

Разработка приложений на платформе .NET

Лекция 16

Dependency Property

Расширение разметки (Markup Extensions)

Привязки (binding)

Сегодня

- Свойства зависимости (Dependency Property)
- Расширение разметки (Markup Extensions)

- Привязка элементов
- Привязка данных
- Преобразование данных
 - Конвертеры
 - `StringFormat`
- Множественная привязка
 - `MultiBinding`
 - `MultiValueConverter`
 - `StringFormat`

Dependency Property

- Расширение обычных свойств
- Для **Dependency Property** важен не только и не столько очередность установки свойства, а способ установки
- **Dependency Property** фактически помнит значения, установленные каждым из способов, а **WPF** использует самое приоритетное значение. При очистке приоритетного значения **WPF** использует следующее по приоритету менее приоритетное значение
- Порядок и приоритет определения значения свойства зависимости:

П
р
и
о
р
и
т
е
т



Установлено системой коррекции
Установлено анимацией
Установлено напрямую в XAML, коде или с помощью Binding
Установлено с помощью TemplateParent (триггер имеет приоритет перед простой установкой)
Установлено триггерами стиля
Установлено триггерами шаблона
Установлено в стиле
Установлено в стиле по умолчанию (темы)
Свойство унаследовано в визуальном дереве
Значение по умолчанию

Создание свойства зависимости

- Объявление и регистрация свойства
 - **public static readonly DependencyProperty MyProperty = DependencyProperty.Register("My", typeof(int), typeof(MainWindow), new FrameworkPropertyMetadata(20));**
- Создание обертки – обычного свойства .NET
 - ```
public int My
{
 get { return (int)GetValue(MyProperty); }
 set { SetValue(MyProperty, value); }
}
```
  - Нельзя использовать в обертке свойства зависимости какую-либо дополнительную логику, поскольку WPF вызывает **GetValue** и **SetValue** напрямую, а при использовании в коде в основном вызываются обертки свойства, а не **GetValue / SetValue**.
- По соглашению свойство зависимости имеет “суффикс” **Property**, а свойство обертка – без нет
- **PropertyMetadata** (наследники **UIPropertyMetadata**, **FrameworkPropertyMetadata**) задает:
  - Значение по умолчанию
  - Опции использования свойства зависимости (например, возможность использования в анимации, наследование, использование двустороннего **binding** по умолчанию, влияние на отрисовку и процесс измерения элемента и т.д.)
  - Методы отвечающие за корректировку значений свойства

# Демонстрация

---

Dependency Property

# Сегодня

---

- Свойства зависимости (Dependency Property)
- Расширение разметки (Markup Extensions)
- Привязка элементов
- Привязка данных
- Преобразование данных
  - Конвертеры
  - StringFormat
- Множественная привязка
  - MultiBinding
  - MultiValueConverter
  - StringFormat

# Расширение разметки

- Позволяет получить значение в зависимости от переданных параметров
- Определение
  - Создать класс наследник от `MarkupExtension`
  - Переопределить метод `ProvideValue`
  - Пометить класс атрибутом `[MarkupExtensionReturnType()]` с указанием возвращаемого типа

```
[MarkupExtensionReturnType(typeof(string))]
class MyExtension : MarkupExtension
{
 public int MyProperty { get; set; }
 public override object ProvideValue(IServiceProvider serviceProvider)
 {
 return (MyProperty + 3).ToString();
 }
}
```

- Использование (по соглашению `Extension` можно опустить)

```
<TextBlock Text="{local:My MyProperty=5}" />
<Button>
 <local:My MyProperty="5"/>
</Button>
```

# Демонстрация

---

MarkupExtension



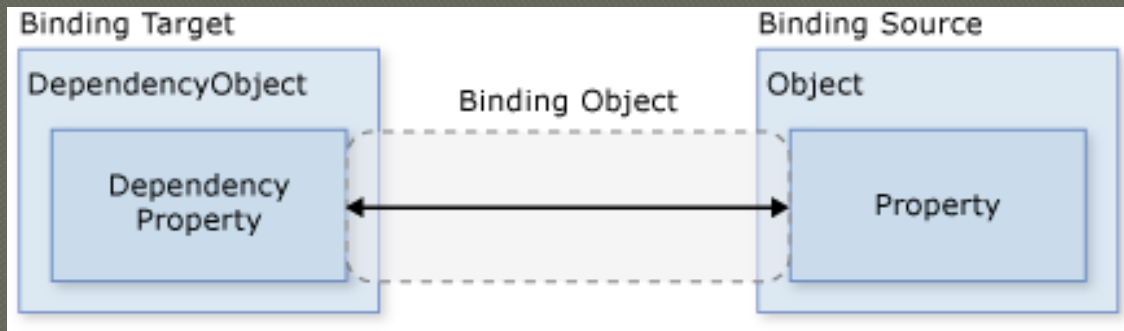
# Сегодня

---

- Свойства зависимости (Dependency Property)
- Расширение разметки (Markup Extensions)
- **Привязка элементов**
- Привязка данных
- Преобразование данных
  - Конвертеры
  - `StringFormat`
- Множественная привязка
  - `MultiBinding`
  - `MultiValueConverter`
  - `StringFormat`

# Binding

- Привязка данных (**binding**) – это отношение, которое сообщает WPF о необходимости извлечения некоторой информации из исходного объекта и использования его для установки свойства в целевом объекте.
- Целевое свойство – обязательно свойство зависимости
- Объект источник – абсолютно любой объект



- Поддерживает автоматическое обновление свойств при изменении объектов (при наличии уведомления об изменении)

# Привязка к элементу

- Объект **Binding** в XAML устанавливается для целевого свойства
- **ElementName** – задает имя элемента-источника
- **Path** – задает путь к свойству в объекте-источнике

```
<Slider Name="slider" Value="10"/>
```

```
<TextBlock Text="Тестовый текст" FontSize="{Binding
ElementName=slider, Path=Value}" />
```

- WPF автоматически получает уведомления об изменении свойства источника и изменяет целевое свойство.

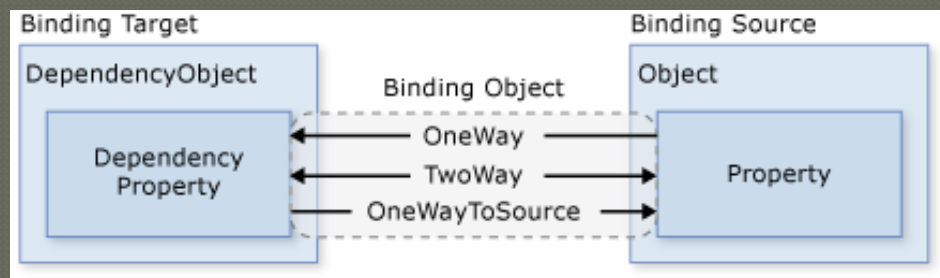
# Демонстрация

---

Привязка к элементу

# Направление привязки

- задается свойством **Mode**
- **OneWay** – целевое свойство обновляется при изменении исходного свойства
- **TwoWay** - целевое свойство обновляется при изменении исходного свойства, а исходное свойство обновляется при изменении целевого свойства
- **OneTime** – целевое свойство устанавливается один раз на основе начального значения исходного свойства. Далее целевое свойство не изменяется
- **OneWayToSource** – обратно **OneWay**. Исходное свойство обновляется при изменении целевого свойства. Целевое свойство никогда не изменяется.
- **Default** – **OneWay** или **TwoWay**, устанавливается при определении самого свойства.



```
<Slider Name="slider" Minimum="1" Maximum="40" Value="10" />
```

```
<TextBox Text="{Binding ElementName=slider, Path=Value, Mode=TwoWay}"/>
```

# Демонстрация

---

Направление привязки

# Момент обновления привязки

---

- задается свойством **UpdateSourceTrigger**
- **PropertyChanged** – источник обновляется немедленно, когда изменяется целевое свойство
- **LostFocus** – источник обновляется немедленно, когда элемент теряет фокус и целевое свойство изменилось
- **Explicit** – источник не обновится пока не будет вызван метод `BindingExpression.UpdateSource()`;
  - `BindingExpression be = slider.GetBindingExpression(Slider.ValueProperty);`
  - `be.UpdateSource();`
- **Default** – поведение определено при задании свойства. Почти всегда это `PropertyChanged`, но для `TextBox.Text` – `LostFocus`.

# Создание привязки в коде

---

```
Binding binding = new Binding();
binding.Source = slider;
binding.Path = new PropertyPath("Value");
binding.Mode = BindingMode.TwoWay;
textBlock.SetBinding(TextBlock.FontSizeProperty, binding);
```



# Сегодня

---

- Свойства зависимости (Dependency Property)
- Расширение разметки (Markup Extensions)
- Привязка элементов
- **Привязка данных**
- Преобразование данных
  - Конвертеры
  - `StringFormat`
- Множественная привязка
  - `MultiBinding`
  - `MultiValueConverter`
  - `StringFormat`

# Привязка данных

- Выбор источника данных. Свойства **Binding**
  - **ElementName** – имя элемента-источника при привязке к элементу WPF
  - **Source** – имя объекта при привязке не к элементу WPF
  - **RelativeSource** – задает источник связывания, относительно текущего элемента в визуальном дереве.
  - **DataContext** – если источник объекта для **Binding** не установлено, по **Binding** ищет элемент в визуальном дереве, у которого установлено свойство **DataContext** и считает источником значение свойства **DataContext**
- **Path** – путь к свойству в объекте-источнике.
  - Может быть не задан, тогда привязка осуществляется к самому объекту

# RelativeSource

---

## ○ Значения **RelativeSource**

- **Self** – ссылка на сам элемент
- **FindAncestor** – поиск объекта заданного типа выше по визуальному дереву. **AncestorType** – указывает элемент какого типа ищется
- **PreviousData** – предыдущий элемент в списке (**ItemsControl**)
- **TemplatedParent** – используется в шаблоне. Обозначает элемент, к которому применен шаблон
- `<TextBlock Text="{Binding RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type Window}}, Path=Height}"/>`

# Демонстрации

---

Source  
Relative Source  
DataContext

# Привязка к данным

---

- ◎ Привязка к данным – это создание связи между двумя свойствами разных объектов
  - Не обязательно визуальных
- ◎ Характеристики связи
  - Источник и получатель
  - Направление
  - Динамичность (один раз или постоянно)
  - Сложность (один к одному или привязка к коллекции)

# Уведомления об изменении

---

- Для того, чтобы целевое свойство автоматически обновлялось необходимо, чтобы привязанное свойство-источник извещало о своем изменении
- Извещения об изменении
  - Поддерживается всеми `DependencyProperty`
  - Частный класс должен реализовывать интерфейс **`INotifyPropertyChanged`** и вызывать событие `PropertyChanged` при изменении свойства
  - Коллекции должны реализовывать интерфейс **`ICollectionChanged`**. Например, коллекция **`ObservableCollection<T>`**

# Демонстрация

---

Уведомления об изменении

# Сегодня

---

- Свойства зависимости (Dependency Property)
- Расширение разметки (Markup Extensions)
- Привязка элементов
- Привязка данных
- Преобразование данных
  - Конвертеры
  - **StringFormat**
- Множественная привязка
  - MultiBinding
  - MultiValueConverter
  - StringFormat



# Конвертеры

- Конвертер – класс, преобразующий один тип в другой
- XAML использует их повсеместно
  - Преобразование строки в объект
- Реализует **IValueConverter**
  - Методы **Convert** и **ConvertBack**
  - Преобразование от источника к целевому свойству:
    - `object Convert(object value, Type targetType, object parameter, CultureInfo Culture)`
  - Преобразование от целевого свойства к источнику:
    - `object ConvertBack(object value, Type targetType, object parameter, CultureInfo Culture)`
- Конвертер необходимо пометить атрибутом **ValueConversion**, указывающем типы между которыми происходит преобразование
  - `[ValueConversion(typeof(Product), typeof(int))]`

# Дополнительные свойства

---

- **ConverterParameter** – дополнительный параметр. Любой объект
- **ConverterCulture** – культура конвертера
  - Нужна для локализации

# Использование конвертера

---

Для использования конвертера необходимо:

- Указать пространство имен, где находится конвертер
  - `<xmlns:src="clr-namespace:MyProg">`
- Создать в ресурсах экземпляр конвертера (обычно)
  - `<src:MyConverter x:Key="myconv">`
- Использовать конвертер в **Binding**
  - `{Binding ... Converter={StaticResource myconv} }`

# Часто реализуют как MarkupExtension

```
[MarkupExtensionReturnType(typeof(BoolToVisibilityConverter))]
public class BoolToVisibilityConverter: MarkupExtension, IValueConverter
{
 public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
 {
 return; // логика конвертера
 }

 public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
 {
 throw new NotSupportedException();
 }

 public override object ProvideValue(IServiceProvider serviceProvider)
 {
 if (_converter == null) _converter = new BoolToVisibilityConverter();
 return _converter;
 }

 private static BoolToVisibilityConverter _converter = null;
}
```

- Использовать конвертер в Binding
  - `<TextBlock Visibility="{Binding ... Converter={converters: BoolToVisibilityConverter} }"/>`
- Конвертер должен быть stateless

# Демонстрация

---

Конвертер

# Форматирование

- Форматирование вывода (**StringFormat**):
  - Text="{Binding ... StringFormat=Цена {0} руб}"
- Можно использовать обычные шаблоны форматированного вывода строк
- Дополнительный параметр должен быть только один.
  - Text="{Binding ... StringFormat=Цена {0} \* {0} руб}"
  - Неверно Text="{Binding ... StringFormat=Цена {2} \* {1} руб}"
- Если параметр используется первым, то необходимо добавить пустые скобки {}
  - Text="{Binding ... StringFormat={}{0} руб}"
- Замена или дополнение конвертеру

# Демонстрация

---

StringFormat

# Сегодня

---

- Свойства зависимости (Dependency Property)
- Расширение разметки (Markup Extensions)
  
- Привязка элементов
- Привязка данных
- Преобразование данных
  - Конвертеры
  - `StringFormat`
- Множественная привязка
  - `MultiBinding`
  - `MultiValueConverter`
  - `StringFormat`



# MultiBinding

- Привязка нескольких источников к одному целевому свойству

```
<TextBlock>
 <TextBlock.Text>
 <MultiBinding StringFormat="{0} {1}">
 <Binding Path="Cost"/>
 <Binding Path="Currency"/>
 </MultiBinding>
 </TextBlock.Text>
</TextBlock>
```

- Целевое свойство изменяется при изменении любого из свойств
- Необходимо использовать либо **MultiValueConverter**, либо **StringFormat**
- **StringFormat** может использовать столько параметров, сколько объектов **Binding** содержится в **MultiBinding**. Очередность параметров соответствует очередности объектов **Binding**

# MultiValueConverter

---

- Реализует **IMultiValueConverter**
  - Методы **Convert** и **ConvertBack**
  - Преобразование от источников к целевому свойству:  
`object Convert(object[] values, Type targetType, object parameter, System.Globalization.CultureInfo culture)`
  - Преобразование от целевого свойства к источнику:  
`object[] ConvertBack(object value, Type[] targetTypes, object parameter, System.Globalization.CultureInfo culture)`
- Отличие от обычных конвертеров только во множестве **values** (и соответственно типов)

# Демонстрации

---

MultiBinding  
MultiValueConverter  
StringFormat