Собственный массив

<x:Array x:Key="list1" Type="{x:Type system:Int32}">

<system:Int32>1</system:Int32>

<system:Int32>2</system:Int32>

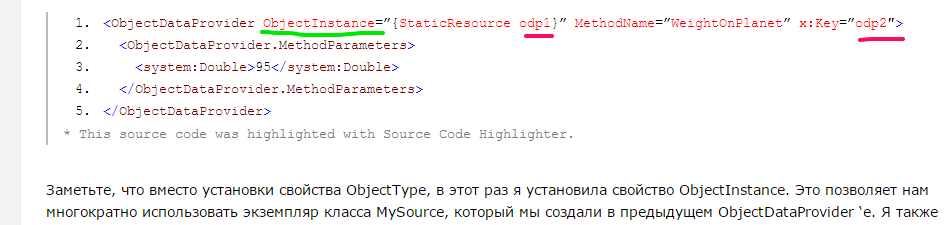
<system:Int32>3</system:Int32>

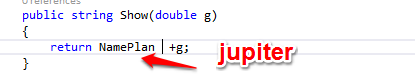
<system:Int32>4</system:Int32>

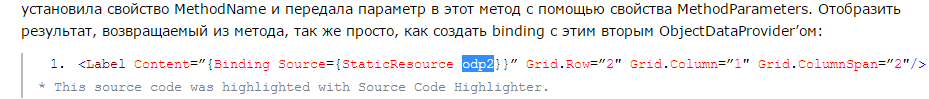
<system:Int32>5</system:Int32>

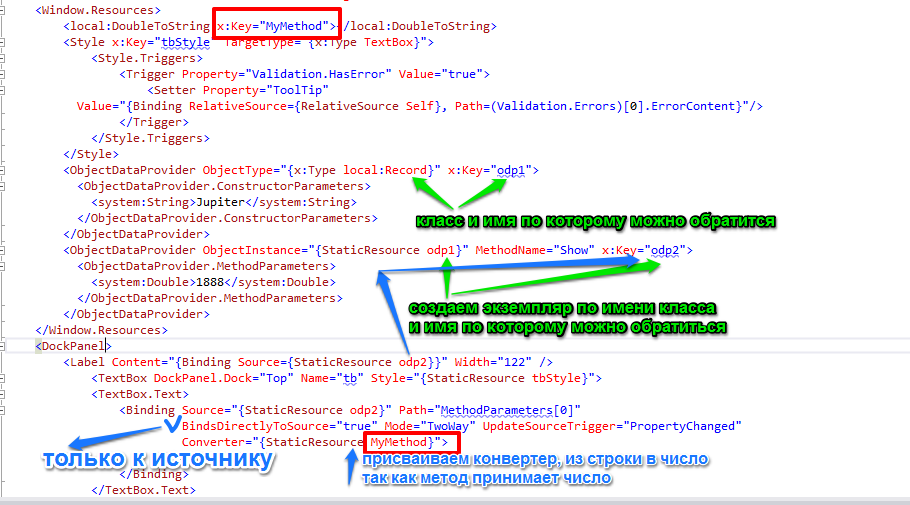
</x:Array>

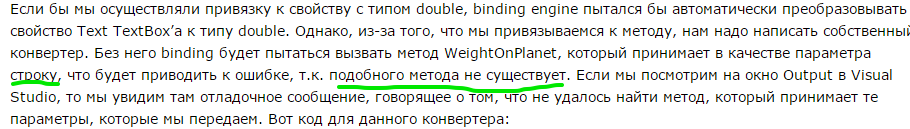
ObjectDataProvider



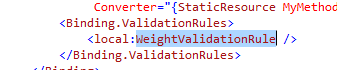






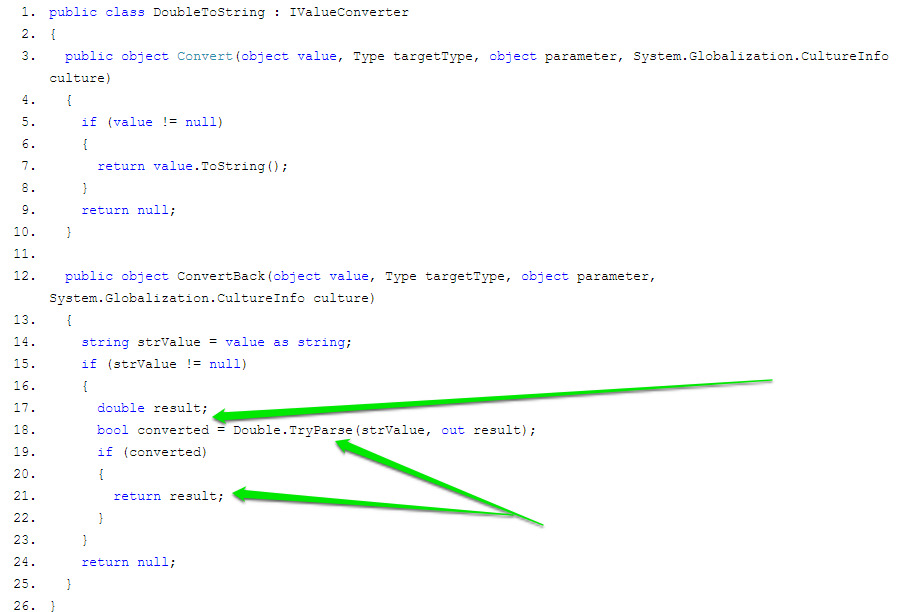


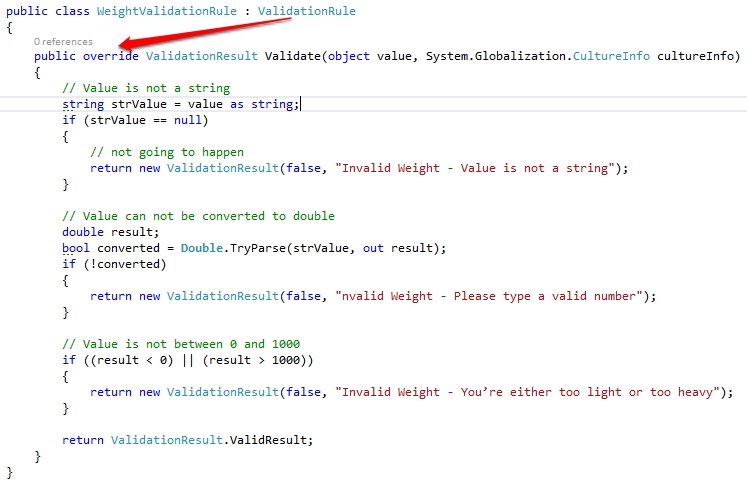
Добавление валидации

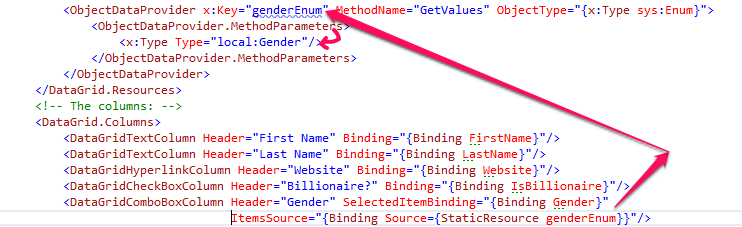


Не забываем указать стиль объекту которому принадлежит валидация









DependencyProperty

using System.Windows;

namespace WpfDependencyProperty

{

// Простое свойство зависимостей.

class SimpleProperty : DependencyObject

{

// Идентификатор свойства зависимости - поле представляющее свойство зависимости. Согласно условию должно иметь имя плюс слово Property в конце.

// Таким образом можно отделить имя свойства от имени свойства зависимости.

// свойство зависимости обязательно должно быть public и static

public static DependencyProperty TestDataProperty;

// Статический конструктор. Сработает до выполнения любого кода данного экземпляра.

static SimpleProperty()

{

// Регистрация свойства зависимости. Это нужно сделать до того, как свойство начнет использоваться в коде.

TestDataProperty = DependencyProperty.Register("TestData", typeof(string), typeof(SimpleProperty));

// параметр 1: Имя свойства.

// параметр 2: Тип данных свойства.

// параметр 3: Тип, которому принадлежит это свойство.

}

// Упаковка свойства зависимостей в традиционное свойство.

// Методы SetValue и GetValue унаследованы от класса DependencyObject

public string TestData

{

set

{

SetValue(TestDataProperty, value);

}

get

{

return (string)GetValue(TestDataProperty);

}

}

}

}

class MyFirstControl : FrameworkElement

{

// 1. Создание свойства зависимостей.

// Идентификатор свойства зависимости - поле представляющее свойство зависимости.

public static DependencyProperty DataProperty;

// 2. регистрация свойства зависимостей

static MyFirstControl()

{

// параметр 1: Имя свойства.

// параметр 2: Тип данных свойства.

// параметр 3: Тип, которому принадлежит это свойство.

DataProperty = DependencyProperty.Register("Data", typeof(int), typeof(MyFirstControl));

}

// 3. Упаковка свойства зависимостей в традиционное свойство.

// Методы SetValue и GetValue унаследованы от класса DependencyObject

public int Data

{

get

{

return (int)GetValue(DataProperty);

}

set

{

SetValue(DataProperty, value);

}

}

}

}

<Window x:Class="WpfDependency.MainWindow"

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

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

Title="Dependency Property Samples" Height="350" Width="525"

xmlns:myControls="clr-namespace:WpfDependency">

<Grid>

<!--Выражение привязки {Binding ....} может быть присвоено только свойству зависимостей-->

<myControls:MyFirstControl

x:Name="myFirstControl1"

Data="{Binding ElementName=slider1, Path=Value}">

</myControls:MyFirstControl>

<Button Name="button1"

Content="Get Value"

Height="23"

HorizontalAlignment="Left"

Margin="12,276,0,0"

VerticalAlignment="Top"

Width="75"

Click="button1\_Click" />

<Slider Name="slider1"

Height="23"

HorizontalAlignment="Left"

Margin="30,37,0,0"

VerticalAlignment="Top"

Width="437" />

</Grid>

</Window>

using System.Windows;

namespace WpfDependencyProperty

{

// Свойство зависимостей, которое использует метаданные

class PropertyMetadata : DependencyObject

{

// Поле представляющее свойство зависимости. Согласно условию должно иметь имя плюс слово Property в конце.

// Таким образом, можно отделить имя свойства от имени свойства зависимости.

public static readonly DependencyProperty TestDataProperty;

// Статический конструктор. Сработает до выполнения любого кода данного экземпляра.

static PropertyMetadata()

{

// Объект указывает какие службы вы хотите использовать вместе со своим свойством зависимостей (например, поддержку привязки данных, анимации и т.д.)

FrameworkPropertyMetadata metadata = new FrameworkPropertyMetadata("default value", FrameworkPropertyMetadataOptions.NotDataBindable);

// Регистрация свойства зависимости. Это нужно сделать до того, как свойство начнет использоваться в коде.

TestDataProperty = DependencyProperty.Register("TestData", typeof(string), typeof(PropertyMetadata), metadata, new ValidateValueCallback(Validate));

// параметр 1: Имя свойства.

// параметр 2: Тип данных свойства.

// параметр 3: Тип, которому принадлежит это свойство.

// параметр 4: Метаданные.

// параметр 5: Метод для проверки корректности значения.

}

// Упаковка свойства зависимостей в традиционное свойство.

// Методы SetValue и GetValue унаследованы от класса DependencyObject

public string TestData

{

set { SetValue(TestDataProperty, value); }

get { return (string)GetValue(TestDataProperty); }

}

// Проверка значения.

static bool Validate(object value)

{

if (value.ToString() == "error")

return false;

return true;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

namespace WpfDependencyPropCallback

{

// Свойство зависимостей и CallBack методы проверяющие значение установленное свойству.

class SimplePropClass : FrameworkElement

{

// Метаданные.

static FrameworkPropertyMetadata metadata = new FrameworkPropertyMetadata(

new PropertyChangedCallback(ChangedCallbackMethod) – метод Callback

, new CoerceValueCallback(CoerceValueCallbackMethod)); - на подобии валидации

// Свойство зависимостей.

public static readonly DependencyProperty MyDataProperty =

DependencyProperty.Register("MyData",

typeof(int),

typeof(SimplePropClass),

metadata,

new ValidateValueCallback(ValidateValueCallbackMethod));

// Свойство-обертка для свойства зависимостей.

public int MyData

{

get { return (int)GetValue(MyDataProperty); }

set

{

SetValue(MyDataProperty, value);

// В методах доступа нет смысла вставлять свой код, так как они используются только на этапе проектирования.

// При присвоении значений из XAML кода, данные будут записаны непосредственно в DependencyProperty минуя этот код.

MessageBox.Show("Hello");

}

}

// Срабатывает первым.

// Метод, который будет срабатывать при обновлении значения свойства для корректирования значения если оно не подходит. (Указывается через метаданные).

static object CoerceValueCallbackMethod(DependencyObject d, object baseValue)

{

if ((int)baseValue <= 100)

return baseValue;

return 100;

}

// Срабатывает вторым.

// Проверка на валидность введенных данных. (Привязывается при регистрации свойства).

static bool ValidateValueCallbackMethod(object value)

{

if ((int)value < 0)

return false;

return true;

}

// Срабатывает третьим.

// Метод который будет срабатывать при обновлении значения свойства. (Указывается через метаданные)

static void ChangedCallbackMethod(DependencyObject d, **DependencyPropertyChangedEventArgs** e)

{

Application.Current.MainWindow.Title = e.NewValue.ToString();

}

}

}