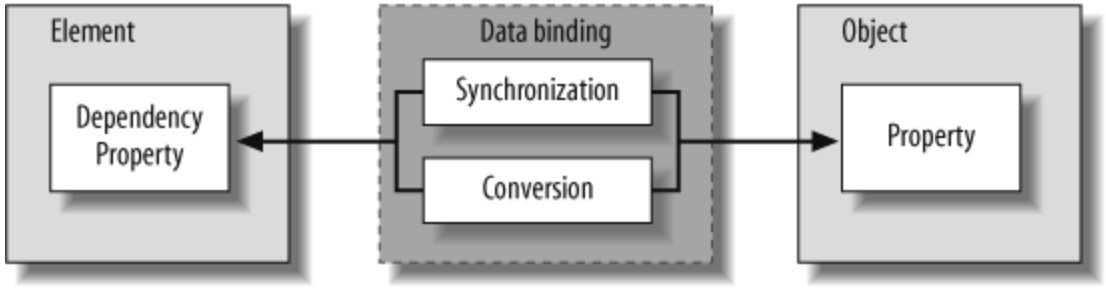
What is Data Binding?

Data binding is the association of object properties with control properties, facilitated by a data binding engine.

- When data changes the window must be updated
  - The changed data must be copied from the data object in the model to the control on the GUI.
- When data in the control is changed (by the user) the data object must be updated
  - The changed data must be copied from the control on the GUI to the data object in the model.

The Duties of Data Binding

- In WPF, **data binding** is the act of registering two properties with the data binding engine and letting the engine keep them synchronized, converting types as appropriate.



GUI — Model or GUI

Target ***Binding*** ***Source***

You can think of the Binding as the bridge between the Source and the Target

BINDING EXAMPLE IN C#

Data binding in C#

- In C# the **Binding** class is used to represent a data point to bind to (Data point is an abstract concept -> *the idea of a single "node" of data)*

- To construct a binding we provide:
  - a **source** (object or collection)
  - and **path** (typical a property name)

```
Binding bindName = new Binding();
bindName.Source = person;
bindName.Path = new PropertyPath("Name");
```

- In WPF one end of a binding (or both) must be to a dependency property.
  - We use the SetBinding method on a UI element to define the other end of the binding.

```
tbxName.SetBinding(TextBox.TextProperty, bindName);
```

Simple Bindings in C#

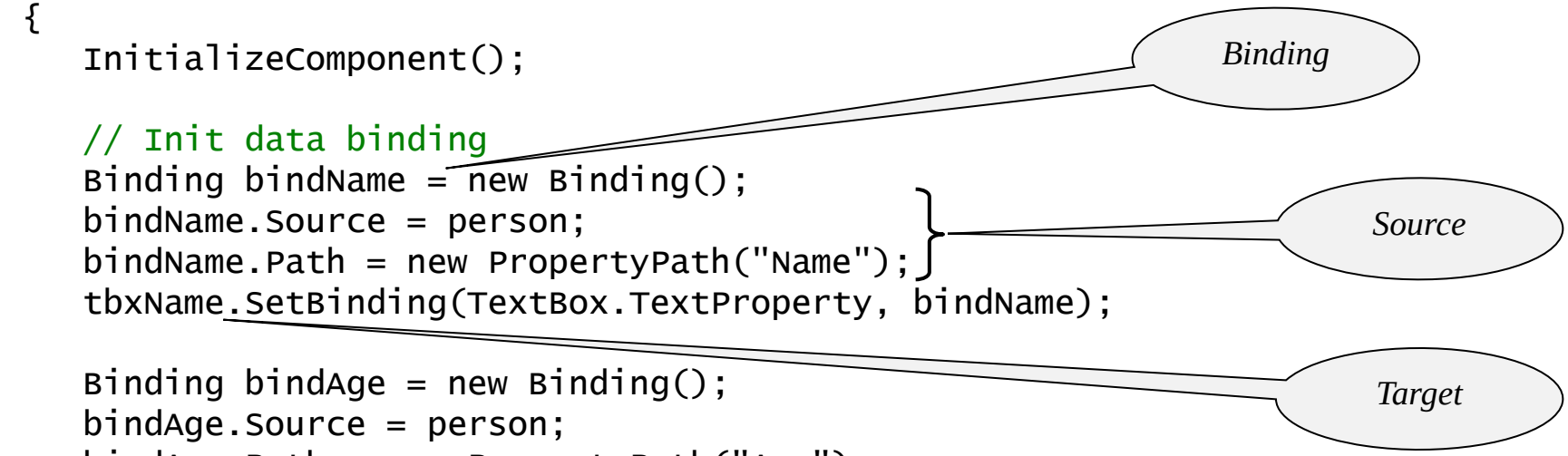
```
// Window1.xaml.cs
public partial class Window1 : Window
{
   Person person = new Person("Tom", 11);

   public Window1()
   {
      InitializeComponent();

      // Init data binding
      Binding bindName = new Binding();
      bindName.Source = person;
      bindName.Path = new PropertyPath("Name");
      tbxName.SetBinding(TextBox.TextProperty, bindName);

      Binding bindAge = new Binding();
      bindAge.Source = person;
      bindAge.Path = new PropertyPath("Age");
      tbxAge.SetBinding(TextBox.TextProperty, bindAge);

      this.birthdayButton.Click += birthdayButton_Click;
   }
```



DataBinding
Demo 02

BASIC DATA BINDING CONCEPTS

Elements of a Data Binding

- *Source*
  - The authoritative data, the *stuff* we want to display.
- *Target*
  - The object that will reflect the data in some manner, such as a control that displays values or changes color based on the values and so on.
- *Binding*
  - The rules around how the data will be reflected. Eg.:
    - Is the source read-only?
    - Does the source change?
    - When does it change?
    - How often does it change?
    - Is the source a list?
    - Is the target a list?

Binding Sources

There are four sources that WPF can bind to out of the box:

- *CLR objects*
  - Individual objects or collections.
- *ADO.NET data types*
  - DataTable, DataView, DataSet, and so on.
  - There's also direct support for binding to LINQ to SQL.
- *XML data*
  - Via XPath or
  - LINQ to XML.
- *DependencyObjects* (Element Binding)
  WPF objects that participate in the WPF property system.

The binding system has increasing support to bind to objects as they implement various interfaces.

Explicit Data Source in C#

- If you have a named source of data, you can be explicit about the source in the binding
  - instead of relying on implicitly binding to a DataContext property set somewhere in the tree.

- You set the source explicit by use of the Source property on the binding.

```
Binding bindName = new Binding();
bindName.Source = person;
bindName.Path = new PropertyPath("Name");
tbxName.SetBinding(TextBox.TextProperty, bindName);
```

- Being explicit is useful in some scenarios
  - E.g. if you've got more than one source of data.
  - **But often it's easier to use the DataContext**.

Explicit Static Data Source in XAML

- You can also set the Source property in XAML, but only to:
  - A resource
  - Or a static property or function
- The XAML code below shows binding to a static property in the App class.

```
<TextBox Grid.Row="0" Grid.Column="1" Margin="5"
   Text="{Binding Source={x:Static Application.Current},
                  Path=Person.Name}" />
```

```
public partial class App : Application
{
   public Person Person {get; set;}

   private void Application_Startup(object s, StartupEventArgs e)
   {
     Person = new Person("Tom", 12);
   }
}
```

Note:
You cannot bind directly to an object created in the code behind file (the C# file)!

DataBinding Demo 03

Explicit Resource Data Source in XAML

- The XAML code below shows binding to an object created as a resource in the Windows Resources collection.

Create an instance of the class Person and give that instance the key ”Tom”.

Note:
In the code behind file you must search for that object in the resources!

```
<Window x:Class="WithXAMLResource.MainWin
        xmlns="http://schemas.microsoft.com/...
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:WithXAMLResource"
        Title="Binding To XAML Resource" Height="150" Width="300">
    <Window.Resources>
        <local:Person x:Key="Tom" Name="Tom" Age="12" />
    </Window.Resources>
    <Grid>
    <TextBox Grid.Row="0" Grid.Column="1" Margin="5"
             Text="{Binding Source={StaticResource Tom},
                            Path=Name}"
             />
```

DataBinding Demo 04

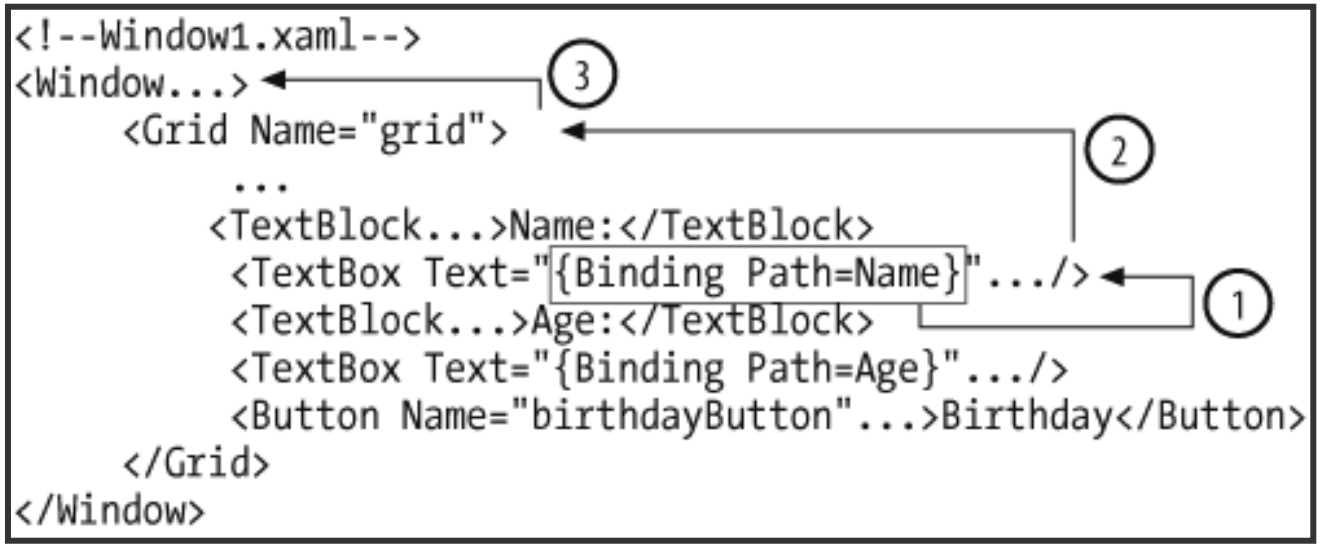
```
void birthdayButton_Click(object …)
{
    Person person = (Person)this.FindResource("Tom");
```

Implicit Data Source

- A **data context** is a place for bindings to look for the data source
  - if they don't have any other special instructions (Source not set).
- Every FrameworkElement and every FrameworkContentElement has a DataContext property.
- The DataContext property is of type Object
  - so we can plug anything we like into it (e.g., string, Person, List<Person>).
- When looking for an object to use as the binding source, the binding object logically traverses up the tree from where it's defined, looking for a DataContext property that has been set.

Searching the element tree for a non-null DataContext

1. The binding looks for a DataContext that has been set on the TextBox itself.
2. The binding looks for a DataContext that has been set on the Grid.
3. The binding looks for a DataContext that has been set on the Window.



This traversal is handy because it means that any two controls with a common logical parent can bind to the same data source.

Simple Bindings in XAML with DataContext

- Binding a UI target property to an implicit data source

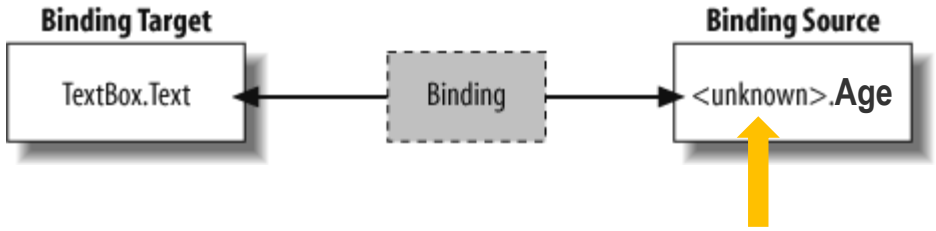
```
<TextBox ...>
  <TextBox.Text>
    <Binding Path="Age" />
  </TextBox.Text>
</TextBox>
```

- The shortcut binding syntax

```
<TextBox Text="{Binding Path=Age}" />
```

- The shortest cut binding syntax

```
<TextBox Text="{Binding Age}" />
```



Use the DataContext as Source

DataContext Set in C#

- It's often very convenient to set the DataContext in the code behind file.

```
public partial class MainWindow : Window
  {
    Person person = new Person("Tom", 12);

    public MainWindow()
    {
      InitializeComponent();
      DataContext = person;
      birthdayButton.Click += birthdayButton_Click;
    }
```

```
<TextBox Grid.Row="0" Grid.Column="1" Margin="5”
         Text="{Binding Path=Name}" />
```

DataBinding Demo 05

DataContext Set in XAML

- You can set the DataContext to a resource or a static property or a static function.

```
<Window x:Class="BindingByContextAndResource.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/p..."
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:BindingByContextAndResource"
        Title="Binding By Context " Height="150" Width="300">
    <Window.Resources>
        <local:Person x:Key="Tom" Name="Tom" Age="11" />
    </Window.Resources>
    <Grid  DataContext="{StaticResource Tom}">
     …
    <TextBox Grid.Row="0" Grid.Column="1" Margin="5"
             Text="{Binding Path=Name}" />
```

DataBinding Demo 06

Direction of the Data Flow

You can control this by setting the **Mode property** of your Binding object.

Eg.: `binding.Mode = BindingMode.TwoWay;`

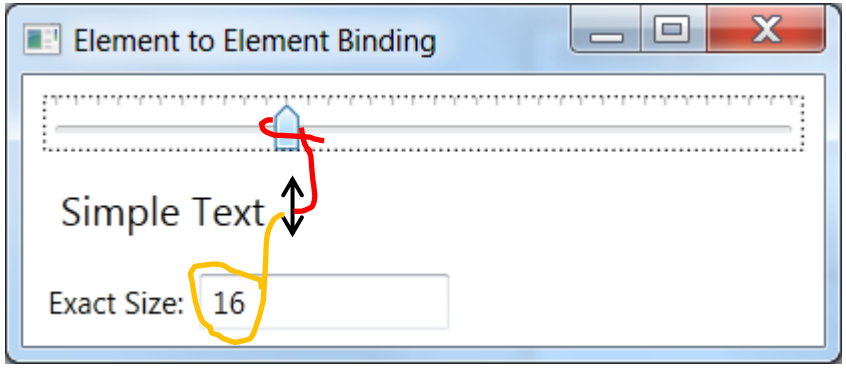
- `OneTime`
  - It copies the source data value to the target of the data binding operation once.
- `OneWay`
  - For the cases when you want to expose some underlying data in the UI, but the data is readonly or otherwise not meaningful to modify.
- `OneWayToSource`
  - Whenever the binding target changes, OneWayToSource copies the data to the source.
- `TwoWay`
  - The common business logic scenario in which you have some business data to load and reflect in the UI; when the data is changed in the UI, the changes are automatically propagated back to the business object.

Note that to detect source changes (applicable to OneWay and TwoWay bindings), the source must implement a suitable property change notification mechanism such as INotifyPropertyChanged.

ELEMENT BINDING

Element to Element Binding

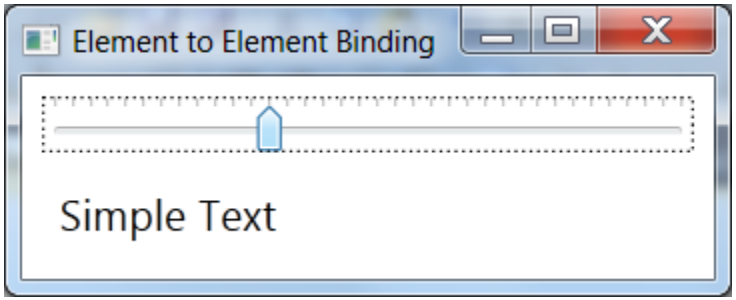
- The simplest data binding scenario occurs when your source object is a WPF element and your source property is a dependency property.
- When you change the value of the dependency property in the source object, the bound property in the target object is updated immediately.
  - E.g. the Height of the TextBlock is changed when the slider is moved.



Element to Element Binding XAML

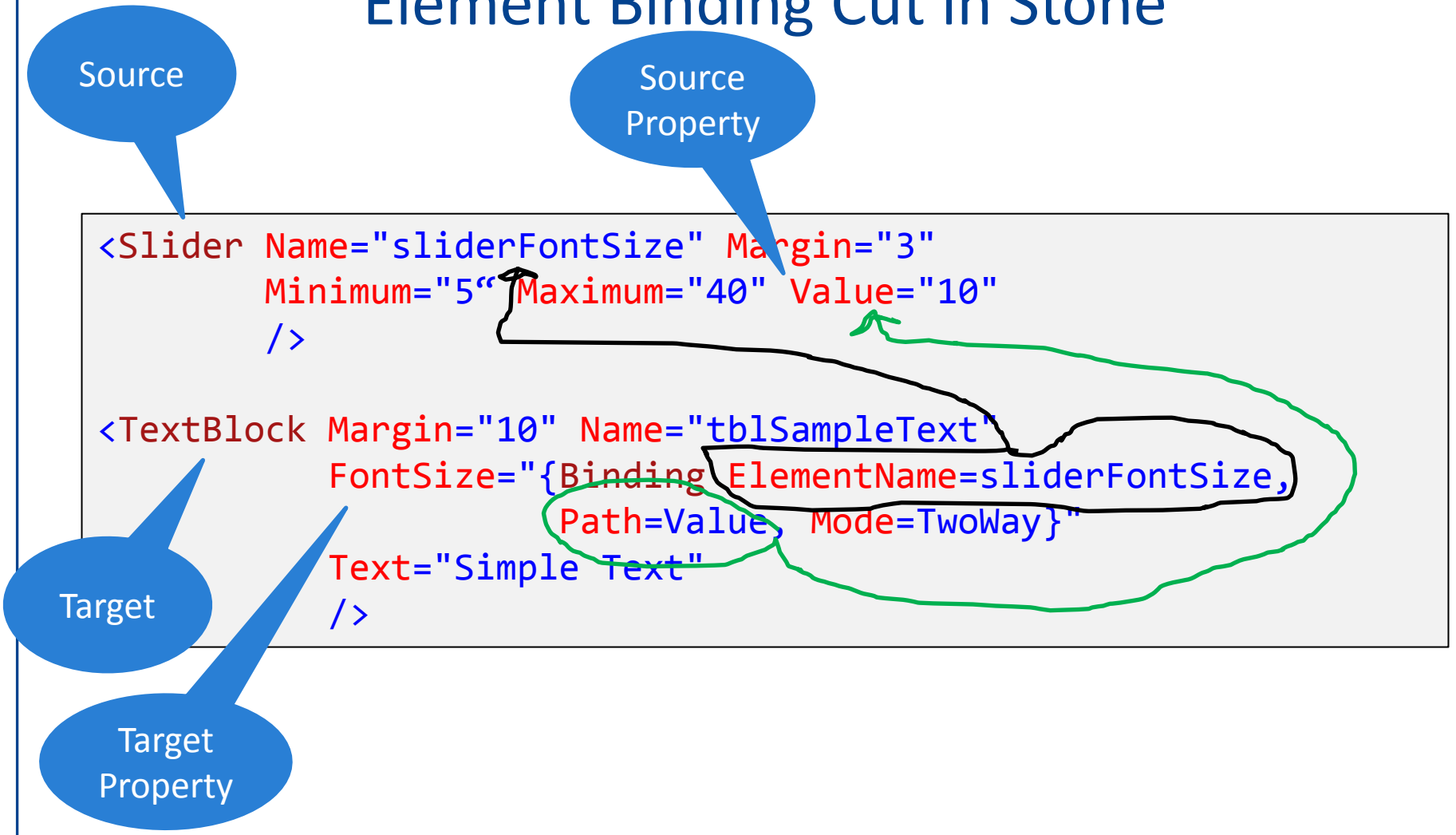
```
<Slider Name="sliderFontSize" Margin="3"
        Minimum="5“ Maximum="40" Value="10"
        />

<TextBlock Margin="10" Name="tblSampleText"
           FontSize="{Binding ElementName=sliderFontSize,
                      Path=Value, Mode=TwoWay}"
           Text="Simple Text"
           />
```



DataBinding Demo 07

Element Binding Cut in Stone



Binding to List Data

Collections of Data

- In many cases the data that you work with is a collection of objects.
  - A common scenario in data binding is to use an ItemsControl such as a ListBox, ListView, or Grid to display a collection of records.
- To set up dynamic bindings so that insertions or deletions in the collection update the UI automatically, the collection must implement the INotifyCollectionChanged interface.
  - This interface exposes the CollectionChanged event.
- WPF provides:
  - `ObservableCollection<T>`
    - which implements the INotifyCollectionChanged interface.

> *To fully support transferring data values from binding source objects to binding targets, each object in your collection that supports bindable properties must implement an appropriate property changed notification mechanism such as the INotifyPropertyChanged interface.*

Binding to List Data

- All Controls that follow the ItemsControls content model support binding to item sources through the ItemsSource Property:

```
ObservableCollection<string> listData;
listData = new ObservableCollection<string>();
listData.Add("Item 1");
listData.Add("Item 2");
ListBox listBox1 = new ListBox();

Binding binding1 = new Binding();
binding1.Source = listData;
ListBox1.SetBinding(ListBox.ItemsSourceProperty, binding1);
```

Or simply

```
ListBox1.ItemsSource = listData;
```

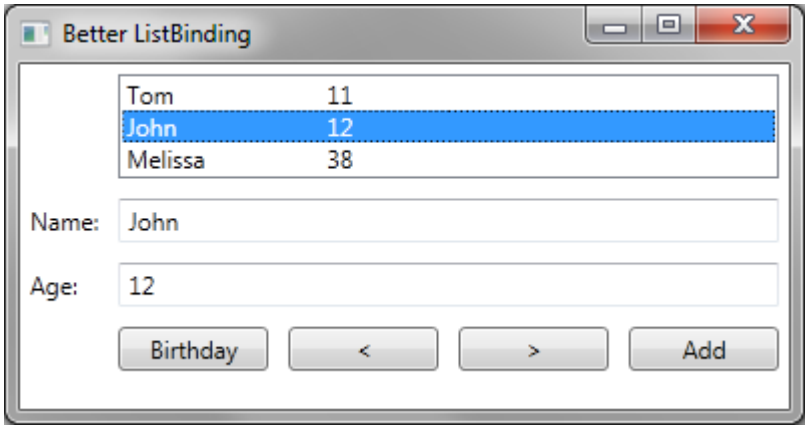
Or if more controls shall bind to the same data:

```
this.DataContext = listData;  // E.g. in the windows constructor
```

DataBinding Demo 10

Synchronisation of Controls

- The data binding engine has the concept of a current item
  - So even single item controls like a TextBox can be bound to a list data source.
- You can set the property IsSynchronizedWithCurrentItem to True on an ItemsControl e.g. a ListBox to automatically synchronize the bindings current element with the selected element in the list control.



DataBinding Demo 11

Debugging Bindings

- Debugging declarative code like XAML, can be frustrating.
  - When data binding works, it's lovely; but, when it doesn't, it can be an extremely frustrating experience.
- Always run your program in Debug mode (F5) and watch the output console window.
- But you can increase the amount of information written out via WPF's support for debugging bindings.
- To enable debug assistance, you need to add another reference to the XAML.
  - Add the following namespace on the Window element:
    ```
    xmlns:debug="clr-
    namespace:System.Diagnostics;assembly=WindowsBase"
    ```
- If you want to debug a particular binding, you can add a tracelevel to the statement:
  ItemsSource="{Binding Source={StaticResource processes},
  debug:PresentationTraceSources.TraceLevel=High}"
- Now when you run, a whole host of useful messages will be written to the output window, telling you step-by-step what's happening during the bind.

Binding Performance

- Under the hood, binding to CLR uses a lot of reflection
  - and wherever there's reflection, there are potential performance problems.
- In the simple case, the framework gets Type and Property descriptors on the CLR objects and sets up the binding appropriately.

Binding Performance

- In the case where performance is more critical, .NET and WPF provide the following interfaces to increase binding speed:
  - *ICustomTypeDescriptor*
    Provides a way for the binding code to find out about the object and its properties without using reflection.
  - *INotifyPropertyChanged*
    Provides an interface to implement a custom scheme for notifying the property system that the source data has been updated.
    WPF native DependencyProperties already provide this notification logic.

References

- Mainly the text book
  Pro WPF 4.5

- Data Binding in WPF Overview
  http://msdn.microsoft.com/en-us/library/ms752347.aspx

- Data Binding **How-to** Topics
  http://msdn.microsoft.com/en-us/library/ms752039.aspx