## Создание свойств зависимостей

В предыдущих темах мы рассмотрели теоретические основы свойств зависимостей, теперь посмотрим, как мы можем определять какие-то свои свойства зависимостей. Итак, определим в нашем проекте новый класс Phone:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | public class Phone : DependencyObject  {      public static readonly DependencyProperty TitleProperty;      public static readonly DependencyProperty PriceProperty;        static Phone()      {          TitleProperty = DependencyProperty.Register("Title", typeof(string), typeof(Phone));          PriceProperty = DependencyProperty.Register("Price", typeof(int), typeof(Phone));      }      public string Title      {          get { return (string)GetValue(TitleProperty); }          set { SetValue(TitleProperty, value); }      }      public int Price      {          get { return (int)GetValue(PriceProperty); }          set { SetValue(PriceProperty, value); }      }  } |

Если мы хотим применять свойства зависимостей, то нам надо унаследовать свой класс от абстрактного класса DependencyObject. В нашем классе мы определяем два свойства зависимостей: TitleProperty и PriceProperty. Обратите внимание, что они объявляются с модификаторами public static readonly.

Затем свойства регистрируются в статическом конструкторе нашего класса с помощью метода **Register**. И в конце для них создаются обычные свойства-обертки, в которых мы получаем доступ к значению свойств с помощью методов GetValue и SetValue.

Теперь используем наш класс. Для этого определим следующую разметку XAML:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | <Window x:Class="DependencyApp.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:DependencyApp"          mc:Ignorable="d"          Title="MainWindow" Height="250" Width="300" FontSize="20">      <Window.Resources>          <local:Phone Price="600" Title="iPhone 6S" x:Key="iPhone6s" />      </Window.Resources>      <Grid x:Name="grid1" DataContext="{StaticResource iPhone6s}">          <Grid.RowDefinitions>              <RowDefinition />              <RowDefinition />              <RowDefinition />          </Grid.RowDefinitions>          <Grid.ColumnDefinitions>              <ColumnDefinition />              <ColumnDefinition />          </Grid.ColumnDefinitions>          <TextBlock Text="Модель" />          <TextBlock Text="{Binding Title}" Grid.Column="1"  />          <TextBlock Text="Цена" Grid.Row="1" />          <TextBox Text="{Binding Price, Mode=TwoWay}" Grid.Column="1" Grid.Row="1"  />          <Button Content="Check" Grid.Row="2" Grid.Column="2" Click="Button\_Click" />      </Grid>  </Window> |

Здесь в ресурсах окна определяется объект Phone с установкой его свойств Price и Name:

|  |  |
| --- | --- |
| 1 | <local:Phone Price="600" Title="iPhone 6S" x:Key="iPhone6s" /> |

Данный ресурс имеет ключ iPhone6s, по которому мы можем к нему обратиться. Далее для контейнера Grid мы задаем этот ресурс как контекст данных:

|  |  |
| --- | --- |
| 1 | <Grid x:Name="grid1" DataContext="{StaticResource iPhone6s}"> |

Установка контекста данных позволяет внутри грида привязать отдельные элеметы к свойства ресурса, то есть объекта Phone:

|  |  |
| --- | --- |
| 1 | <TextBox Text="{Binding Price, Mode=TwoWay}" Grid.Column="1" Grid.Row="1"  /> |

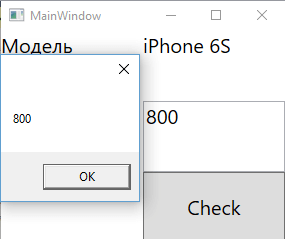
То же самое и для элемента TextBlock. Однако для TextBox у нас действует не просто привязка, а двусторонняя привязка, обозначенная параметром Mode=TwoWay. А это значит, что любые изменения свойства Price в ресурсе будут отображаться в текстовом поле. И наоборот - любые изменения в текстовом поле будут менять значения в ресурсе.

В этом состоит одно из преимуществ использования свойств зависимостей - для обычных свойств подобные привязки бы не работали.

Для проверки значения ресурса я добавил кнопку, у которой установил обработчик нажатия - Button\_Click. И также добавим код этого обработчика в файл кода C#:

|  |  |
| --- | --- |
| 1  2  3  4  5 | private void Button\_Click(object sender, RoutedEventArgs e)  {      Phone phone = (Phone)this.Resources["iPhone6s"]; // получаем ресурс по ключу      MessageBox.Show(phone.Price.ToString());  } |

Запустим приложение, изменим значение в текстовом поле, например, на 800 и нажмем на кнопку:



### **Добавление валидации**

WPF предоставляет два способа валидации значения свойства:

* **ValidateValueCallback**: делегат, который возвращает true, если значение проходит валидацию, и false - если не проходит
* **CoerceValueCallback**: делегат, который может подкорректировать уже существующее значение свойства, если оно вдруг не попадает в диапазон допустимых значений

При установке значения сначала срабатывает делегат ValidateValueCallback, который возвращает true, если значение проходит валидацию. Далее срабатывает делегат CoerceValueCallback, который может модифицировать это значение, если оно вдруг является некорректным. В случае, если эти два делегата успешно отработали, то срабатывает делегат PropertyChangedCallback, который извещает систему об изменении значени свойства.

При этом нам необязательно использовать сразу оба эти делеата, можно применять лишь один из них, а можно их комбинировать вместе.

Вначале применим делегат ValidateValueCallback. Для этого изменим код класса Phone на следующий:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | public class Phone : DependencyObject  {      public static readonly DependencyProperty TitleProperty;      public static readonly DependencyProperty PriceProperty;        static Phone()      {          TitleProperty = DependencyProperty.Register("Title", typeof(string), typeof(Phone));            FrameworkPropertyMetadata metadata = new FrameworkPropertyMetadata();            PriceProperty = DependencyProperty.Register("Price", typeof(int), typeof(Phone), metadata,              new ValidateValueCallback(ValidateValue));      }        private static bool ValidateValue(object value)      {          int currentValue = (int)value;          if (currentValue >= 0) // если текущее значение от нуля и выше              return true;          return false;      }        public string Title      {          get { return (string)GetValue(TitleProperty); }          set { SetValue(TitleProperty, value); }      }      public int Price      {          get { return (int)GetValue(PriceProperty); }          set { SetValue(PriceProperty, value); }      }  } |

Делегат ValidateValueCallback указывает на метод ValidateValue, который в качестве параметра принимает новое значение свойства. Если оно равно нулю или выше, так как отрицательные цены не имеют смысла, то значение проходит валидацию, и метод возвращает true.

При подобных проверках надо учитывать, что в процессе запуска приложения значения ресурса Phone устанавливаются два раза. Первый раз свойство Price получает значение по умолчанию для типов int, то есть число 0. И только потом на следующем этапе инициализации оно получает число 600, которое и установлено в ресурсе Phone. Поэтому если мы, к примеру, изменим логику валидации:

|  |  |
| --- | --- |
| 1  2 | if (currentValue > 0)      return true; |

То мы получим ошибку. Ведь число 0 теперь не является допустимым, но на начальной инициализации объекта Phone, именно 0 устанавливается в качестве значения для свойства Price. Поэтому подобные моменты надо учитывать.

Теперь, если мы запустим приложение и попробуем установить число меньше 0, то оно не пройдет валидацию, и ресурс Phone сохранит старое значение для свойства Price.

Теперь применим делегат **CoerceValueCallback:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | public class Phone : DependencyObject  {      public static readonly DependencyProperty TitleProperty;      public static readonly DependencyProperty PriceProperty;        static Phone()      {          TitleProperty = DependencyProperty.Register("Title", typeof(string), typeof(Phone));            FrameworkPropertyMetadata metadata = new FrameworkPropertyMetadata();          metadata.CoerceValueCallback = new CoerceValueCallback(CorrectValue);            PriceProperty = DependencyProperty.Register("Price", typeof(int), typeof(Phone), metadata,              new ValidateValueCallback(ValidateValue));      }        private static object CorrectValue(DependencyObject d, object baseValue)      {          int currentValue = (int)baseValue;          if (currentValue > 1000)  // если больше 1000, возвращаем 1000              return 1000;          return currentValue; // иначе возвращаем текущее значение      }        private static bool ValidateValue(object value)      {          int currentValue = (int)value;          if (currentValue >= 0) // если текущее значение от нуля и выше              return true;          return false;      }        public string Title      {          get { return (string)GetValue(TitleProperty); }          set { SetValue(TitleProperty, value); }      }      public int Price      {          get { return (int)GetValue(PriceProperty); }          set { SetValue(PriceProperty, value); }      }  } |

В данном случае мы будем корректировать значение свойства Price, так как оно изменяется через текстовое поле. Для этого делегат CoerceValueCallback будет вызывать метод CorrectValue. Данный метод должен принимать два параметра: **DependencyObject** - валидируемый объект (в данном случае объект Phone) и **object** - новое значение для свойства Price (либо для другого устанавливаемого свойства).

В самом методе CorrectValue() мы получаем новое значение и модифицируем его, если оно некорректно.

Единственное, что надо учитывать при работе с делегатом CoerceValueCallback, то, что он вызывается только тогда, когда вызов делегата ValidateValueCallback возвращает значение true.

Больше нам ничего менять не надо, разметка xaml остается прежней. И если мы запустим приложение и введем в текстоое поле некорректное значение, то сработает этот делегат, который подкорректирует значение.