Урок 5

[1. DataBinding 1](#_Toc486861893)

[1.1. Основные понятия связывания данных 1](#_Toc486861894)

[1.2. Синтаксис связывания данных 3](#_Toc486861895)

[1.3. Форматирование значений привязки и конвертеры значений 3](#_Toc486861896)

[1.4. Использование DataContext 5](#_Toc486861897)

[1.5. Обновление привязки. UpdateSourceTrigger. 6](#_Toc486861898)

[1.6. Реакция на изменения 8](#_Toc486861899)

[1.7. Отладка связывания данных 9](#_Toc486861900)

[2. Триггеры 12](#_Toc486861901)

[2.1. Триггеры свойств 12](#_Toc486861902)

[2.2. Триггеры данных 13](#_Toc486861903)

[2.3. Триггеры событий 13](#_Toc486861904)

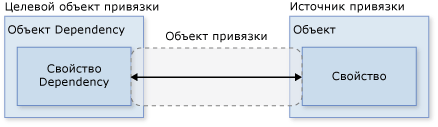
[3. Примечание 14](#_Toc486861905)

1. DataBinding

Привязка данных Windows Presentation Foundation (WPF) предоставляет приложениям простой и последовательный способ представления и взаимодействия с данными. Элементы можно связать с данными из разнообразных источников данных в форме объектов среды CLR и XML.

* 1. Основные понятия связывания данных

Вне зависимости от того, какие элементы связываются и какой источник данных используется, каждая привязка всегда соответствует следующей модели:



Как показано в приведенном выше рисунке, связывание данных является по существу мостом между [целью привязки](https://msdn.microsoft.com/ru-ru/library/gtmt#binding_target) и [источником привязки](https://msdn.microsoft.com/ru-ru/library/gtmt#binding_source).

Привязка данных представляет собой взаимодействие двух объектов: источника и целевого объекта. Целевой объект создает привязку к определенному свойству объекта-источника. При изменении объекта-источника, целевой объект также будет модифицирован.

Каждая привязка имеет четыре компонента: [целевой объект привязки](https://msdn.microsoft.com/ru-ru/library/gtmt#binding_target), свойство целевого объекта, [источник привязки](https://msdn.microsoft.com/ru-ru/library/gtmt#binding_source) и путь к значению используемого [источника привязки](https://msdn.microsoft.com/ru-ru/library/gtmt#binding_source). Например, если требуется связать содержимое [TextBox](https://msdn.microsoft.com/ru-ru/library/system.windows.controls.textbox%28v=vs.110%29.aspx) со свойством «Имя» объекта «Сотрудник», целевым объектом является [TextBox](https://msdn.microsoft.com/ru-ru/library/system.windows.controls.textbox%28v=vs.110%29.aspx), свойством целевого объекта является свойство [Text](https://msdn.microsoft.com/ru-ru/library/system.windows.controls.textbox.text%28v=vs.110%29.aspx), используемым значением является «Имя», а объектом источника является объект «Сотрудник».

Пример, демонстрирующий связывание значения TextBlock со свойством Text у TextBox.

TextBlock автоматически изменяет свое значение при вводе текста в TextBox. Без возможностей data binding потребовалось бы реализовывать обработчик событий для событий TextBox и затем изменять значение TextBlock при каждом изменении вводимого текста.

MainWindow.xaml

<Window x:Class="BindingSample.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:BindingSample"

mc:Ignorable="d"

Title="MainWindow" Height="200" Width="200">

<StackPanel Margin="10">

<TextBox Name="txtValue" />

<WrapPanel Margin="0,10">

<TextBlock Text="Текст: " FontWeight="Bold" />

<TextBlock Text="{Binding ElementName=txtValue, Path=Text }" />

</WrapPanel>

</StackPanel>

</Window>

Аналогично, можно установить привязку в файле отделенного кода:

MainWindow.xaml

<Window x:Class="BindingSample.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:BindingSample"

mc:Ignorable="d"

Title="MainWindow" Height="200" Width="200">

<StackPanel Margin="10">

<TextBox Name="txtValue" />

<WrapPanel Margin="0,10">

<TextBlock Text="Текст: " FontWeight="Bold" />

<TextBlock x:Name="mirrorTextBlock"/>

</WrapPanel>

</StackPanel>

</Window>

MainWindow.xaml.cs

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

namespace BindingSample

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

Binding binding = new Binding();

binding.ElementName = "txtValue"; // элемент-источник

binding.Path = new PropertyPath("Text"); // свойство элемента-источника

mirrorTextBlock.SetBinding(TextBlock.TextProperty, binding); // установка привязки для элемента-приемника

}

}

}

* 1. Синтаксис связывания данных

Для связывания данных используется Binding расширение XAML, которое позволяет описывать взаимодействие целевого объекта и источника.

Для определения привязки используется выражение типа:

{Binding «ElementName»= «Имя\_объекта-источника», Path=«Свойство\_объекта-источника», Mode=«Mode»}

Свойство Mode объекта Binding, которое представляет режим привязки, может принимать следующие значения:

* OneWay: свойство целевого объекта изменяется после модификации свойства объекта-источника.
* OneTime: свойство целевого объекта устанавливается по свойству объекта-источника только один раз. В дальнейшем изменения в источнике никак не влияют на объект-приемник.
* TwoWay: целевой объект и источник могут изменять привязанные свойства друг друга.
* OneWayToSource: целевой объект в котором объявлена привязка, меняет объект-источник.
* Default: по умолчанию (если меняется свойство TextBox.Text, то имеет значение TwoWay, в остальных случаях OneWay).
  1. Форматирование значений привязки и конвертеры значений

Часто возникают ситуации, когда необходимо изменить вид привязанных значений при их отображении в интерфейсе. Например, по-разному отображать положительные и отрицательные значения или отобразить размер файла в байтах, килобайтах и мегабайтах в зависимости от его размера.

Для таких случаев существует форматирование значений и конвертеры значений.

Пример форматирования значений:

MainWindow.xaml

<Window x:Class="StringFormat.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:StringFormat"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Window.Resources>

<local:Employee x:Key="Employee" Name="Петя" Age="30" Salary="25000" />

</Window.Resources>

<Grid>

<TextBlock Text="{Binding StringFormat=Зарплата составляет {0} рублей, Source={StaticResource Employee}, Path=Salary}" />

</Grid>

</Window>

MainWindow.xaml.cs

using System.Windows;

namespace StringFormat

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

}

public class Employee

{

public string Name { get; set; }

public string Age { get; set; }

public int Salary { get; set; }

}

}

Конвертеры значений (value converter) позволяют преобразовать значение из источника привязки к типу, который понятен целевому объекту привязки. Так как не всегда два связываемых привязкой свойства могут иметь совместимые типы. И в этом случае как раз и нужен конвертер значений.

Пример конвертера значений, который преобразовывает строковые значения к типу Boolean и наоборот:

MainWindow.xaml

<Window x:Class="CheckBoxValueConverter.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:CheckBoxValueConverter"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Window.Resources>

<local:YesNoToBooleanConverter x:Key="YesNoToBooleanConverter" />

</Window.Resources>

<StackPanel Margin="10">

<TextBox Name="txtValue" />

<WrapPanel Margin="0,10">

<TextBlock Text="Текущее значение: " />

<TextBlock Text="{Binding ElementName=txtValue, Path=Text, Converter={StaticResource YesNoToBooleanConverter}}"></TextBlock>

</WrapPanel>

<CheckBox IsChecked="{Binding ElementName=txtValue, Path=Text, Converter={StaticResource YesNoToBooleanConverter}}" Content="Yes" />

</StackPanel>

</Window>

MainWindow.xaml.cs

using System;

using System.Windows;

using System.Windows.Data;

namespace CheckBoxValueConverter

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

}

public class YesNoToBooleanConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

switch(value.ToString().ToLower())

{

case "yes":

case "true":

return true;

case "no":

case "false":

return false;

}

return false;

}

public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

if(value is bool)

{

if((bool)value == true)

return "yes";

else

return "no";

}

return "no";

}

}

}

Метод конверт позволяет преобразовать строку к типу данных boolean, метод ConvertBack выполняет обратное преобразование.

* 1. Использование DataContext

По-умолчанию, свойство DataContext является объектом-источником для привязки, если не определен иной объект-источник. Это свойство определено в классе FrameworkElement, от которого наследуется большинство элементов интерфейса, включая Window.

При запуске приложения значение DataContext не определено, но т.к. свойство DataContext присутствует в классе Window, то возможно присвоить значение DataContext в классе Window и использовать его, в дальнейшем, в других элементах, входящих в Window.

MainWindow.xaml

<Window x:Class="DataContextWPF.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:DataContextWPF"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<StackPanel Margin="15">

<WrapPanel>

<TextBlock Text="Заголовок: " />

<TextBox Text="{Binding Title, UpdateSourceTrigger=PropertyChanged}" Width="150" />

</WrapPanel>

<WrapPanel Margin="0,10,0,0">

<TextBlock Text="Размеры: " />

<TextBox Text="{Binding Width}" Width="50" />

<TextBlock Text=" x " />

<TextBox Text="{Binding Height}" Width="50" />

</WrapPanel>

</StackPanel>

</Window>

MainWindow.xaml.cs

using System.Windows;

namespace DataContextWPF

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

this.DataContext = this;

}

}

}

В файле отделенного кода происходит присвоение this.DataContext = this, т.е. сам объект выступает в качестве DataContext.

Использование DataContext позволяет избежать необходимости указывать множество объектов-источников для связывания.

* 1. Обновление привязки. UpdateSourceTrigger.

Односторонняя привязка от источника к приемнику практически мгновенно изменяет свойство приемника. Но при использовании двусторонней привязки в случае с текстовыми полями (как в примере выше), то при изменении целевого объекта свойство источника не изменяется мгновенно. Так, в примере выше, чтобы текстовое поле-источник изменилось, нужно перевести фокус с текстового поля целевого объекта. Как будет происходить обновление определяет свойство UpdateSourceTrigger класса Binding. Это свойство в качестве значения принимает одно из значений перечисления UpdateSourceTrigger:

* PropertyChanged: источник привязки обновляется сразу после обновления свойства в целевом объекте;
* LostFocus: источник привязки обновляется только после потери фокуса целевым объектом;
* Explicit: источник не обновляется до тех пор, пока не будет вызван метод BindingExpression.UpdateSource()
* Default: значение по умолчанию. Для большинства свойств это значение PropertyChanged. А для свойства Text элемента TextBox это значение LostFocus.

Изменим предыдущий пример так, чтобы заголовок формы изменялся явно, по нажатию кнопки, ширина формы изменялась при потере фокуса, а высота – сразу после изменения значения.

MainWindow.xaml

<Window x:Class="UpdateSource.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:UpdateSource"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<StackPanel Margin="15">

<WrapPanel>

<TextBlock Text="Заголовок: " />

<TextBox Name="txtWindowTitle" Text="{Binding Title, UpdateSourceTrigger=Explicit}" Width="150" />

<Button Name="btnUpdateSource" Click="btnUpdateSource\_Click" Margin="5,0" Padding="5,0">Изменить</Button>

</WrapPanel>

<WrapPanel Margin="0,10,0,0">

<TextBlock Text="Размеры: " />

<TextBox Text="{Binding Width, UpdateSourceTrigger=LostFocus}" Width="50" />

<TextBlock Text=" x " />

<TextBox Text="{Binding Height, UpdateSourceTrigger=PropertyChanged}" Width="50" />

</WrapPanel>

</StackPanel>

</Window>

MainWindow.xaml.cs

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

namespace UpdateSource

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

this.DataContext = this;

}

private void btnUpdateSource\_Click(object sender, RoutedEventArgs e)

{

BindingExpression binding = txtWindowTitle.GetBindingExpression(TextBox.TextProperty);

binding.UpdateSource();

}

}

}

* 1. Реакция на изменения

Существует два сценария обработки изменений в источнике данных: внести изменения в интерфейс приложения и внести изменения в связанные данные.

В WPF этой цели служат ObservableCollection и интерфейс INotifyPropertyChanged.

[INotifyPropertyChanged](http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged.aspx) используется для уведомления представления об изменениях свойств объекта. Реализация классом интерфейса предполагает генерацию события [PropertyChanged](http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged.propertychanged.aspx) каждый раз, когда значение свойства объекта изменяется. Такое поведение позволяет привязкам данных отслеживать состояние объекта и обновлять данные пользовательского интерфейса при изменении значения связанного свойства.

MainWindow.xaml

<Window x:Class="Respond.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Respond"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<DockPanel Margin="10">

<StackPanel DockPanel.Dock="Right" Margin="10,0,0,0">

<Button Name="btnAddUser" Click="btnAddUser\_Click">Добавить</Button>

<Button Name="btnChangeUser" Click="btnChangeUser\_Click" Margin="0,5">Изменить</Button>

<Button Name="btnDeleteUser" Click="btnDeleteUser\_Click">Удалить</Button>

</StackPanel>

<ListBox Name="lbUsers" DisplayMemberPath="Name"></ListBox>

</DockPanel>

</Window>

MainWindow.xaml.cs

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Windows;

namespace Respond

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

private ObservableCollection<User> users = new ObservableCollection<User>();

public MainWindow()

{

InitializeComponent();

users.Add(new User() { Name = "Петя" });

users.Add(new User() { Name = "Коля" });

lbUsers.ItemsSource = users;

}

private void btnAddUser\_Click(object sender, RoutedEventArgs e)

{

users.Add(new User() { Name = "Вася" });

}

private void btnChangeUser\_Click(object sender, RoutedEventArgs e)

{

if (lbUsers.SelectedItem != null)

(lbUsers.SelectedItem as User).Name = "Иван";

}

private void btnDeleteUser\_Click(object sender, RoutedEventArgs e)

{

if (lbUsers.SelectedItem != null)

users.Remove(lbUsers.SelectedItem as User);

}

}

public class User : INotifyPropertyChanged

{

private string name;

public string Name

{

get { return this.name; }

set

{

if (this.name != value)

{

this.name = value;

this.NotifyPropertyChanged("Name");

}

}

}

public event PropertyChangedEventHandler PropertyChanged;

public void NotifyPropertyChanged(string propName)

{

if (this.PropertyChanged != null)

this.PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

}

* 1. Отладка связывания данных

Ошибки в связывании данных довольно сложно выявить, поскольку связывание данных выполняется во время исполнения программы и не вызывает исключений. Ошибки в связывании могут иметь разные причины, но наиболее распространена попытка связывания с несуществующим свойством.

MainWindow.xaml

<Window x:Class="BindingDebug.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:BindingDebug"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid Margin="10" Name="pnlMain">

<TextBlock Text="{Binding NonExistingProperty, ElementName=pnlMain}" />

</Grid>

</Window>

В первую очередь ошибку связывания можно обнаружить в VisualStudio в окне Output.

Output:

System.Windows.Data Error: 40 : BindingExpression path error: 'NonExistingProperty' property not found on 'object' ''Grid' (Name='pnlMain')'. BindingExpression:Path=NonExistingProperty; DataItem='Grid' (Name='pnlMain'); target element is 'TextBlock' (Name=''); target property is 'Text' (type 'String')

Это сообщение говорит, что была попытка использовать свойство NonExistingProperty объекта pnlMain типа Grid.

Еще один пример ошибки связывания:

MainWindow.xaml

<Window x:Class="TraceLevel.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TraceLevel"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid Margin="10">

<TextBlock Text="{Binding Title}" />

</Grid>

</Window>

В данном примере происходит связывание со свойством Title, но не указано к какому объекту оно относится. WPF будет пытаться использовать DataContext для получения значения, но в данном примере DataContext не инициализирован. В результате WPF не обнаружит никакой ошибки и окно Output не будет содержать сообщений об ошибке.

Если данное поведение программы не совпадает с ожиданиями разработчика, то необходимо повысить TraceLevel у объекта PresentationTraceSources из пространства имен System.Diagnostics.

В результате в окне Output будут отображаться все подробности выполнения связывания данных:

System.Windows.Data Warning: 67 : BindingExpression (hash=51220585): Resolving source

System.Windows.Data Warning: 70 : BindingExpression (hash=51220585): Found data context element: TextBlock (hash=28990061) (OK)

System.Windows.Data Warning: 71 : BindingExpression (hash=51220585): DataContext is null

System.Windows.Data Warning: 67 : BindingExpression (hash=51220585): Resolving source

System.Windows.Data Warning: 70 : BindingExpression (hash=51220585): Found data context element: TextBlock (hash=28990061) (OK)

System.Windows.Data Warning: 71 : BindingExpression (hash=51220585): DataContext is null

System.Windows.Data Warning: 67 : BindingExpression (hash=51220585): Resolving source (last chance)

System.Windows.Data Warning: 70 : BindingExpression (hash=51220585): Found data context element: TextBlock (hash=28990061) (OK)

System.Windows.Data Warning: 78 : BindingExpression (hash=51220585): Activate with root item <null>

System.Windows.Data Warning: 106 : BindingExpression (hash=51220585): Item at level 0 is null - no accessor

'TraceLevel.vshost.exe' (CLR v4.0.30319: TraceLevel.vshost.exe): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC\_MSIL\PresentationFramework-SystemXml\v4.0\_4.0.0.0\_\_b77a5c561934e089\PresentationFramework-SystemXml.dll'. Skipped loading symbols. Module is optimized and the debugger option 'Just My Code' is enabled.

System.Windows.Data Warning: 80 : BindingExpression (hash=51220585): TransferValue - got raw value {DependencyProperty.UnsetValue}

System.Windows.Data Warning: 88 : BindingExpression (hash=51220585): TransferValue - using fallback/default value ''

System.Windows.Data Warning: 89 : BindingExpression (hash=51220585): TransferValue - using final value ''

Просматривая текст в окне Output можно увидеть все действия, выполняющиеся в попытке найти подходящее значение для TextBlock: несколько сообщений DataContext is null и в конце сообщение, что будет использовано значение по умолчанию, пустая строка.

Другой способ обнаружения ошибок связывания состоит в использовании отладчика и файла отделенного кода. В файле отделенного кода необходимо реализовать «искусственный» конвертер данных, цель которого состоит в том, чтобы диагностировать

успешность связывания данных.

MainWindow.xaml

<Window x:Class="ValueConverter.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ValueConverter"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525" Name="wnd">

<Window.Resources>

<local:DebugDummyConverter x:Key="DebugDummyConverter" />

</Window.Resources>

<Grid Margin="10">

<TextBlock Text="{Binding Title, ElementName=wnd, Converter={StaticResource DebugDummyConverter}}" />

</Grid>

</Window>

MainWindow.xaml.cs

using System;

using System.Diagnostics;

using System.Windows;

using System.Windows.Data;

namespace ValueConverter

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

}

public class DebugDummyConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

Debugger.Break();

return value;

}

public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

Debugger.Break();

return value;

}

}

}

Если отладчик не остановиться в методах Convert или ConvertBack, это будет означать, что конвертер не используется. Это, обычно, говорит об ошибке связывания данных.

1. Триггеры

С помощью стилей происходит присваивание статических значений свойствам объектов. Триггеры, в свою очередь, позволяют изменять значения определенных свойств в зависимости от заданных условий.

Существует 4 категории триггеров:

* Триггеры свойств: вызываются в ответ на изменения свойствами зависимостей своего значения.
* Триггеры данных: вызываются в ответ на изменения значений любых свойств (они необязательно должны быть свойствами зависимостей).
* Триггеры событий: вызываются в ответ на генерацию событий.
* Мультитриггеры: вызываются при выполнении ряда условий.
  1. Триггеры свойств

Триггеры свойств задаются с помощью объекта Trigger. Они следят за значениями свойств и когда эти значения оказываются равными заданным величинам, свойства могут быть изменены. Изменения свойств осуществляются с помощью объекта Setter.

MainWindow.xaml

<Window x:Class="WpfTrigger.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfTrigger"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid>

<TextBlock Text="Триггер" FontSize="28" HorizontalAlignment="Center" VerticalAlignment="Center">

<TextBlock.Style>

<Style TargetType="TextBlock">

<Setter Property="Foreground" Value="Blue"></Setter>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Foreground" Value="Red" />

<Setter Property="TextDecorations" Value="Underline" />

</Trigger>

</Style.Triggers>

</Style>

</TextBlock.Style>

</TextBlock>

</Grid>

В данном примере, изначально, с помощью стиля, устанавливается синий цвет текста. С помощью триггера происходит анализ свойства IsMouseOver и, когда значение свойства становиться равным true, происходит изменение цвета текста и добавляется оформление текста в виде подчеркивания.

* 1. Триггеры данных

DataTrigger отслеживает изменение свойств, которые необязательно должны представлять свойства зависимостей. Для соединения с отслеживаемыми свойствами триггеры данных

используют выражения привязки.

MainWindow.xaml

<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">

<CheckBox Name="cbSample" Content="Вы знаете C#?" />

<TextBlock HorizontalAlignment="Center" Margin="0,20,0,0" FontSize="48">

<TextBlock.Style>

<Style TargetType="TextBlock">

<Setter Property="Text" Value="Нет" />

<Setter Property="Foreground" Value="Red" />

<Style.Triggers>

<DataTrigger Binding="{Binding ElementName=cbSample, Path=IsChecked}" Value="True">

<Setter Property="Text" Value="Да" />

<Setter Property="Foreground" Value="Green" />

</DataTrigger>

</Style.Triggers>

</Style>

</TextBlock.Style>

</TextBlock>

* 1. Триггеры событий

Триггеры событий, <EventTrigger>, чаще всего используются для определения анимации. Триггер событий реагирует на определенные события также как обработчик событий.

MainWindow.xaml

<Window x:Class="EventTrigger.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:EventTrigger"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid>

<TextBlock Name="lbl" Text="EventTrigger" FontSize="18" HorizontalAlignment="Center" VerticalAlignment="Center">

<TextBlock.Style>

<Style TargetType="TextBlock">

<Style.Triggers>

<EventTrigger RoutedEvent="MouseEnter">

<EventTrigger.Actions>

<BeginStoryboard>

<Storyboard>

<DoubleAnimation Duration="0:0:0.300" Storyboard.TargetProperty="FontSize" To="28" />

</Storyboard>

</BeginStoryboard>

</EventTrigger.Actions>

</EventTrigger>

<EventTrigger RoutedEvent="MouseLeave">

<EventTrigger.Actions>

<BeginStoryboard>

<Storyboard>

<DoubleAnimation Duration="0:0:0.800" Storyboard.TargetProperty="FontSize" To="18" />

</Storyboard>

</BeginStoryboard>

</EventTrigger.Actions>

</EventTrigger>

</Style.Triggers>

</Style>

</TextBlock.Style>

</TextBlock>

</Grid>

</Window>

Триггер событий подписан на два события MouseEnter и MouseLeave. Если курсор мыши оказывается над областью занятой текстом, то происходит постепенное увеличение размера символов до 28 пикселей. Когда курсор мыши покидает область текста, то происходит плавное уменьшение размеров символов до 18.

1. Примечание

<https://msdn.microsoft.com/ru-ru/library/ms752347(v=vs.110).aspx>

<http://www.wpf-tutorial.com/about-wpf/wpf-vs-winforms/>

<https://professorweb.ru/my/WPF/base_WPF/level1/1_1.php>

<https://metanit.com/sharp/wpf/1.php>

<http://www.c-sharpcorner.com/uploadfile/mahesh/listview-in-wpf/>