Урок 5

[1. DataBinding 1](#_Toc486795639)

[1.1. Основные понятия связывания данных 1](#_Toc486795640)

[1.2. Синтаксис связывания данных 3](#_Toc486795641)

[1.3. Использование DataContext 3](#_Toc486795642)

[1.4. Обновление привязки. UpdateSourceTrigger. 4](#_Toc486795643)

[2. Форматирование значений привязки и конвертеры значений. 5](#_Toc486795644)

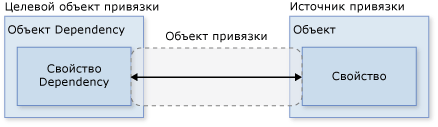
[3. Примечание 5](#_Toc486795645)

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. 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. Форматирование значений привязки и конвертеры значений.
2. 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/>