Тема 2. WPF

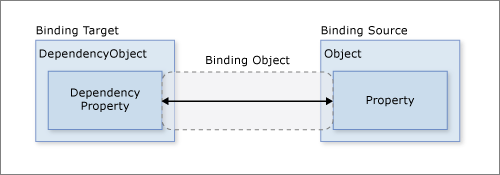
2.1. Привязка значений к элементу управления

<https://docs.microsoft.com/ru-ru/dotnet/desktop/wpf/data/?view=netdesktop-5.0>

Как работает?

Привязка значений к полям элемента управления осуществляется с использованием объекта Binding. При правильной привязке изменения значений данных сопровождаются правильными уведомлениями, привязанные к данным элементы автоматически отражают изменения. Привязка данных может также означать, что, если внешнее представление данных в элементе изменяется, то базовые данные могут автоматически обновляться для отражения изменений. Например, если пользователь изменяет значение в элементе TextBox, базовое значение данных автоматически обновляется, чтобы отразить это изменение.

Независимо от того, какой элемент привязывается и какой источник данных используется, каждая привязка всегда соответствует модели, показанной на приведенном ниже изображении.



Привязка данных является, по существу, мостом между целью привязки и источником привязки.

Контекст данных

Если привязка данных объявляется для элементов XAML, эти элементы разрешают привязку данных путем проверки текущего свойства DataContext. Как правило, контекст данных представляет собой объект источника привязки, используемый для оценки пути к значению источника привязки.

Если для объекта, в котором размещена привязка, не задано свойство DataContext, то проверяется свойство DataContext родительского элемента и т. д. до корня дерева объектов XAML.

Можно настроить разрешение привязки с использованием определенного объекта, а не контекста данных. Явное указание исходного объекта используется, например, при привязке цвета переднего плана объекта к цвету фона другого объекта. В этом случае контекст данных не нужен, так как привязка разрешается между этими двумя объектами.

При изменении свойства DataContext все привязки, на которые может повлиять контекст данных, оцениваются повторно.

<local:ContactListControl DataContext="{Binding Path=CurrentContactList}"/>

Направление потока данных

Поток данных связывания может идти от цели привязки к источнику привязки (например, исходное значение изменяется, когда пользователь изменяет значение TextBox) или от источника привязки к цели привязки (например, содержимое TextBox обновляется при изменениях в источнике привязки), если источник привязки предоставляет соответствующие уведомления.

Для управления потоком данных можно задать Binding.Mode.

OneWay приводит к автоматическому изменению свойства цели при изменении свойства источника, но изменения свойства цели не передаются обратно к свойству источника. Этот тип привязки подходит, если привязываемый элемент управления неявно доступен только для чтения. Если нет необходимости отслеживать изменения целевого свойства, можно работать в режиме привязки OneWay — в этом случае удастся избежать издержек режима привязки TwoWay.

TwoWay приводит к изменению либо исходного свойства, либо целевого свойства для автоматического обновления другого свойства. Этот тип привязки подходит для изменяемых форм или других полностью интерактивных сценариев пользовательского интерфейса. Большинство свойств по умолчанию применяют привязку OneWay, но некоторые свойства зависимости (обычно это свойства пользовательских элементов управления, таких как TextBox.Text и CheckBox.IsChecked) по умолчанию имеют значение привязки TwoWay. Существует способ определить программно, использует ли свойство зависимостей односторонние или двухсторонние привязки по умолчанию: для этого нужно получить метаданные свойства, воспользовавшись методом DependencyProperty.GetMetadata, а затем проверить логическое значение свойства FrameworkPropertyMetadata.BindsTwoWayByDefault.

OneWayToSource — это обратная привязка OneWay; она обновляет свойство источника при изменении свойства цели. Одним из примеров является пересчет исходного значения из пользовательского интерфейса.

<TextBox Text="{Binding Path=Nameб, Mode=TwoWay}" Grid.Column="1" Grid.Row="0"/>

Модель обновления

Модель обновления задается свойством UpdateSourceTrigger. Значение по умолчанию для большинства свойств зависимостей — PropertyChanged, а свойство Text имеет значение по умолчанию LostFocus.

Explicit Обновляет источник привязки только при вызове метода UpdateSource().

LostFocus Обновляет источник привязки каждый раз, когда целевой элемент привязки теряет фокус.

PropertyChanged Обновляет источник привязки сразу же при изменении целевого свойства привязки.

Для поддержки модели PropertyChanged обновляемый объект должен поддерживать интерфейс INotifyPropertyChanged:

public class Contact : INotifyPropertyChanged

{

private string \_name, \_birthDay;

private Gender \_gender;

public string Name

{

set { \_name = value; RaisePropertyChanged("Name"); }

get { return \_name; }

}

public Gender Gender

{

set { \_gender = value; RaisePropertyChanged("Gender"); }

get { return \_gender; }

}

public string BirthDay

{

set { \_birthDay = value; RaisePropertyChanged("BirthDay"); }

get { return \_birthDay; }

}

public event PropertyChangedEventHandler PropertyChanged = delegate{};

protected void RaisePropertyChanged(string property)

{

PropertyChanged(this, new PropertyChangedEventArgs(property));

}

2.2. Отображение элементов типа Enum.

Используется элемент типа ObjectDataProvider в свойстве Resources элементов управления WPF:

<Window.Resources>

<ObjectDataProvider x:Key="Genders"

MethodName="GetValues"

ObjectType="{x:Type sys:Enum}">

<ObjectDataProvider.MethodParameters>

<x:Type TypeName="local:Gender"/>

</ObjectDataProvider.MethodParameters>

</ObjectDataProvider>

Пространство sys определяется через

xmlns:sys="clr-namespace:System;assembly=mscorlib"

Подробнее <https://docs.microsoft.com/en-us/dotnet/api/system.windows.data.objectdataprovider?view=net-5.0>

Для использования локализованных значений используется:

1. аттрибуты:

В этом случае используется класс подобного содержания:

public class EnumTypeConverter : EnumConverter

{

public EnumTypeConverter(Type type) : base(type)

{

}

public override bool CanConvertFrom(ITypeDescriptorContext context, Type sourceType)

{

return sourceType == typeof(string) || TypeDescriptor.GetConverter(typeof(Enum)).CanConvertFrom(context, sourceType);

}

public override object ConvertFrom(ITypeDescriptorContext context, CultureInfo culture, object value)

{

if (value is string)

return GetEnumValue(EnumType, (string)value);

if (value is Enum)

return GetEnumDescription((Enum)value);

return base.ConvertFrom(context, culture, value);

}

public override object ConvertTo(ITypeDescriptorContext context, CultureInfo culture, object value, Type destinationType)

{

return value is Enum && destinationType == typeof(string)

? GetEnumDescription((Enum)value)

: (value is string && destinationType == typeof(string)

? GetEnumDescription(EnumType, (string)value)

: base.ConvertTo(context, culture, value, destinationType));

}

public static string GetEnumDescription(Enum value)

{

var fieldInfo = value.GetType().GetField(value.ToString());

var attributes = (DescriptionAttribute[])fieldInfo.GetCustomAttributes(typeof(DescriptionAttribute), false);

return (attributes.Length > 0) ? attributes[0].Description : value.ToString();

}

public static string GetEnumDescription(Type value, string name)

{

var fieldInfo = value.GetField(name);

var attributes = (DescriptionAttribute[])fieldInfo.GetCustomAttributes(typeof(DescriptionAttribute), false);

return (attributes.Length > 0) ? attributes[0].Description : name;

}

public static object GetEnumValue(Type value, string description)

{

var fields = value.GetFields();

foreach (var fieldInfo in fields)

{

var attributes = (DescriptionAttribute[])fieldInfo.GetCustomAttributes(typeof(DescriptionAttribute), false);

if (attributes.Length > 0 && attributes[0].Description == description)

return fieldInfo.GetValue(fieldInfo.Name);

if (fieldInfo.Name == description)

return fieldInfo.GetValue(fieldInfo.Name);

}

return description;

}

}

А Enum объекты описываются с использованием аттрибута Description.

[TypeConverter(typeof(EnumTypeConverter))]

public enum Gender

{

[Description("Муж")]

Male,

[Description("Жен")]

Female

}

2. Конвертеры типов

Первый способ имеет недостаток – слишком жесткая связь с исходным кодом. Динамическая подгрузка описаний возможна при использовании объектов-конвертеров. Например,

public class GenderEnumConverterRu : IValueConverter

{

const string \_female = "Женский";

const string \_male = "Мужской";

const string \_other = "Другой";

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

{

if (value is Gender[])

{

Gender[] arr = value as Gender[];

object[] ret = new object[arr.Length];

for (int i = 0; i < arr.Length; ++i)

ret[i] = Convert(arr[i], targetType, parameter, culture);

return ret;

}

else if (value is Gender)

{

switch ((Gender)value)

{

case Gender.Female: return \_female;

case Gender.Male: return \_male;

default: return \_other;

}

}

return null;

}

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

{

if (value is string[])

{

Gender[] arr = value as Gender[];

object[] ret = new object[arr.Length];

for (int i = 0; i < arr.Length; ++i)

ret[i] = Convert(arr[i], targetType, parameter, culture);

return ret;

}

switch ((string)value)

{

case \_female: return Gender.Female;

case \_male: return Gender.Male;

default: return Gender.Male;

}

}

}

В Xaml коде необходимо указать конвертер

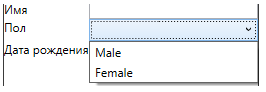
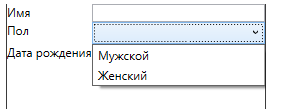
<ComboBox SelectedItem="{Binding Path=Gender}" Grid.Column="1" Grid.Row="1"

ItemsSource="{Binding Source={StaticResource Genders}, Converter={StaticResource enumConverter}}"/>

А сам конвертер указать в ресурсах элемента управления.

<local:GenderEnumConverterRu x:Key="enumConverter"/>

</Window.Resources>

Было стало

Задание:

1. Скачать проект с https://github.com/erv1988/WpfApp1.

2. Доделать обработчики кнопок сохранения списка контактов, редактирования и удаления контакта. Для редактирования контакта использовать диалог создания нового контакта.

3. Установить согласно инструкции пакет Metro.Mahapps и изменить стиль приложения.

<https://mahapps.com/docs/guides/quick-start>

3. Создать Enum с вариантами хранения списка контактов: Xml, JSON. Сделать альтернативную реализацию ContactListFrom**Json**. Использовать класс Newton.Json

4. В заголовок главного окна добавить кнопку настроек, при нажатии на который будет открываться диалоговое окно с настройками. Пока настройка одна – источник контактов. Добиться того, чтобы список контактов обновлялся при смене источника.

Бонус: изменить классы ContactList\* так, чтобы список контактов был один и изменялся лишь способ их сохранения.

5. Изучить класс Timer. Создать таймер, который с определенной периодичностью сохраняет список контактов. Удалить кнопку ручного сохранения списка контактов.

6. Изучить класс Trigger. Изменить шаблон в ContactListControl таким образом, чтобы визуально (цветом или иконкой) можно было определить пол контакта (например, юноши в голубой рамке, а девушки – в розовой).

Вопросы писать в чат.