

# **ПРИВЯЗКА (BINDING)**

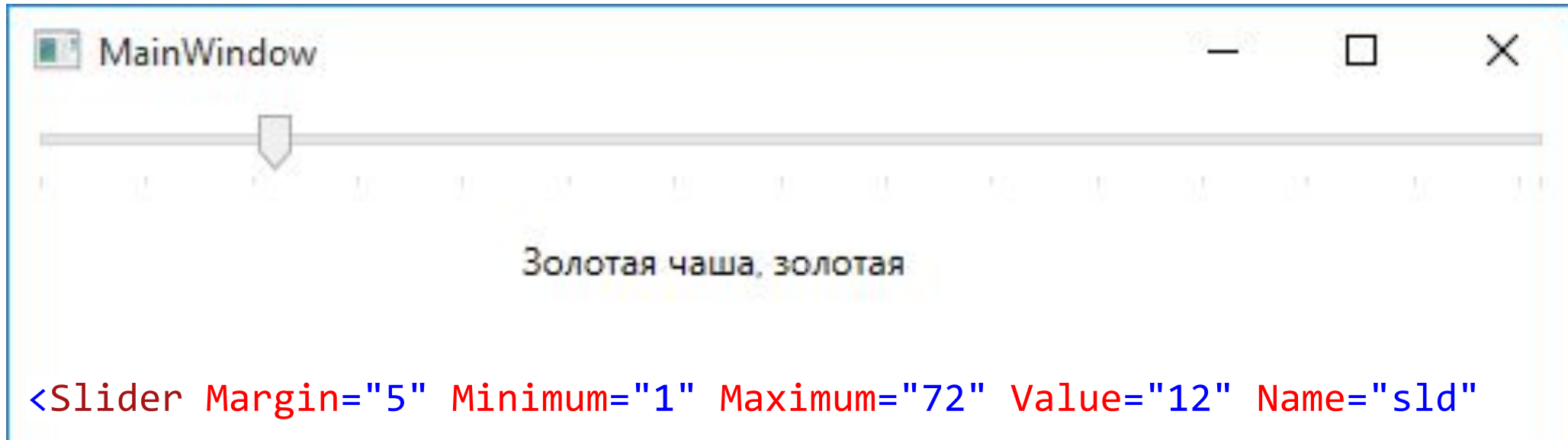
---

Основы и особенности применения

# Привязка. Введение

- Простейший сценарий привязки данных подразумевает ситуацию, когда исходным объектом является **элемент WPF**, а исходным свойством — **свойство зависимости**. Причина в том, что свойство зависимости имеет встроенную поддержку уведомлений об изменениях. В результате, когда значение свойства зависимости изменяется в исходном объекте, привязанное свойство целевого объекта немедленно обновляется. Это именно то, что требуется, и происходит оно без необходимости построения любой дополнительной инфраструктуры.

# Связывание элементов



```
<Slider Margin="5" Minimum="1" Maximum="72" Value="12" Name="sld"
        TickFrequency="5" TickPlacement="BottomRight"></Slider>
```

```
<TextBlock Margin="167,44,152,253" Text="Золотая чаша, золотая"
            FontSize="{Binding ElementName=sld, Path=Value}"></TextBlock>
```

# Ошибки привязки

- WPF не генерирует исключения для уведомления о проблемах привязки данных. Если указан несуществующий элемент или свойство, никакого сообщения об этом не будет; вместо этого данные просто не попадут в целевое свойство.
- WPF выводит трассировочную информацию, которая детализирует сбои в привязке. Во время отладки приложения эта информация появляется в выходном окне Visual Studio.

# Режимы привязки

```
<Slider Margin="5" Minimum="1" Maximum="72" Value="12" Name="sld"  
    TickFrequency="5" TickPlacement="BottomRight"></Slider>  
<TextBlock Margin="167,44,152,253" Text="Золотая чаша, золотая"  
    FontSize="{Binding ElementName=sld, Path=Value, Mode=TwoWay}"></TextBlock>
```

# Режимы привязки

- **OneWay** - Целевое свойство обновляется при изменениях исходного свойства
- **TwoWay** - Целевое свойство обновляется при изменениях исходного свойства, а исходное свойство обновляется при изменении целевого свойства
- **OneTime** - Целевое свойство устанавливается изначально на основе значения исходного свойства. Однако с этого момента изменения игнорируются (если только привязка не устанавливается на совершенно другой объект или не вызывается `BindingExpression.UpdateTarget()`). Обычно этот режим используется для сокращения накладных расходов, если известно, что целевое свойство не изменится
- **OneWayToSource** - Подобно `OneWay`, но действует в обратном направлении. Исходное свойство обновляется, когда изменяется целевое свойство (что может показаться несколько странным), но целевое свойство никогда не обновляется
- **Default** - Этот тип привязки зависит от целевого свойства. Это либо `TwoWay` (для устанавливаемых пользователем свойств, таких как `TextBox.Text`), либо `OneWay` (для всего остального). Все привязки используют данный подход, если только не указано иное

# Создание привязки в коде

```
Binding bind = new Binding();  
bind.Source = sld;  
bind.Path = new PropertyPath("Value");  
bind.Mode = BindingMode.TwoWay;  
txb.SetBinding(TextBlock.FontSizeProperty, bind);  
//удаляем конкретную привязку  
BindingOperations.ClearBinding(txb, TextBlock.FontSizeProperty);  
//удалить все привязки  
BindingOperations.ClearAllBindings(txb);
```

# Обновление привязок

Изменения, протекающие в обратном направлении — от цели к источнику — не обязательно происходят немедленно. Их поведение управляется свойством **Binding.UpdateSourceTrigger**, которое принимает одно из значений, описанных ниже

- ***PropertyChanged*** - Источник обновляется немедленно, когда изменяется целевое свойство
- ***LostFocus*** - Источник обновляется немедленно, когда изменяется целевое свойство и цель теряет фокус
- ***Explicit*** - Источник не обновляется, пока не будет вызван метод `BindingExpression.UpdateSource()`
- ***Default*** - Поведение обновления определяется метаданными целевого свойства (формально — его свойства `FrameworkPropertyMetadata.DefaultUpdateSourceTrigger`). Для большинства свойств поведением по умолчанию будет `PropertyChanged`, хотя свойство `TextBox.Text` обладает поведением по умолчанию `LostFocus`



# Обновление привязок

```
Binding bind = new Binding();  
bind.Source = sld;  
bind.Path = new PropertyPath("Value");  
bind.Mode = BindingMode.TwoWay;  
bind.UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged;  
txb.SetBinding(TextBlock.FontSizeProperty, bind);
```

# Привязка к объектам

Информация, которую необходимо отобразить, должны храниться в ***общедоступных свойствах***. Инфраструктура привязки данных WPF не может извлекать приватную информацию или читать общедоступные поля.

При привязке к объекту, не являющемуся элементом, следует отказаться от свойства `Binding.ElementName` и применять вместо него одно из следующих свойств:

# Привязка к объектам

- **Source** - Ссылка, указывающая на исходный объект; другими словами, это объект, поставляющий данные.
- **RelativeSource** - Указывает на исходный объект, использующий объект RelativeSource, который позволяет базировать ссылку на текущем элементе. Это специализированный инструмент, который удобен при написании шаблонов элементов управления и шаблонов данных.
- **DataContext** - Если источник не был указан с помощью свойства Source или RelativeSource, то среда WPF производит поиск в дереве элементов, начиная с текущего элемента. Она проверяет свойство DataContext каждого элемента и использует первый из них, который не равен null. Свойство DataContext исключительно полезно, когда нужно привязать несколько свойств одного объекта к разным элементам, потому что можно установить свойство DataContext высокоуровневого объекта контейнера, вместо его установки непосредственно на целевой элемент.

# Source

```
public MainWindow()  
{  
    InitializeComponent();  
    TestClass testClass = new TestClass();  
    Binding bind = new Binding();  
    bind.Source = testClass;  
    bind.Path = new PropertyPath("testText");  
    bind.Mode = BindingMode.TwoWay;  
    bind.UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged;  
    txb.SetBinding(TextBlock.TextProperty, bind);  
}
```

ССЫЛОК: 3

```
public class TestClass  
{  
    ССЫЛКА: 1  
    public TestClass() { }  
    ССЫЛОК: 0  
    public string testText { get; set; } = "Наполняет ароматом чая";  
}
```

# RelativeSource

```
<TextBlock Name="txb" Margin="10,5,0,10"  
    FontSize="{Binding ElementName=sld, Path=Value, Mode=TwoWay}">  
    <TextBlock.Text>  
        <Binding Path="Title">  
            <Binding.RelativeSource>  
                <RelativeSource Mode="FindAncestor"  
                    AncestorType="{x:Type Window}"></RelativeSource>  
            </Binding.RelativeSource>  
        </Binding>  
    </TextBlock.Text>  
</TextBlock>
```

# RelativeSource

```
<TextBlock Name="txb" Margin="10,5,0,10"  
    FontSize="{Binding ElementName=sld, Path=Value, Mode=TwoWay}"  
    Text="{Binding Path=Title, RelativeSource={RelativeSource Mode=FindAncestor,  
    AncestorType={x>Type Window}}}">  
</TextBlock>
```



# RelativeSource. Режимы

- **Self** - Выражение привязывается к другому свойству того же элемента
- **FindAncestor** - Выражение привязывается к родительскому элементу. WPF будет проводить поиск вверх по дереву элементов, пока не найдет нужный родительский элемент. Чтобы указать родителя, необходимо также установить свойство `AncestorType` для индикации типа родительского элемента, который должен быть найден. Дополнительно с помощью свойства `AncestorLevel` можно пропустить определенное количество совпадений указанного элемента. Например, если требуется привязка к третьему элементу типа `ListBoxItem` при восхождении вверх по дереву, то следует установить `AncestorType={x:Type ListBoxItem}` и `AncestorLevel=3`, тем самым пропуская первые два `ListBoxItem`. По умолчанию `AncestorLevel` равен 1, и поиск прекращается на первом найденном элементе.

# RelativeSource. Режимы

- ***PreviousData*** - Выражение осуществляет привязку к предыдущему элементу данных в списке, привязанном к данным. Это можно использовать в элементе списка
- ***TemplatedParent*** - Выражение осуществляет привязку к элементу, к которому применен шаблон. Этот режим работает, только если привязка находится внутри шаблона элемента управления или шаблона данных



# DataContext

```
Binding bind = new Binding();
```

```
bind.Source = План;
```

```
bind.Path = new PropertyPath("Element[" + r.Key +  
"].Elements[Раздел][0]");
```

```
tab.SetBinding(TabItem.DataContextProperty, bind);
```

# DataContext

```
b = new Binding();  
b.Mode = BindingMode.TwoWay;  
b.Path = new PropertyPath("Element[" + p.Key + "].Value");  
b.UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged;  
tb.SetBinding(TextBox.TextProperty, b);
```