Windows Phone

Создание приложений Silverlight

Лекция 4

Windows Phone

Улучшение приложения

Раздел 4.1

Темы раздела

- Управление свойствами элементов Silverlight
- Редактирование XAML-кода элементов Silverlight
- Вывод окна с сообщением
- Добавление и использование ресурсов
- Выбор способа размещения ресурсов

Элементы Silverlight

- Базовыми часто используемыми элементами пользовательского интерфейса Silverlight являются:
  - `TextBlock` — для вывода на экран сообщений и текста
  - `TextBox` — для получения вводимых пользователями данных
  - `Button` — для получения событий
- В приложениях эти элементы используются совместно

Элементы Silverlight как объекты

- Визуальные элементы Silverlight являются объектами
- Объект является экземпляром определённого типа элемента
- Можно управлять значениями свойств и вызывать методы этих объектов
- Результатом этих действий будет изменение внешнего вида объектов на экране

Свойства элемента TextBlock

```
resultTextBlock.Text = result.ToString();
```

- Чтобы изменить текст в элементе `TextBlock`, нужно присвоить строку с текстом свойству `Text`
- При установке значения свойства выполняется программный код, который изменяет значение текста и перерисовывает элемент на экране

Использование метода TryParse

```
float v1 = 0;
if (!int.TryParse(firstNumberTextBox.Text, out v1))
{
    // Код для обработки введённого недопустимого значения
}
```

- Метод `TryParse` пытается преобразовать введённый текст в целое число
- Если преобразование завершится неудачей, метод вернёт значение `false`
- Этот метод не генерирует

Изменение цвета текста

```
float v1 = 0;

if (!float.TryParse(firstNumberTextBox.Text, out v1))
{
    firstNumberTextBox.Foreground =
                    new SolidColorBrush(Colors.Red);
    return;
}
```

- В этом коде при вводе недопустимого значения в элемент TextBox цвет текста становится красным

Изменение цвета текста

```
float v1 = 0;

if (!float.TryParse(firstNumberTextBox.Text, out v1))
{
    firstNumberTextBox.Foreground =
                    new SolidColorBrush(Colors.Red);
    return;
}
```

- При выводе текста используется объект Silverlight *кисть*
- Часто используется кисть непрерывной заливки, но можно также создать кисть на основе изображения или

Настройка кисти

```
private SolidColorBrush errorBrush =
                        new SolidColorBrush(Colors.Red);
private Brush correctBrush = null;
```

- В программе используются две кисти
  - кисть `errorBrush` используется в случае ввода недопустимого значения
  - кисть `correctBrush` используется по умолчанию при отсутствии ошибок

Сохранение кисти по умолчанию

```
private void calculateResult()
{
    bool errorFound = false;

    if (correctBrush == null)
        correctBrush = firstNumberTextBox.Foreground;

    // остальной код метода

}
```

- При первом вызове метода сохраняется текущая кисть элемента TextBox

Использование кистей

```
private void calculateResult()
{
    // настройка кистей

    if (!float.TryParse(firstNumberTextBox.Text, out v1))
    {
        firstNumberTextBox.Foreground = errorBrush;
        errorFound = true;
    }
    else
    {
        firstNumberTextBox.Foreground = correctBrush;
    }
}
```

you can make bushes from images and gradients

Изменение XAML-кода элемента

- XAML-код используется для настройки визуальных элементов Silverlight, которые размещаются на странице приложения
- Можно изменить объявление элемента `TextBox`, чтобы при его использовании выводилась клавиатура для ввода чисел
- Это лучше сделать в XAML-коде, хотя можно и в коде на языке C#

XAML-код элемента TextBlock

```
<TextBox Height="72" HorizontalAlignment="Left"
Margin="8,175,0,0" Name="secondNumberTextBox" Text="0"
VerticalAlignment="Top" Width="460" TextAlignment="Center"
/>
```

- Этот XAML-код описывает настройки свойств элемента TextBox
- Код содержит настройки расположения элемента на экране и выравнивания текста
- Все настройки элемента представляются набором пар «имя—

Свойство InputScope

- Тип клавиатуры для элемента TextBox выбирается на основе значения свойства InputScope элемента TextBox
- Элемент InputScope является набором значений типа InputScopeName
- В XAML-коде описания элемента свойство InputScope может быть представлено
в виде элемента или атрибута

Описание свойства InputScope

```
<TextBox Height="72" HorizontalAlignment="Left"
Margin="8,19,0,0" Name="firstNumberTextBox" Text="0"
VerticalAlignment="Top" Width="460"
TextAlignment="Center">
    <TextBox.InputScope>
        <InputScope>
            <InputScopeName NameValue="Digits" />
        </InputScope>
    </TextBox.InputScope>
</TextBox>
```

```
<TextBox InputScope="Number" Height="72"
HorizontalAlignment="Left" Margin="144,44,0,0"
Name="startHourTextBox" Text="00" VerticalAlignment="Top"
Width="104"
TextAlignment="Center" />
```

Атрибуты и элементы

```
<TextBox HorizontalAlignment="Left" Margin="8,175,0,0"
Name="secondNumberTextBox" Text="0"
VerticalAlignment="Top" Width="460"
TextAlignment="Center">
    <TextBox.Height>
        72
    </TextBox.Height>
</TextBox>
```

- Одни и те же свойства элементов могут быть представлены в XAML-коде и в виде атрибутов, и в виде элементов

Настройка свойств в коде на C#

```
// Создание нового элемента InputScope
InputScope digitScope = new InputScope();

// Создание нового элемента InputScopeName
InputScopeName digits = new InputScopeName();
// Установка свойства name в значение Digits
digits.NameValue = InputScopeNameValue.Digits;

// Добавление элемента InputScopeName в элемент InputScope
digitScope.Names.Add(digits);

// Присвоить элементу TextBox созданное свойство
firstNumberTextBox.InputScope = digitScope;
```

- Этот код также устанавливает значение свойства InputScope элемента TextBox

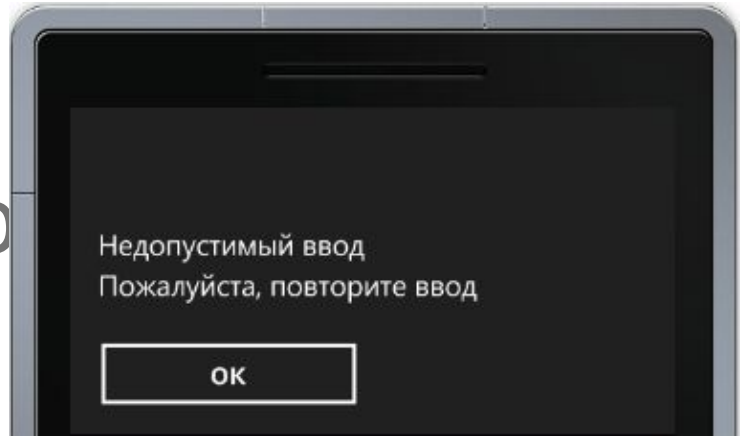
Вывод окна с сообщением

- Можно вывести на экран окно с сообщением с информацией для пользователя или для запроса подтверждения выполняемого действия
- Класс `MessageBox` содержит метод `Show` для вывода на экран окна с сообщением
- Окно может содержать несколько кнопок для получения подтверждения или ответа

Пример простого сообщения

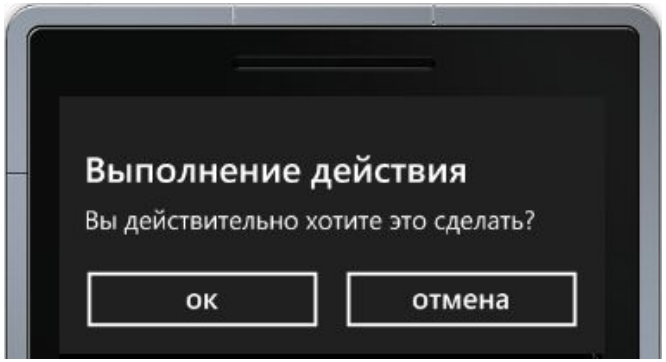
```
MessageBox.Show("Недопустимый ввод"
    + System.Environment.NewLine +
    "Пожалуйста, повторите ввод");
```

- Можно создать многострочное сообщение с помощью свойства NewLine
- Метод Show выполняется до тех пор, пока пользователь не нажмёт кнопку ok



Сообщение с выбором

```
if (MessageBox.Show(
     "Вы действительно хотите это сделать?",
     "Выполнение действия", MessageBoxButton.OKCancel)
     == MessageBoxResult.OK)
{
// Действия, если пользователь
// нажмёт "ok"
}
else
{
// Действия, если пользователь нажмёт "отмена"
}
```



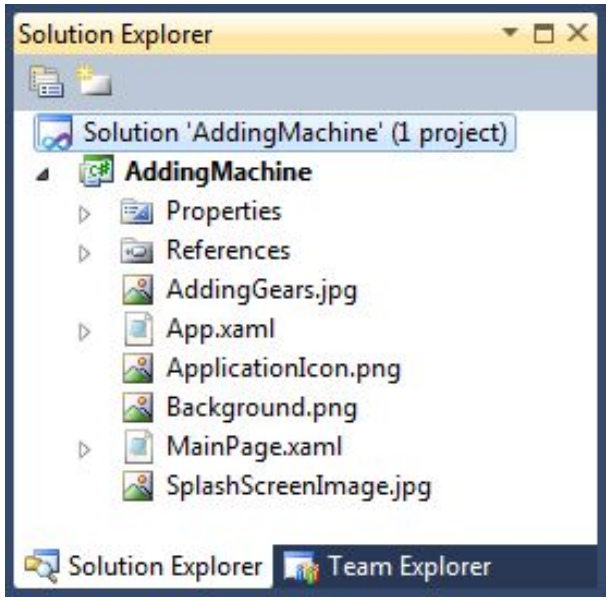
- Можно использовать возвращаемое методом значение для управления поведением программы

Использование ресурсов

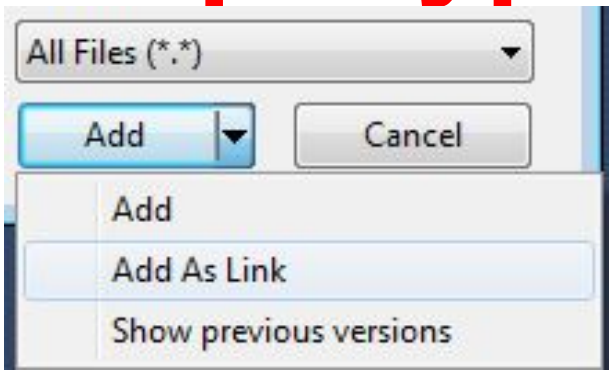
- В приложении можно использовать различные ресурсы, например, изображения и звуки
- При использовании изображений в приложении важно помнить, что целевое устройство не содержит большой объём памяти или экран высокого разрешения
- Самое высокое разрешение для Windows Phone составляет 800×480 пикселей

Добавление ресурса

- Файл ресурса можно добавить, перетащив его
  в окно обозревателя решений
- В текущем проекте будет создана копия файла,
  на которую будет установлена ссылка



Ссылка на ресурс



- При добавлении ссылки на файл ресурса не создаётся копия этого файла
- Ссылка на ресурс используется, если один и тот же файл ресурса должен использоваться сразу в нескольких проектах

Действие при построении

- При добавлении ресурса в проект можно настроить способ использования
  в свойстве Действие при построении
- Если свойству присвоено значение Содержание, ресурс копируется в каталог
  с исполняемым файлом программы
- Если свойству присвоено значение Внедренный ресурс, ресурс будет располагаться в сборке программы

Использование файла ресурса

```
<Image Height="611" HorizontalAlignment="Left"
Margin="8,0,0,0" Name="gearBackgroundImage"
Stretch="Fill" VerticalAlignment="Top" Width="472"
Source="AddingGears.jpg" />
```

- Silverlight предоставляет элемент `Image` для вывода изображений на экран
- Элемент использует атрибут `Source` для указания файла с изображением
- Silverlight выводит элементы в порядке их описания в XAML-коде

Использование внедрённого

```
<Image Height="611" HorizontalAlignment="Left"
Name="gearBackgrounds" Stretch="Fill"
VerticalAlignment="Top" Width="472"
Source="/AddingMachine;component/Images/AddingGears.jpg"
/>
```

- В этом коде используется изображение, которое содержится в файле сборки
- Изображение выглядит в приложении точно так же, как и изображение, которое хранится в

Выбор способа размещения ресурсов

- Ресурсы как часть содержимого
  - уменьшается размер файла программы
  - программа быстрее запускается, но чуть дольше загружает ресурс
- Внедрённые ресурсы
  - увеличивается размер программы
  - программа дольше запускается, но быстрее загружает ресурс
  - не используются дополнительные файлы

Краткие итоги

- У элементов Silverlight много свойств, которыми можно управлять в программе
- Инициализацию свойств лучше выполнять в XAML-описании элементов
- Свойства описываются в виде атрибутов и элементов XAML
- Элемент `MessageBox` используется для получения реакции пользователя

Windows Phone

Изменение и отображение данных

Раздел 4.2

Темы раздела

- Генерация событий элементами Silverlight
- Использование привязки данных для связи элементов Silverlight с классами приложений
  - добавление классов на страницу
  - однонаправленная привязка данных
  - двунаправленная привязка данных

События элементов

```
<Button Content="equals" Height="72"
HorizontalAlignment="Left" Margin="158,275,0,0"
Name="equalsButton" VerticalAlignment="Top" Width="160"
Click="equalsButton_Click" />
```

```
private void equalsButton_Click(
                            object sender, RoutedEventArgs e)
{
    calculateResult();
}
```

- Элементы Silverlight связываются с обработчиками страницы в коде приложения

Событие TextChanged

- Элемент `Button` генерирует событие `Click` при нажатии на кнопку
- Элемент `TextBox` генерирует событие `TextChanged`, когда пользователь вводит текст
- При использовании этого события можно обойтись без необходимости нажатия
  на кнопку

Автоматическое вычисление результата

```
private void firstNumberTextBox_TextChanged(
                    object sender,TextChangedEventArgs e)
{
 if (firstNumberTextBox.Text == oldFirstNumber) return;
    oldFirstNumber = firstNumberTextBox.Text;

    calculateResult();
}
```

- Каждый раз при вводе текста в элемент TextBox происходит автоматическое вычисление результата

Привязка данных

- Привязка данных позволяет связывать данные в программе с элементами пользовательского интерфейса
- Существует два вида привязки данных
  - однонаправленная
  - двунаправленная

Однонаправленная привязка

- Связывает свойство визуального объекта со свойством класса C#
- При изменении свойства класса также изменяется связанное свойство визуального элемента
- В программе Сумматор можно связать вычисляемый результат с визуальным элементом, в котором отображается его значение

Двунаправленная привязка

- Этот вид привязки работает в двух направлениях
  - изменение визуального элемента вызывает изменение в связанном классе C#
  - изменение свойств класса C# вызывает обновление связанного визуального элемента на экране
- Такую привязку можно использовать в программе Сумматор для ввода чисел

Создание класса для привязки

- Необходимо создать класс, который инкапсулирует поведение программы Сумматор
- Класс будет содержать свойства, связанные с визуальными элементами
  - текст верхнего элемента `TextBox`
  - текст нижнего элемента `TextBox`
  - текст элемента `TextBlock`

Создания объекта для привязки

```
public class AdderClass
{
    private int topValue;
    public int TopValue
    {
        get { return topValue; }
        set { topValue = value; }
    }
    // то же самое для свойства BottomValue

    public int AnswerValue
    {
        get { return topValue + bottomValue;}
    }
}
```

Добавление уведомления

```
public interface INotifyPropertyChanged
{
    // Событие происходит при изменении значений свойства
    event PropertyChangedEventHandler PropertyChanged;
}
```

- Для того чтобы класс можно было связать
  с визуальным элементом, в нём должен быть реализован интерфейс `INotifyPropertyChanged`

Визуальные элементы Silverlight

```
PropertyChanged(this,
                new PropertyChangedEventArgs("AnswerValue"));
```

- Если визуальный элемент Silverlight должен быть уведомлён об изменении свойства, он должен быть связан с событием `PropertyChanged`
- В этом коде Silverlight уведомляется об изменении значения свойства `AnswerValue`

Определение значения свойства

```
public int AnswerValue
{
    get
    {
        return topValue + bottomValue;
    }
}
```

- Когда элемент Silverlight считывает значение свойства, происходит вычисление и возврат результата
- Значение результата автоматически отображается на экране

Связывание класса в XAML-коде

```
xmlns:local="clr-namespace:AddingMachine"
```

- Класс C# должен быть связан с XAML-кодом страницы перед его использованием
- Необходимо добавить пространство имён, в котором описан класс, который должен быть доступен элементам на странице
- Файл XAML содержит список используемых пространств имён

Связывание с классом-ресурсом

```
<phone:PhoneApplicationPage.Resources>
    <local:AdderClass x:Key="AdderClass" />
</phone:PhoneApplicationPage.Resources>
```

- После добавления пространства имён нужно объявить имя класса, который описан в этом пространстве имён и используется в качестве ресурса

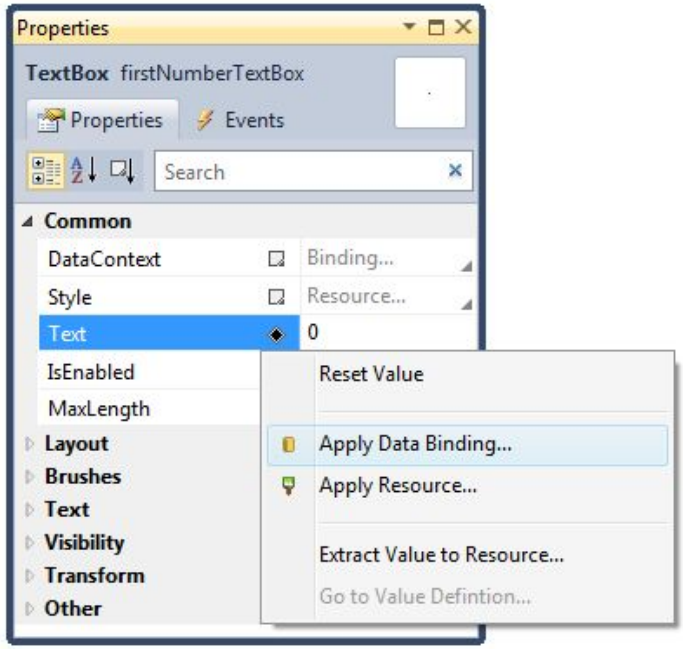
Добавление ресурса к элементу

```
<Grid x:Name="LayoutRoot" Background="Transparent"
DataContext="{StaticResource AdderClass}">
```

- Элемент `Grid` содержит все элементы страницы
- Указание объекта в качестве значения свойства `DataContext` делает его доступным для всех элементов, которые описаны внутри элемента

Применение привязки данных

- Привязка к свойствам выполняется в области свойств Visual Studio
- После выбора пункта Применить привязку данных… можно выбрать необходимое свойство и тип привязки



Привязка данных в XAML-коде

```
<TextBox Height="72" HorizontalAlignment="Left"
Margin="8,19,0,0" Name="firstNumberTextBox" Text="{Binding
TopValue, Mode=TwoWay}" VerticalAlignment="Top"
Width="460" TextAlignment="Center" >
```

- Редактирование кода XAML является более простым способом задания привязки данных к элементам на странице
- Можно указать свойство для привязки и присвоить значению свойства визуального элемента DataContext объект привязки

Установка свойства DataContext

```
 // Конструктор
public MainPage()
{
    InitializeComponent();

    AdderClass adder = new AdderClass();
    ContentGrid.DataContext = adder;
}
```

- Элемент ContentGrid является контейнером для других элементов
- При присваивании свойству DataContext экземпляра класса AddClass выполняется привязка свойств этого объекта

Краткие итоги

- Элементы Silverlight могут генерировать события, с которыми можно связать код C#
- Свойства элементов Silverlight могут быть связаны со свойствами классов C#
- Привязка может быть однонаправленной и двунаправленной
- Тип привязки и свойства можно задать

Windows Phone

Управление ориентацией страницы приложения

Раздел 4.3

Краткие итоги

- Альбомная и книжная ориентация
- Событие `OrientationChanged`
- Использование контейнеров для группировки элементов
- Элемент `StackPanel`

Ориентация в Windows Phone

- В отличие от настольных компьютеров устройство Windows Phone может использоваться два вида ориентации:
  - книжная — горизонтальное расположение
  - альбомная — вертикальное расположение
- Некоторые приложения могут корректно работать в обоих режимах ориентации

Выбор ориентации

```
SupportedOrientations="Portrait" Orientation="Portrait"
```

- Тип ориентации приложения для Windows Phone указывается в XAML-файле страницы в атрибутах `SupportedOrientations` и `Orientation`
- По умолчанию приложение использует книжную ориентацию

Режим нескольких

```
SupportedOrientations="PortraitOrLandscape“
Orientation="Portrait"
```

- С этими настройками страница работает
  в обоих режимах ориентации
- Начальная ориентация — книжная
- При повороте телефона программа попытается перерисовать дизайн
  в соответствии с ориентацией устройства

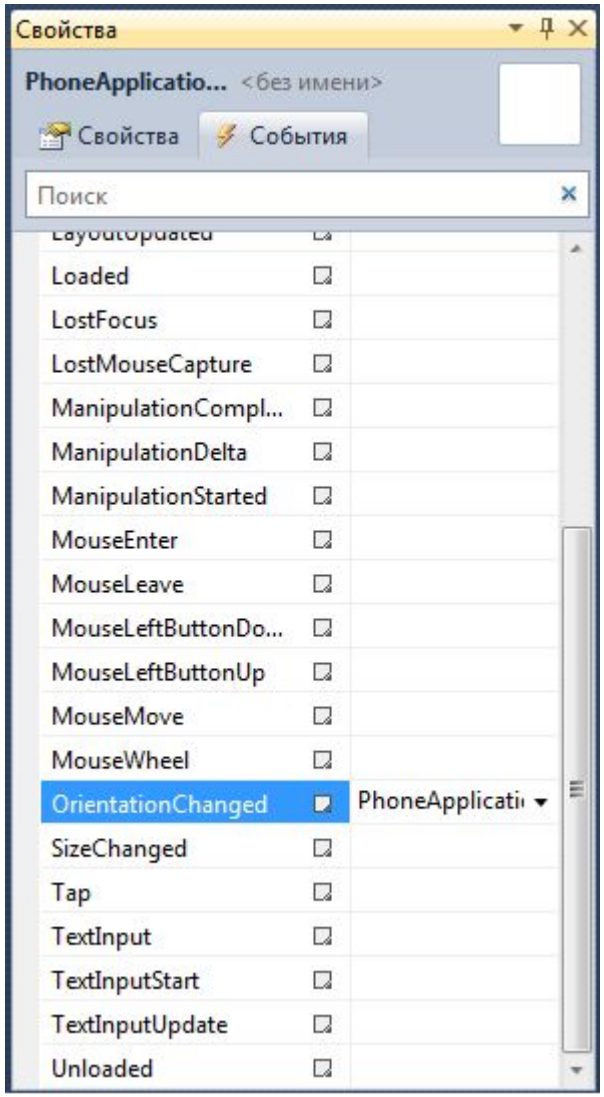
Расположение элементов

```
<TextBox Height="72" HorizontalAlignment="Left"
Margin="8,19,0,0“ Name="firstNumberTextBox" Text="0"
VerticalAlignment="Top" Width="460"
TextAlignment="Center" />
```

- Система Silverlight использует координаты для определения позиции элементов
- При перемещении элементов в дизайнере Visual Studio для них задаётся отступ от границ экрана
- При изменении ориентации эти

Событие OrientationChanged

- Приложение может получить уведомление об изменении ориентации страницы
- При возникновении события изменения ориентации телефона приложение может изменить расположение элементов



Поддержка смены

```
private void PhoneApplicationPage_OrientationChanged(
            object sender, OrientationChangedEventArgs e)
{
    if (e.Orientation == PageOrientation.PortraitUp) {
        setPortrait();
    }
    else {
        setLandscape();
    }
}
```

- Этот метод запускается при изменении ориентации телефона и вызывает подходящий метод `set`

Метод setLandscape

```
private void setLandscape()
{
    firstNumberTextBox.Margin = new Thickness(8,19,0,0);
    firstNumberTextBox.Width = 207;
    secondNumberTextBox.Margin = new Thickness(252,19,0,0);
    secondNumberTextBox.Width = 207;
    plusTextBlock.Margin = new Thickness(221,35,0,0);
    resultTextBlock.Margin = new Thickness(538,35,0,0);
}
```

- Этот метод настраивает элементы для альбомной ориентации
- Класс `Thickness` содержит 4 значения
  - координаты элемента и ширина границ

Использование контейнеров

- Использование отступов от границ удобно использовать в приложениях, поддерживающих только один режим ориентации
- В Silverlight есть элементы-контейнеры, которые могут содержать другие элементы и автоматически располагать их на экране
- Контейнеры можно эффективно использовать в приложениях,

Элемент-контейнер StackPanel

- Элемент `StackPanel` может содержать несколько текстовых элементов
- Вложенные элементы располагаются в определённой последовательности
- Последовательность элементов может быть горизонтальной или вертикальной
- Можно вкладывать один или несколько элементов `StackPanel` в другой элемент `StackPanel`

Использование StackPanel

```
<StackPanel>
    <TextBox InputScope="Digits" Height="72"
             HorizontalAlignment="Center" ... />
    <TextBlock Height="56" HorizontalAlignment="Center"
       Name="plusTextBlock"
       Text="+" ... />
    <TextBox InputScope="Digits" Height="72“
       HorizontalAlignment="Center"
       Name="secondNumberTextBox" ... />
    <TextBlock Height="46" HorizontalAlignment="Center"
       Name="resultTextBlock" ... />
</StackPanel>
```

- Элемент StackPanel выводит вложенные элементы в том порядке, в каком они описываются в XAML-коде

Краткие итоги

- Приложения Windows Phone могут работать в альбомной и/или книжной ориентации
- Можно установить поддерживаемый тип ориентации отдельно для каждой страницы
- Приложения используют событие изменения ориентации устройства
- Контейнеры автоматически располагают элементы при изменении ориентации

Windows Phone

Отображение списков данных

Раздел 4.4

Темы раздела

- Создание списков данных
- Использование элемента `StackPanel` для вывода списка на экран
- Использование элемента `ListBox` для отображения списка элементов
- Использование шаблонов данных
- События выбора элемента

Списки данных в приложениях

- Часто в приложениях необходимо выводить списки каких-либо объектов
- Для этого необходимы специальные средства, предусматривающие отображение списочных данных
- Чтобы отобразить список данных в приложении требуется подготовить содержание элементов списка

Класс Customer

```
public class Customer
{
    public string Name { get; set; }
    public string Address { get; set; }
    public int ID { get; set; }

    public Customer(string inName, string inAddress,
                                                int inID)
    {
        Name = inName;
        Address = inAddress;
        ID = inID;
    }
}
```

- Класс `Customer` будет содержать данные об одном клиенте

Класс Customers

```
public class Customers
{
    public string Name { get; set; }

    public Customers(string inName)
    {
        Name = inName;
        CustomerList = new List<Customer>();
    }

    public List<Customer> CustomerList;

}
```

- Класс Customers содержит список клиентов

Подготовка тестовых

```
public static Customers MakeTestCustomers()
{
    int id = 0;
    foreach (string lastName in lastsNames) {
        foreach (string firstname in firstNames) {
            // формирование имени клиента
            string name = firstname + " " + lastName;
            // добавление клиента в список
            result.CustomerList.Add(new Customer(name,
                                    name + "'s House", id));
            id++;
        }
    }
    return result;
}
```

```
customers = Customers.MakeTestCustomers();
```

Элемент StackPanel

- Для вывода списка на экран можно создать элемент `StackPanel`, в который поместить все элементы списка
- Silverlight позволяет создавать визуальные элементы во время выполнения программы
- Для каждого элемента списка можно создать элементы `TextBlock`, в которые занести выводимую на экран информацию

Тестовые данные

```
<StackPanel HorizontalAlignment="Left"
  Margin="0,0,0,0" Name="customersStackPanel"
  VerticalAlignment="Top"/>
```

```
foreach (Customer c in customers.CustomerList)
{
    TextBlock customerBlock = new TextBlock();
    customerBlock.Text = c.Name;
    customersStackPanel.Children.Add(customerBlock);
}
```

- В свойство `Children` элемента `customersStackPanel` добавляются элементы списка, выводимые на экран

Отображение списка на экране

- В элементе `StackPanel` можно разместить необходимое количество элементов
- При большом количестве элементов списка некоторые элементы могут не помещаться на экране

Элемент ScrollViewer

```
<ScrollViewer>
    <StackPanel HorizontalAlignment="Left“
      Margin="0,0,0,0" Name="customersStackPanel“
      VerticalAlignment="Top" />
</ScrollViewer>
```

- Элемент `ScrollViewer` также может содержать другие элементы
- Этот элемент использует полосы прокрутки, если вложенные элементы
  не помещаются на экран

Элемент ListBox

- Элемент `ListBox` специально создан для вывода списков элементов
- Элемент использует привязку данных
  для связи списка с данными
- Связывание с данными обычно более предпочтительно, поскольку избавляет
  от необходимости писать код для загрузки элементов списка

Создание шаблона данных

```
<DataTemplate>
    <StackPanel>
        <TextBlock Text="{Binding Name}"/>
        <TextBlock Text="{Binding Address}"/>
    </StackPanel>
</DataTemplate>
```

- Шаблон данных `DataTemplate` определяет, как будут выглядеть элементы списка на экране
- В этом коде каждый элемент будет выводиться в блоках `TextBlock` внутри элемента `StackPanel`

Использование шаблона

```
<ListBox Name="customerList">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel>
                <TextBlock Text="{Binding Name}"/>
                <TextBlock Text="{Binding Address}"/>
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

- Это пример XAML-кода списка элементов
- Элемент `customerList` выводит список, состоящий из значений свойств `Name` и `Address` связанных

Указание источника

```
customers = Customers.MakeTestCustomers();
customerList.ItemSource = customers.CustomerList;
```

- Данные для списка задаются в свойстве `ItemSource` элемента `ListBox`
- Элемент `ListBox` создаёт для каждого элемента списка визуальные элементы и помещает в них информацию в соответствии с заданным шаблоном данных

Вывод списка в элементе ListBox

- После задания шаблона данных и указания списка элементов в свойство `ItemSource` элемент `ListBox` будет выводить список на экран
- Элемент `ListBox` при необходимости выводит полосы прокрутки

Улучшенный шаблон

```
<DataTemplate>
   <StackPanel>
      <TextBlock Text="{Binding Name}"
       Style="{StaticResource PhoneTextExtraLargeStyle}"/>
      <TextBlock Text="{Binding Address}"
       Style="{StaticResource PhoneTextSubtleStyle}"/>
   </StackPanel>
</DataTemplate>
```

- Этот шаблон данных задаёт разные стили для элементов, выводящих имя и адрес
- Для этого используются встроенные в Windows Phone стили

Вывод списка с новым шаблоном

- При использовании улучшенного шаблона данных элементы списка удобнее отличать друг от друга
- Шаблоны данных также позволяют использовать изображения, цветной фон и анимацию

Выбор элементов в ListBox

- В программу можно добавить возможность выбора пользователем одного элемента из списка
- Приложение должно отображать и редактировать подробную информацию о выбранном клиенте
- Элемент `ListBox` позволяет легко добавить такую возможность

Событие SelectionChanged

```
<ListBox Name="customerList"
         SelectionChanged="customerList_SelectionChanged">
```

- Элемент `ListBox` может генерировать событие `SelectionChanged`
- Оно происходит, когда пользователь выбирает элемент из списка `ListBox`
- Это событие можно использовать в программе для выполнения необходимых действий при выборе клиента

Событие SelectionChanged

```
private void customerList_SelectionChanged(object sender,
                                SelectionChangedEventArgs e)
{
    // получение содержимого выбранного элемента
    Customer selectedCustomer =
                    customerList.SelectedItem as Customer;

    MessageBox.Show("Выбран элемент " +
                    selectedCustomer.Name);
}
```

- В состав элемента `ListBox` входит свойство `SelectedItem`, которое содержит ссылку на выбранный элемент

Краткие итоги

- Программы Silverlight могут создавать визуальные элементы во время работы
- Если элемент не помещается на экране, его можно поместить в элемент `ScrollViewer`, который использует полосы прокрутки
- Элемент `ListBox` может выводить элементы в соответствии с заданным шаблоном данных
- Элемент `ListBox` может

Windows Phone

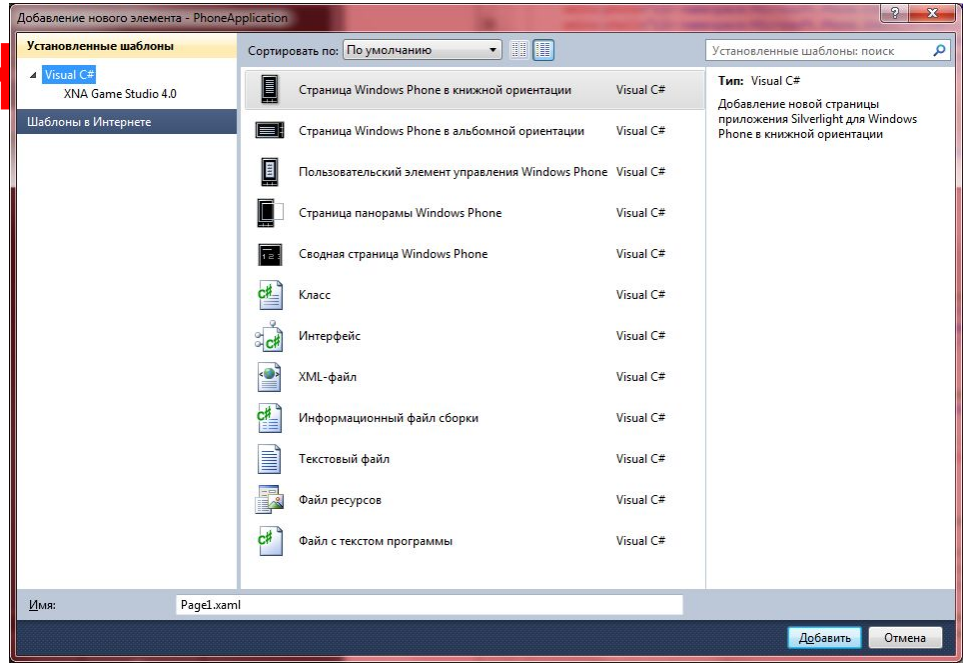
Навигация по страницам приложения

Раздел 4.5

Темы раздела

- Добавление новой страницы в приложение
- Навигация по страницам
- Передача данных между страницами
- Использование событий при навигации по страницам
- Совместное использование объектов несколькими страницами

Добавление новой стран



- Новую страницу можно добавить в проект так же, как и другие объекты
- При этом создаются файлы с кодом XAML и C#

Навигация по страницам

```
private void page2Button_Click(object sender,
                               RoutedEventArgs e)
{
    NavigationService.Navigate(
                    new Uri("/CustomerDetailPage.xaml",
                        UriKind.RelativeOrAbsolute));
}
```

- Объект NavigationService выполняет перемещение между страницами
- У каждой страницы Silverlight есть свой URI
- Метод Navigate осуществляет

Типы URI

```
private void page2Button_Click(object sender,
                               RoutedEventArgs e)
{
    NavigationService.Navigate(
                    new Uri("/CustomerDetailPage.xaml",
                        UriKind.RelativeOrAbsolute));
}
```

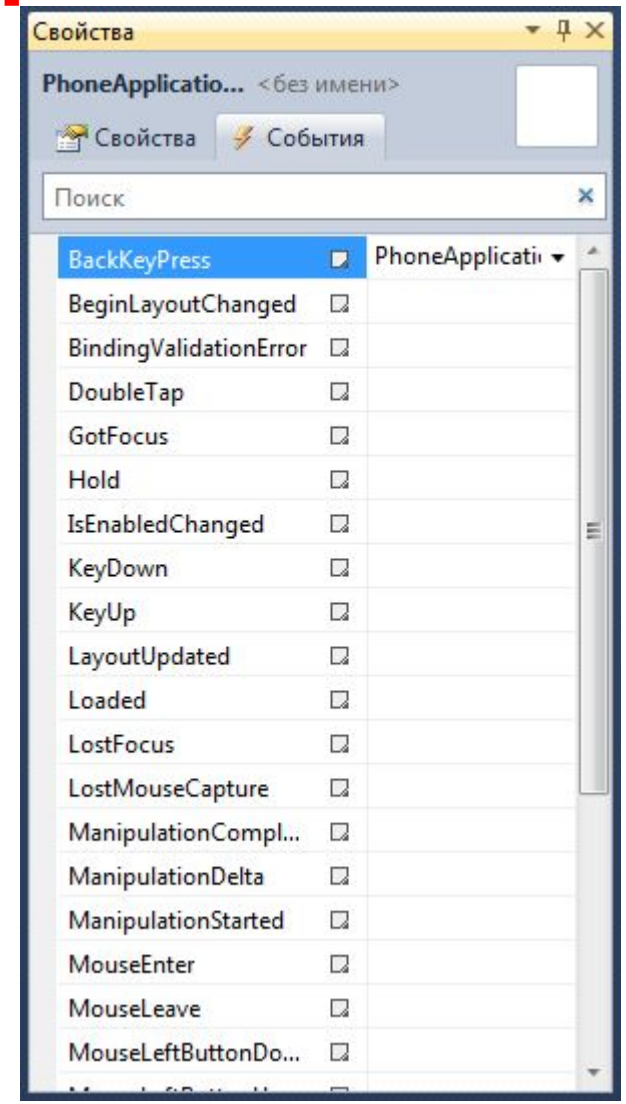
- Адрес страницы может быть задан абсолютно или относительно текущего расположения
- Значение `RelativeOrAbsolute` часто применяется вместо значения `Relative`

Использование кнопки Назад

- Кнопка Назад используется в Windows Phone для перехода к предыдущей странице
- Это поведение используется при навигации по страницам Silverlight
- При нажатии на кнопку Назад происходит переход к предыдущей странице приложения или завершение работы программы

Изменение поведения кнопки

- Часто необходимо переопределить действие кнопки Назад по умолчанию
- Можно создать обработчик события нажатия на кнопку Назад и указать необходимые действия



Отмена перехода

```
private void PhoneApplicationPage_BackKeyPress(
                    object sender,
                    System.ComponentModel.CancelEventArgs e)
{
    e.Cancel = true;
}
```

- Этот обработчик события, связанный с событием нажатия на кнопку Назад, отменяет переход на предыдущую страницу

Использование MessageBox

```
private void PhoneApplicationPage_BackKeyPress(
                    object sender,
                    System.ComponentModel.CancelEventArgs e)
{
    if (MessageBox.Show(
    "Вы действительно хотите перейти на другую страницу?",
        "Подтверждение перехода",
        MessageBoxButton.OKCancel) != MessageBoxResult.OK)
    {
        e.Cancel = true;
    }
}
```

- Этот код выводит диалоговое окно для подтверждения перехода

Передача данных между страницами

- Каждая страница Silverlight является независимой от других страниц
- Страница может содержать данные, которые недоступны другим страницам
- Часто необходимо передавать данные
  от одной страницы к другой
- Простые данные можно передавать в строке URI с адресом целевой

Добавление данных в URI

```
// получение информации о выбранном клиенте
Customer selectedCustomer = customerList.SelectedItem
                                as Customer;
// формирование строки адреса с информацией о клиенте
    NavigationService.Navigate(
          new Uri("/CustomerDetailPage.xaml?" +
          "name=" + selectedCustomer.Name + "&" +
          "address=" + selectedCustomer.Address,
          UriKind.Relative));
```

- Этот код добавляет в строку URI целевой страницы два параметра: name и address
- Значения параметров передаются в виде открытой строки

Использование событий при навигации по страницам

- Если целевая страница использует параметры из URI, необходимо добавить код для чтения значений параметров
- Можно переопределить методы страницы, которые вызываются при переходе на страницу или на другую страницу
  - `OnNavigatedTo`
  - `OnNavigatedFrom`

Получение данных из URI

```
protected override void OnNavigatedTo
          (System.Windows.Navigation.NavigationEventArgs e)
{
  string name, address;
  if (NavigationContext.QueryString.TryGetValue("name",
                                                out name))
      nameTextBlock.Text = name;
}
```

- В объекте NavigationContext есть свойство QueryString
- Метод TryGetValue ищет значение в URI и возвращает значение true, если такой параметр существует

Совместное использование объектов несколькими страницами

- Каждая страница в программе может хранить данные, которые недоступны другим страницам
- При переходе на страницу она не содержит ссылку на исходную страницу
- Для совместного использования данных можно использовать страницу App.xaml

Страница App.xaml

- Эта страница является главной страницей приложения
- Она содержит только методы, которые вызываются при запуске приложения
- Также на этой странице размещаются обработчики некоторых событий, которые необходимы для правильной работы приложения

Класс App

```
public partial class App : Application
{
    // объект используется другими страницами приложения
    public Customer ActiveCustomer;
}
```

- Класс App является расширением класса Silverlight `Application`
- В этот класс можно добавить свои методы
- Созданное свойство `ActiveCustomer` можно использовать в других страницах приложения

Получение ссылки на

```
protected override void OnNavigatedTo(
          System.Windows.Navigation.NavigationEventArgs e)
{
 base.OnNavigatedTo(e);
 // получение ссылки на страницу с информацией
 // о текущем клиенте
 App thisApp = Application.Current as App;

 // установка контекста данных для отображения
 // в элементе Grid
 customerDisplayGrid.DataContext = thisApp.ActiveCustomer;
}
```

- Свойство `Current` класса `Application` содержит ссылку на активную страницу приложения

Установка контекста

```
protected override void OnNavigatedTo(
          System.Windows.Navigation.NavigationEventArgs e)
{
 base.OnNavigatedTo(e);
 // получение ссылки на страницу с информацией
 // о текущем клиенте
 App thisApp = Application.Current as App;

 // установка контекста данных для отображения
 // в элементе Grid
 customerDisplayGrid.DataContext = thisApp.ActiveCustomer;
}
```

- Здесь задаётся контекст данных для отображения информации о выбранном клиенте

Краткие итоги

- Приложение Silverlight может состоять
  из нескольких страниц
- Навигация по страницам выполняется
  с помощью URI, который может содержать строковые параметры
- При необходимости можно переопределить поведение кнопки Назад
- Можно предоставить совместный

Windows Phone

Использование классов ViewModel

Раздел 4.6

Темы раздела

- Классы ViewModel и дизайн интерфейса
- Использование шаблона Model –View–ViewModel
- Создание класса ViewModel
- Передача данных класса ViewModel
- Использование наблюдаемых коллекций

Отображение изменений

- Когда пользователь изменяет информацию о клиенте, она должна обновляться при нажатии на кнопку сохранить
- При нажатии на кнопку отмена (или Назад) все изменения должны отменяться

Связывание с данными

- Можно связать информацию о клиенте
  с визуальными элементами на форме
- При изменении значений элементов связанные данные изменяются сразу же
- Это не очень удобный путь построения приложения
  - информация обновляется до окончания редактирования

Model-View-ViewModel

- Класс ViewModel связывает данные (класс `Customer`) и элементы интерфейса (класс `CustomerDetailPage`)
- Класс содержит все события изменения данных и при необходимости выполняет их передачу и проверку
- Информация о клиенте загружается в класс при открытии формы для редактирования и сохраняется при

Класс CustomerView

```
public class CustomerView : INotifyPropertyChanged
{
  private string name;
  public string Name
  {
    get { return name; }
    set {
        name = value;
        if (PropertyChanged != null) {
          PropertyChanged(this,
                new PropertyChangedEventArgs("name"));
      }
    }
  }
  ...
}
```

Класс CustomerView

- Класс `CustomerView` связывает свойства класса `Customer` с визуальными элементами `TextBox`
- Он генерирует события, необходимые для осуществления двунаправленной привязки данных
- Класс также содержит методы для загрузки и сохранения объекта `Customer`

Методы Load и Save

```
public class CustomerView : INotifyPropertyChanged
{
    ...
    public void Load(Customer cust)
    {
        Name = cust.Name;
        Address = cust.Address;
        id = cust.ID;
    }
    public void Save(Customer cust)
    {
        cust.Name = Name;
        cust.Address = Address;
    }
}
```

Начало редактирования

```
protected override void OnNavigatedTo(
          System.Windows.Navigation.NavigationEventArgs e)
{
    // получить ссылку на страницу, содержащую информацию
    // об активном клиенте
    App thisApp = Application.Current as App;

    // загрузить объект с информацией об активном клиенте
    // в класс ViewModel
    view.Load(thisApp.ActiveCustomer);

    // установить контекст данных для отображения
    customerDisplayGrid.DataContext = view;
}
```

Окончание

```
private void saveButton_Click(object sender,
                                        RoutedEventArgs e)
{
    // получить ссылку на страницу, содержащую информацию
    // об активном клиенте
    App thisApp = Application.Current as App;

    // скопировать данные из класса ViewModel в свойство
    // ActiveCustomer
    view.Save(thisApp.ActiveCustomer);

    // вернуться на предыдущую страницу
    NavigationService.GoBack();
}
```

Метод GoBack

```
private void saveButton_Click(object sender,
                                        RoutedEventArgs e)
{
    ...
    // вернуться на предыдущую страницу
    NavigationService.GoBack();
}
```

- Метод GoBack осуществляет переход к предыдущей странице
- Такое же действие выполняется при нажатии на кнопку Назад

Наблюдаемые коллекции

- Класс ViewModel является «наблюдаемым»
  - можно использовать события, происходящие при изменении данных
- Для отображения изменений элементов
  в списке нужно использовать другой тип коллекции
- Наблюдаемые коллекции могут генерировать события, которые может использовать элемент

Класс ObservableCollection

```
ObservableCollection<Customer> observableCustomers;

observableCustomers = new ObservableCollection<Customer>
                (thisApp.ActiveCustomerList.CustomerList);

// отображение элементов наблюдаемой коллекции
customerList.ItemsSource = observableCustomers;
```

- Этот код создаёт наблюдаемую коллекцию на основе списка клиентов
- Эта коллекция устанавливается в качестве значения свойства `ItemsSource` визуального элемента

Подтверждение изменений

```
// найти нужного клиента в списке
int pos = observableCustomers.IndexOf(
                                    thisApp.ActiveCustomer);
// удалить клиента
observableCustomers.RemoveAt(pos);
// вернуть клиента назад
observableCustomers.Insert(pos, thisApp.ActiveCustomer);
```

- Класс `ObservableCollection` не реагирует на изменение данных в элементе списка
- Можно принудительно вызвать изменение данных, удалив элемент списка и вернув его назад

Сохранение данных

```
thisApp.ActiveCustomerList.CustomerList =
                        observableCustomers.ToList<Customer>();
```

- После окончания редактирования списка клиентов необходимо получить обновлённые данные из класса `ObservableCollection`
- Класс предоставляет несколько возможностей извлечения данных
- Приведённый код возвращает список клиентов

Краткие итоги

- Класс ViewModel содержит свойства данных, которые копируются из источника и возвращаются после редактирования
- Класс ViewModel реагирует на события пользовательского интерфейса и выполняет проверку и преобразование данных
- Класс `ObservableCollection` Windows Phone предоставляет возможности ViewModel для