# «Привязка данных»

Привязка данных (data binding) в графической системе WPF представляет собою отношение, которое сообщает WPF о необходимости извлечения данных из свойства исходного объекта (Source) и использования её для задания значения некоторого свойства целевого объекта (Target) (и, в некоторых случаях, наоборот).

Объектом-источником может быть как элемент WPF, так и объект ADO.NET или пользовательский объект, хранящий данные. В данной лабораторной работе рассматривается связывание элементов управления WPF.

Рассмотрим пример приложения из двух элементов управления: ползунка (Slider) и текстового блока (TextBlock). При изменении положения ползунка размер шрифта текстового блока должен меняться. Такое поведение можно реализовать за счет обработки события изменения положения ползунка ValueChaned:

|  |
| --- |
| **Пример 1 Код XAML** |
| <Slider Minimum="8" Maximum="30" ValueChanged="Slider\_ValueChanged"></Slider>  <TextBlock x:Name="Message" FontSize="20">  Пример WPF-приложения для демонстрации привязки данных  </TextBlock> |
| **Код C#** |
| private void Slider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)  {  if (Message != null)  Message.FontSize = ((Slider)sender).Value;  } |
| **Результат** |
|  |

Как видно из исходного кода, возникает необходимость проверки существования объекта Message, т.к. первый вызов обработчика Slider\_ValueChanged происходит в момент обработки элемента Slider XAMLфайла, когда элемент TextBlock еще не обработан и, соответственно, объект Message еще не создан. Второй проблемой является несоответствие начального значения ползунка и начального размера шрифта.

Для решения поставленной задачи с помощью привязки данных, необходимо указать в качестве значения свойства FontSize текстового блока следующее выражение привязки:

{Binding ElementName=SliderFontSize, Path=Value}

Выражение привязки данных задается в виде расширения разметки XAML в фигурных скобках. Составляющие выражения привязки:

**Binding** – означает, что будет создан объект класса System.Windows.Data.Binding **ElementName** – имя исходного объекта,

**Path** – имя свойства (или **путь** до свойства) исходного объекта. Пример пути до свойства: Background.Opacity

|  |
| --- |
| **Пример 2 Код XAML** |
| <Slider Minimum="8" Maximum="30" x:Name="SliderFontSize"></Slider> |
| <TextBlock x:Name="Message" FontSize="{Binding ElementName=SliderFontSize, Path=Value}">  Пример WPF-приложения для демонстрации привязки данных  </TextBlock> |
| **Результат** |
|  |

В данном примере отсутствуют проблемы, обнаруженные в предыдущем примере. Начальные значения связанных свойств будут согласованы даже в том случае, если элемент TextBlock будет предшествовать элементу Slider.

## Задание 1

Проверьте реакцию среды разработки на неверные значения параметров ElementName и Path.

Проанализируйте сообщения, которые выводятся в окне вывода (Вид → Вывод) при построении и при запуске приложения.

## Режимы привязки

В выражении привязки с помощью параметра Mode можно задать одно из следующих пяти значений

режима привязки:

1. OneWay – целевое свойство обновляется при изменении исходного свойства.

Свойство

Исходный объект

Свойство

Целевой объект

OneWay

1. OneTime – первоначально значение исходного свойства копируется в целевое свойство, но дальнейшие изменения исходного свойства не учитываются.

Свойство

Исходный объект

Свойство

Целевой объект

OneTime

1. TwoWay - целевое свойство обновляется при изменении исходного свойства, исходное свойство

обновляется при изменении целевое свойства.

Свойство

Исходный объект

Свойство

Целевой объект

TwoWay

1. OneWayToSource – исходное свойство обновляется при изменении целевое свойства.

Свойство

Исходный объект

Свойство

Целевой объект

OneWayToSource

1. Default – значение по умолчанию. Если целевое свойство устанавливается пользователем (например, TextBox.Text, Slider.Value, CheckBox.IsChecked, …), то это TwoWay, в остальных случаях – это OneWay.

Пример выражения привязки с параметром Mode: {Binding ElementName=slider1, Path=Value, Mode=OneTime}

## Задание 2

Запустите приложение со следующим XAML-кодом:

<TextBox x:Name="t1" />

<TextBox x:Name="t2" Text="{Binding ElementName=t1, Path=Text}" />

<Slider x:Name="slider1" />

<Slider x:Name="slider2" Value="{Binding ElementName=slider1, Path=Value}" />

Определите различие в поведении полей t1 и t2 и модифицируйте код, чтобы устранить это различие.

## Задание 3

Дополните пример №2 текстовым полем ввода TextBox, в котором пользователь может ввести размер шрифта, и задайте выражения привязки таким образом, чтобы значение ползунка, текст текстового поля и размер шрифта текстового блока соответствовали друг другу.

|  |
| --- |
| ***Задание 4***  Модифицируйте приложения, разработанные в предыдущей лабораторной работе: удалите как можно больше обработчиков событий и реализуйте ту же функциональность приложения с помощью привязки данных.    Подсказки:    Свойство EditingMode (тип данных InkCanvasEditingMode) элемента управления InkCanvas нельзя напрямую связать с текстовым свойством выпадающего списка ComboBox или списка ListBox, т.к. в этом случае будет несовпадение типов. Для привязки данных необходимо, чтобы тип элементов списка совпадал с типом свойства EditingMode. Для этой цели необходимо добавить в ресурсы окна приложения (элемент Windows.Resources) массив (элемент x:Array) элементов типа InkCanvasEditingMode (артибут x:Type), данному ресурсу необходимо задать ключ (атрибут x:Key), который необходимо указать в свойстве ItemSource списка ListBox или выпадающего списка ComboBox. В этом случае можно будет осуществить привязку данных между свойством EditingMode и выделенным элементом списка:    <Window x:Class="WpfApplication1.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">  <Window.Resources>  <x:Array x:Key="MyEditingModes" x:Type="{x:Type InkCanvasEditingMode}">  <x:Static Member="InkCanvasEditingMode.Ink"/>  <x:Static Member="InkCanvasEditingMode.Select"/>  <x:Static Member="InkCanvasEditingMode.EraseByPoint"/>  <x:Static Member="InkCanvasEditingMode.EraseByStroke"/>  </x:Array>  </Window.Resources>  <StackPanel>  <InkCanvas EditingMode="{Binding ElementName=lbEditingModes, Path=SelectedValue}" />  <ListBox x:Name="lbEditingModes" ItemsSource="{StaticResource MyEditingModes}" />  </StackPanel>  </Window>    Аналогичным образом можно задать привязку данных между свойством DefaultDrawingAttributes и выделенным элементом списка (в данном случае массив x:Array будет содержать элементы типа DrawingAttributes):    <Window x:Class="WpfApplication1.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">  <Window.Resources>  <x:Array x:Key="MyDrawingAttributes" x:Type="{x:Type DrawingAttributes}">  <DrawingAttributes Color="Red" Width="3" Height="3"/>  <DrawingAttributes Color="Green" Width="10" Height="10"/>  <DrawingAttributes Color="Blue" Width="15" Height="15"/> </x:Array>  </Window.Resources>  <StackPanel>  <InkCanvas DefaultDrawingAttributes="{Binding ElementName=lbColors, Path=SelectedValue}" />  <ListBox x:Name="lbColors" ItemsSource="{StaticResource MyDrawingAttributes}" />  </StackPanel>  </Window>    Недостатком последнего примера является то, что все элементы в списке выводятся с текстом «System.Windows.Ink.DrawingAttributes». Для придания элементам списка осмысленного содержания, необходимо определить шаблон элементов (ListBox.ItemTemplate), в котором определить, каким образом элементы списка будут отображены на экране (например, в виде текстового блока, содержащего поле Color):  <ListBox x:Name="lbColors" ItemsSource="{StaticResource MyDrawingAttributes}"> <ListBox.ItemTemplate>  <DataTemplate> |

<TextBlock Text="{Binding Path=Color}"></TextBlock>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>