МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**Нижегородский государственный университет**

**им. Н. И. Лобачевского**

Факультет Вычислительной математики и кибернетики

Кафедра «Математического обеспечения ЭВМ»

**Н.А.Борисов**

**Методические указания**

к выполнению лабораторных работ по спецкурсу

«Программирование мобильных устройств»

Рекомендовано методической комиссией факультета ВМК для студентов ННГУ, обучающихся по направлениям подготовки 010500 «Прикладная математика и информатика», 010400 «Информационные технологии»

Нижний Новгород

2013

УДК 004.41, 004.42

ББК 32.973-018.2

Борисов Н.А. МЕТОДИЧЕСКИЕ УКАЗАНИЯК ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО КУРСУ «ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ УСТРОЙСТВ» : Н.Новгород: ННГУ, 2013 – 64 с.

Рецензент:

к.ф.-м.н., доц. каф. численного и функционального анализа Кузенков О.А

Методические указания к выполнению лабораторных работ по курсу «Программирование мобильных устройств» содержат рекомендации по использованию интегрированной среды MS Visual Studio для разработки программ для мобильных устройств, работающих под управлением операционной системы Windows Phone. При разработке программ используются платформы Silverlight и XNA.

Рекомендовано методической комиссией факультета ВМК для студентов ННГУ, обучающихся по направлениям подготовки 010500 «Прикладная математика и информатика», 010400 «Информационные технологии» и специальностям 010501 «Прикладная математика и информатика», 080801 «Прикладная информатика»

УДК 004.41, 004.42

ББК 32.973-018.2

Нижегородский государственный университет

имени Н. И. Лобачевского, 2013

Содержание

[Лабораторная работа № 1. Разработка программы Silverlight для Windows 5](#_Toc358109714)

[1. Проектирование программы Silverlight 5](#_Toc358109715)

[1.1. Средства разработки 5](#_Toc358109716)

[1.2. Metro-стиль 5](#_Toc358109717)

[1.3. Элементы и объекты Silverlight 6](#_Toc358109718)

[1.4. Панель инструментов и графический конструктор 8](#_Toc358109719)

[1.5. Управление именами элементов в Visual Studio 9](#_Toc358109720)

[1.6. Свойства в элементах Silverlight 9](#_Toc358109721)

[1.7. Создание методов get и set 10](#_Toc358109722)

[1.8. Использование свойств 11](#_Toc358109723)

[1.9. Дизайн страницы на Silverlight 13](#_Toc358109724)

[2. Язык XAML 13](#_Toc358109725)

[3. Пример создания приложения Silverlight для Windows Phone 16](#_Toc358109726)

[3.1. Создание приложения 17](#_Toc358109727)

[3.2. Вычисление результата 18](#_Toc358109728)

[3.3. События и программы 18](#_Toc358109729)

[3.4. События в Silverlight 19](#_Toc358109730)

[3.5. Управление свойствами событий 20](#_Toc358109731)

[4. Краткие итоги 21](#_Toc358109732)

[5. Упражнения 21](#_Toc358109733)

[Упражнение 1. Пользовательский интерфейс программы Калькулятор времени 21](#_Toc358109734)

[Упражнение 2. Синхронизация данных калькулятора времени 22](#_Toc358109735)

[6. Справочный материал 23](#_Toc358109736)

[6.1. Среда разработки 23](#_Toc358109737)

[6.2. Шаблоны приложений 23](#_Toc358109738)

[6.3. Структура простого приложения 24](#_Toc358109739)

[6.4. Основные элементы управления 25](#_Toc358109740)

[7. Дополнительные материалы 26](#_Toc358109741)

[Лабораторная работа № 2. Модернизация программы Silverlight для Windows Phone 27](#_Toc358109742)

[1. Улучшение приложения 27](#_Toc358109743)

[1.1. Управление свойствами элементов 27](#_Toc358109744)

[1.2. Редактирование XAML-кода элементов Silverlight 29](#_Toc358109745)

[1.3. Вывод окна с сообщением 31](#_Toc358109746)

[1.4. Добавление и использование ресурсов 32](#_Toc358109747)

[2. Изменение и отображение данных 34](#_Toc358109748)

[1.1. Событие TextChanged 34](#_Toc358109749)

[1.2. Привязка данных 36](#_Toc358109750)

[1.3. Привязка данных через свойство DataContext 40](#_Toc358109751)

[3. Управление ориентацией страницы приложения 41](#_Toc358109752)

[1.1. Альбомная и книжная ориентация 41](#_Toc358109753)

[1.2. Использование контейнеров для группировки элементов 44](#_Toc358109754)

[4. Отображение списков данных 45](#_Toc358109755)

[4.1. Создание данных для списка 46](#_Toc358109756)

[4.2. Использование элемента StackPanel для вывода списка на экран 47](#_Toc358109757)

[4.3. Использование ListBox для отображения списка элементов 49](#_Toc358109758)

[5. Навигация по страницам приложения 53](#_Toc358109759)

[5.1. Навигация по страницам 53](#_Toc358109760)

[5.2. Передача данных между страницами 54](#_Toc358109761)

[5.3. Совместное использование объектов несколькими страницами 56](#_Toc358109762)

[6. Использование классов ViewModel 57](#_Toc358109763)

[6.1. Создание класса ViewModel 58](#_Toc358109764)

[6.2. Навигация по страницам с использованием метода GoBack 60](#_Toc358109765)

[6.1. Наблюдаемые коллекции 61](#_Toc358109766)

[7. Краткие итоги 63](#_Toc358109767)

# Лабораторная работа № 1. Разработка программы Silverlight для Windows

## Проектирование программы Silverlight

Не всегда программисты при создании приложений занимаются созданием *интерфейса*. Часто компании поручают эту задачу дизайнерам. Программисты же должны добавить к дизайну программный код, который выполнит необходимую работу. Silverlight позволяет разделить процесс создания *интерфейса* и процесс создания программного кода приложения. Это позволяет программисту набросать начальный пользовательский *интерфейс*, который впоследствии может изменить дизайнер, сделав его более привлекательным. Программист также может взять законченный дизайн пользовательского *интерфейса* и затем запрограммировать поведение каждого элемента.

### Средства разработки

Проектировщики пользовательского интерфейса используют инструменты Expression Blend, а программисты — Visual Studio для создания кода. Система Silverlight предоставляет возможность объединить дизайн пользовательского интерфейса и код. Хороший подходом для программиста является использование шаблона "заполнителя", который позволяет запрограммировать поведение программы и позднее внести в приложение окончательный дизайн.

Windows Phone SDK (Software Development Kit — набор средств для разработки программного обеспечения) включает версии Visual Studio и Expression Blend, которые можно использовать для создания кода и дизайна приложения.

### Metro-стиль

С точки зрения дизайна Windows Phone использует концепцию, которая в руководствах по проектированию называется Metro-стиль. В нем заложены принципы, которые определяют, как выглядят элементы управления, и устанавливают ряд критериев, которым должны соответствовать приложения Windows Phone. Эти особенности стиля отражены во встроенных компонентах, которые могут использовать программы.

Приложения, использующие кнопки, текстовые поля и другие элементы, которые поставляются с Visual Studio, автоматически будут придерживаться требований стиля. Это большая помощь тем, кто не очень хорошо умеет проектировать дизайн. Конечно, можно полностью переопределить свойства поставляемых компонентов, но при создании приложений для Windows Phone лучше этого избегать. Существует руководство по Metro-стилю, которое доступно по адресу:<http://msdn.microsoft.com/ru-ru/library/hh202915.aspx>.

В этой работе мы будем использовать средства проектирования Silverlight в составе Visual Studio. Они не дают всех возможностей создания продвинутого графического интерфейса, которые предоставляет Expression Blend, но для простых программ этих средств их вполне достаточно.

### Элементы и объекты Silverlight

С точки зрения программирования каждый элемент на экране является программным объектом. Объект включает поведение и данные. Если бы мы создавали приложение для управления банковскими счетами, то мы могли бы создать такой класс для хранения информации о счете:

public class Account

{

 private decimal balance;

 private string name;

 public string GetName()

 {

 return name;

 }

 public bool SetName(string newName)

 {

 // в финальной версии здесь можно проверить корректность имени

 name = newName;

 return true;

 }

 // другие методы get и set

}

В этом классе хранится информация о количестве денег на банковском счете (в свойстве balance) и имя владельца счета (в свойстве name). Для создания новой записи можно использовать ключевое слово new:

Account rob = new Account();

rob.SetName("Rob");

Этот код создает новый экземпляр класса Account с именем rob. Затем задается имя владельца счета — "Rob". Использование объектов — это отличный способ представления сущностей, информацию о которых нужно хранить и обрабатывать. С помощью объектов также можно описать элементы на экране. Область отображаемого на экране текста также имеет свойства: позиция на экране, цвет текста, сам текст и др.

На [рис. 1](http://www.intuit.ru/department/se/devmobwphn/3/devmobwphn_3.html#image.3.1) представлен интерфейс простой программы Сумматор, которая находит сумму двух чисел. Числа вводятся в текстовые поля, и при нажатии на кнопку сумма результат выводится в нижнее текстовое поле.

Каждый отдельный элемент на экране относится к классу UIElement, который представляет элементы пользовательского интерфейса. Экран этого приложения содержит семь элементов:

1. Маленький заголовок СУММАТОР. Этот элемент называется название приложения (Application Title).
2. Большой заголовок сложение. Этот элемент называется название страницы (Page Title).
3. Верхнее текстовое поле, в которое можно ввести число.

1. Текстовый элемент, содержащий знак операции сложения - символ "+".
2. Нижнее текстовое поле, в которое можно ввести другое число.
3. Кнопка, при нажатии на которую происходит сложение.
4. Текстовое поле, которое при нажатии на кнопку отображает результат.

Для каждого элемента задается положение на экране, размер текста и другие свойства. Можно изменить цвет текста в текстовом поле, выровнять его по левому краю, по правому краю или по центру поля, а также задать другие свойства.

**Рис. 1.** Экран программы Сумматор

Указанные элементы имеют один из типов:

1. TextBox — позволяет пользователю вводить текст.
2. TextBlock — текстовый блок, который только отображает информацию.
3. Button — кнопка, при нажатии которой вызывается событие.

Свойства этих элементов можно разделить на два типа: свойства, которые должны быть у всех элементов, например положение на экране, и свойства, которые являются специфичными для определенного типа элемента. Например, только для элемента TextBox можно задать позицию курсора для ввода текста. Можно представить все элементы в иерархической структуре, как показано на [рис. 2](http://www.intuit.ru/department/se/devmobwphn/3/devmobwphn_3.html#image.3.2).

Такую иерархическую структуру создали проектировщики Silverlight. Класс UIElement является классом, который может отображаться на экране как визуальный объект и принимать данные, которые вводит пользователь. На телефоне ввод информации выполняется преимущественно с сенсорного экрана.

От класса UIElement наследуется класс FrameworkElement, который содержит всю информацию, характерную для всех отображаемых элементов. Все остальные классы наследуются от него. Наследники имеют все свойства родительского класса и добавляют собственные.

Такая иерархия классов имеет ряд преимуществ. Если нужен собственный тип текстового поля, можно расширить класс TextBox и добавить в него нужные свойства и методы. При этом, система Silverlight будет обрабатывать все эти элементы так же, как и обычные текстовые поля, поскольку новые элементы сохраняют все свойства и поведение исходного класса.

Необходимо помнить, что при изменении элементов на экране и создании и управлении элементами на самом деле мы изменяем свойства объектов. При проектировании пользовательского интерфейса Silverlight мы задаем значения свойствам элементов, которые используются для отображения элементов на экране.

### Панель инструментов и графический конструктор

Создать приложение для Windows Phone очень легко. При создании нового проекта Silverlight открывается пустая страница окна дизайнера, на которую можно добавлять элементы с панели инструментов. Можно создать интерфейс программы, просто перетаскивая элементы с панели инструментов в окно дизайнера.

**Рис. 2.** Структура классов элементов Silverlight

На [рис. 3](http://www.intuit.ru/department/se/devmobwphn/3/devmobwphn_3.html#image.3.3) показан результат перетаскивания элемента TextBox с панели инструментов в окно страницы Windows Phone. Процесс создания интерфейса страницы приложения для Windows Phone такой же, как и приложения Windows Forms. Поскольку все средства управления уже разработаны в стиле Metro, создаваемые приложения автоматически становятся похожими на "правильные" Metro-приложения.

**Рис. 3.** Окно дизайнера Visual Studio

### Управление именами элементов в Visual Studio

После добавления элементов на экран рекомендуется дать элементам имена. Среда проектирования даст каждому элементу имена по умолчанию, например, TextBox1, которые лучше изменить на осмысленные, подобрав каждому элементу имя на основе его назначения.

Для того чтобы изменить свойства элементов, нужно щелкнуть по элементу в окне дизайнера и затем найти нужное свойство в области свойств в Visual Studio.

На [рис. 4](http://www.intuit.ru/department/se/devmobwphn/3/devmobwphn_3.html#image.3.4) изображена область свойств для верхнего элемента TextBox на странице. Имя элемента отображается в верхней части окна. Изменим имя этого текстового поля на firstNumberTextBox. При этом, изменение значения свойства Name переопределит имя переменной, связанной с этим элементом:

TextBox firstNumberTextBox;

Visual Studio отслеживает объявления фактических переменных, которые представляют создаваемые отображаемые элементы, и нам не нужно думать о том, где хранятся соответствующие объявления этих переменных.

### Свойства в элементах Silverlight

После переименования переменной, представляющей текстовое поле, можно настроить ее свойства: ширину, отступы от краев и т.д. Значения свойств отражают текущую позицию и размер элемента на экране. Если перетащить элемент за его край, значения свойств изменятся. Если изменить значения в окне Свойства, то элемент в окне дизайнера автоматически изменит положение. Точно также происходит и при создании приложения Windows Forms. Любые изменения в окне дизайнера приводят к изменению содержимого объекта.

**Рис. 4.** Область свойств элемента TextBox

Свойства в Silverlight и свойства в C#. Когда мы говорим о свойствах элементов Silverlight на странице (например, текст в поле TextBox), мы фактически говорим о значениях свойств в классе, которые содержатся в элементе TextBox. Другими словами, если программа содержит такое объявление:

resultTextBlock.Text = "0";

запустится метод Set объекта resultTextBlock, который устанавливает текст элемента TextBlock в соответствующее значение.

Классы и свойства в C#. Классы в C# могут содержать данные (имя владельца банковского счета) и поведение (методы GetName и SetName, которые позволяют программе узнавать имя и изменить это имя). Данные в классе представляются в виде свойств, с которым работать проще, чем использовать методы Get и Set. Они активно используются в элементах управлении Silverlight.

Свойство — это член класса, который содержит значение. Можно использовать переменные для хранения значений, но нужно сделать их публичными (public), чтобы их можно было использовать в приложениях. Например, банк может захотеть отслеживать информацию о сотрудниках. Один из параметров, значение которого может понадобиться отслеживать, — возраст владельца банковского счета. Можно объявить в классе публичное поле:

public class Account

{

 public int Age;

}

Класс Account содержит общедоступное поле Age. Можно получить доступ к этому полю по его имени:

Account s = new Account();

s.Age = 21;

Проблема заключается в том, что в поле для хранения возраста можно записать недопустимое значение:

s.Age = –100210232;

Поскольку поле является общедоступным, нет возможности воспрепятствовать подобному действию.

### Создание методов get и set

Чтобы контролировать использование полей, можно создать методы get и set и сделать их общедоступными. Они предоставляют возможность управлять доступом к полю. При этом, поле Age лучше сделать закрытым (private), чтобы к нему нельзя было получить прямой доступ за пределами класса.

public class Account

{

 private int age;

 public int GetAge()

 {

 return this.age;

 }

 public void SetAge(int inAge)

 {

 if ((inAge > 0) && (inAge < 120))

 {

 this.age = inAge;

 }

 }

}

Теперь можно контролировать доступ к нашему полю, но для этого нужно написать много дополнительного кода. Для того чтобы получить доступ к значению возраста, необходимо вызвать созданные методы:

Account s = new Account();

s.SetAge(21);

Console.WriteLine("Возраст: " + s.GetAge());

### Использование свойств

Свойства позволяют сделать управление данными более простым. Свойство Age можно объявить следующим образом:

public class Account

{

 private int ageValue;

 public int Age

 {

 set

 {

 if ((value > 0) && (value < 120))

 ageValue = value;

 }

 get

 {

 return ageValue;

 }

 }

}

Здесь значение возраста является свойством. В свойстве объявлены секции для записи и чтения его значения. Описанные в этих секциях действия эквиваленты описанным ранее методам. При этом, свойства используются так же, как и обычные поля:

Account s = new Account();

s.Age = 21;

Console.WriteLine("Возраст: " + s.Age);

Когда свойству Age присваивается значение, вызывается код секции set. Ключевое слово value обозначает значение, которое присваивается свойству. При чтении значения свойства Age происходит вызов кода секции get. Такой подход сочетает преимущества использования методов и позволяет работать со свойствами так же просто, как и с полями класса.

***Проверка правильности данных в свойствах***. При попытке задать недопустимое значение возраста (например, 150), приведенный выше код выполнит проверку допустимости и отклонит это значение (никто старше 120 лет не может иметь счет в нашем банке), оставив прежнее значение возраста. Единственный способ узнать, было ли свойству присвоено значение, заключается в проверке значения свойства после этой операции:

Account s = new Account();

int newAge = 150;

s.Age = newAge;

if (s.Age != newAge)

 Console.WriteLine("Значение возраста не было установлено");

Приведенный код пытается присвоить возрасту недопустимое значение 150, после чего проверяется, было ли это значение установлено. Если бы для присваивания значения использовался метод Set, он мог бы вернуть значение ***false*** в случае неудачи, а при использовании свойства пользователь должен выполнить чуть больше дополнительной работы.

Различные способы считывания значения свойства. Свойства позволяют выполнять и другие полезные действия.

public int AgeInMonths

{

 get

 {

 return this.ageValue \* 12;

 }

}

Здесь описано новое свойство AgeInMonths. Оно предназначено только для чтения, так как не содержит секции set. Оно возвращает значение возраста в месяцах, используя то же самое значение, что и свойство Age. Это означает, что можно использовать несколько различных способов для получения одного и того же значения. Можно создать свойства только для чтения без возможности их изменения напрямую, а также свойства только для записи, хотя последние используются редко.

***Свойства визуальных элементов***. Свойства имеет смысл использовать в описании банковского счета, где нужно защитить данные в объектах. Но в программе Silverlight можно ввести любой текст в элемент TextBlock, и вроде бы нет нужды проверять допустимость вводимого значения. Выполнение этого кода замедлит процесс ввода значения. Так, сделав значение Text публичной строкой, программа содержала бы меньше кода и работала быстрее.

Но при этом, когда мы изменяем текст в элементе TextBlock, мы хотим, чтобы текст на странице Silverlight также изменился, например, когда программа Сумматор выведет на экран результат. Если бы программа просто изменила значение поля, система Silverlight никак не могла бы узнать, что сообщение на экране должно быть обновлено.

Однако, если Text сделать свойством, при обновлении значения элемента TextBlock запустится соответствующий метод, который может обновить хранимое значение текстового поля и вызвать метод для обновления экрана, чтобы отобразить новое значение. Свойства предоставляют возможность управления объектом при изменении его значения. Простая операция:

resultTextBlock.Text = "0";

может привести к выполнению нескольких сотен операций C#, поскольку сохранение нового значения в элементе TextBlock приводит к запуску операций для обновления изображения на экране.

### Дизайн страницы на Silverlight

Мы можем завершить страницу программы Сумматор, перетаскивая нужные элементы на экран и устанавливая их свойства. Первым делом после перетаскивания элемента на страницу рекомендуется определить имя этого элемента. Признаком плохого стиля программы является страница, содержащая много элементов с именами типа Button1, Button2 и т.д.

Разработка страницы является несложной задачей, которая сводится к перетаскиванию элементов на страницу и настройке свойств каждого элемента. Средства Silverlight позволяют добавить к элементам графические свойства, которые могут сделать их прозрачными, добавить изображения к их фону и даже сделать их анимированными.

## Язык XAML

Проектирование дизайна на Silverlight состоит из манипуляций с элементами с помощью дизайнера Visual Studio и присваивания значений нужным свойствам элементов. Выполняя эти действия, можно создать любой пользовательский интерфейс. интерфейс приложения на Silverlight описывается с помощью языка XAML.

XAML означает "расширяемый язык разметки приложений" (eXtensible Application Markup Language). Silverlight использует XAML для описания внешнего вида страницы.

XAML определяет четкую границу между внешним видом приложения (свойства отображаемых элементов) и поведением приложения (какие действия выполняют элементы приложения).

Такое разделение является хорошей идеей, поскольку нельзя гарантировать, что хороший программист будет хорошим дизайнером. Обычно над созданием приложений работают и дизайнеры, и программисты. При этом возникает проблема разделения работы между ними: программист не может ничего делать, пока дизайнер не закончит создание всех страниц приложения.

XAML решает эту проблему. Как только будут изложены требования пользовательского интерфейса (сколько текстовых полей, сколько кнопок, что будет на каждой странице и т.д.), проектировщик может продолжать работу над внешним видом страниц, в то время как программист может сосредоточиться на написании программного кода. Файлы XAML содержат описание элементов пользовательского интерфейса, и дизайнер может работать над их внешним видом и расположением на странице, в то время как программист дополняет их программным кодом. Программисту не нужно передавать дизайнеру файлы с кодом, так как они отделены от дизайна, и наоборот.

Содержимое файла XAML. Файл XAML для страницы содержит конструкции, которые описывают все элементы на странице. Каждое описание содержать набор свойств. Строка XAML, которая описывает элемент TextBox, может выглядеть так:

<TextBox Height="72" HorizontalAlignment="Left" Margin="8,19,0,0"

 Name="firstNumberTextBox" Text="0" VerticalAlignment="Top" Width="460"

 TextAlignment="Center" />

Если сравнить информацию в области свойств со значениями атрибутов в коде XAML, можно установить их соответствие: имя элемента TextBox — firstNumberTextBox, ширина — 460 и т.д. Страница и область свойств синхронизируют значения, так как они используют один и тот же XAML-файл страницы. При перемещении элементов по странице редактор обновляет XAML; область свойств считывает данные из этого файла и обновляет его значения.

XAML является декларативным языком. Он создан, чтобы быть понятным для людей, поэтому у всех выше указанных свойств понятные имена. Можно отредактировать текст XAML-файла в Visual Studio для изменения внешнего вида страницы в дизайнере и значений свойств. Непосредственное редактирование кода на языке XAML для опытных программистов часто происходит намного быстрее, чем перетаскивание элементов по экрану и редактирование в окне свойств. Это особенно полезно, если нужно добавить большое количество однотипных элементов. Для облегчения редактирования XAML-кода Visual Studio предоставляет поддержку Intellisense.

При компиляции программы XAML-файл преобразуется в файл, содержащий низкоуровневые инструкции, которые создают необходимые элементы для работы программы. Объявление элементов страницы в файле XAML означает, что управление ими будет осуществляться через объекты, которые может использовать программа.

Под понятием расширяемый язык разметки подразумевается, что можно использовать правила языка для создания конструкций, которые описывают что-либо. Английский язык для этого хорошо походит. Он состоит из букв и знаков препинания, которые являются символами текста английского языка. Также в языке есть правила (их называют грамматикой), которые определяют, как составить слова и предложения. Существуют различные виды слов. Есть существительные, которые описывают предметы, и глаголы, которые описывают действия. Когда появляется что-то новое, мы придумываем новые слова для того, чтобы описать это.

Языки, основанные на XML (eXtensible Markup Language — расширяемый язык разметки), являются расширяемыми в том смысле, что мы можем придумать новые слова и фразы, которые подчиняются правилам языка, и использовать эти новые конструкции для описания того, что нам нужно. Их называют языками разметки, потому что они часто используются для описания расположения элементов на странице. Слово "разметка" первоначально использовалось при печати. Практически самым известным языком разметки является HTML — язык разметки гипертекста (HyperText Markup Language), который используется во всемирной паутине для описания формата веб-страниц.

Программисты часто создают свои собственные форматы хранения данных, используя XML. В качестве примера рассмотрим фрагмент XML-файла, в котором представлен список рекордов:

<?xml version="1.0" encoding="us-ascii" ?>

<HighScoreRecords count="2">

 <HighScore game="Breakout">

 <playername>Rob Miles</playername>

 <score>1500</score>

 </HighScore>

 <HighScore game="Space Invaders">

 <playername>Rob Miles</playername>

 <score>4500</score>

 </HighScore>

</HighScoreRecords>

В этом файле элемент HighScoreRecords содержит два элемента HighScore: один для игры Breakout и один для игры Space Invaders. Оба элемента HighScore содержатся внутри элемента HighScoreRecords. У каждого элементов есть атрибут, который определяет название игры, и два вложенных элемента: имя игрока playername и количество набранных очков score. Этот текст довольно легко понять.

В самой первой строке файла указана версия стандарта XML и используемая кодировка символов. XAML использует правила расширяемого языка разметки, чтобы создать язык, который описывает компоненты на странице экрана.

<TextBox Height="72" HorizontalAlignment="Left" Margin="8,19,0,0"

 Name="firstNumberTextBox" Text="0" VerticalAlignment="Top" Width="460"

 TextAlignment="Center" />

Если посмотреть на описание элемента TextBox, можно увидеть, что создатели языка XAML просто создали имена полей, которые соответствуют их требованиям.

XML-схема. Стандарт XML также содержит описания, как создать схему, которая описывает особый формат документа. Например, в схеме для информации о рекордах можно указать, что элемент HighScore должен содержать элемент PlayerName, атрибут Score и необязательный атрибут Date, в котором может храниться информация о дате установленного рекорда.

Система стандарта формата и схемы означает, что разработчики могут легко создавать форматы данных для конкретных целей. Этому способствует факт, что существует много инструментальных средств, которых помогают создавать документы и схемы. Среда .NET даже предоставляет программам возможность сохранения объектов в виде отформатированных XML-документов. Даже файл решения Visual Studio хранится как XML-документ.

Язык XAML на страницах приложений. В XAML-файле можно описать целую страницу экрана Windows Phone. При создании нового проекта Windows Phone автоматически создается новая страница с несколькими элементами. При добавлении на страницу элементов увеличивается и размер файла, поскольку для каждого элемента создается его описание. Некоторые элементы являются контейнерами — они могут содержать другие компоненты, например, элемент Grid может содержать набор других элементов и располагать их в виде сетки. XAML-код также может включать описания анимации и преобразований, которые можно применять к элементам на странице для создания усовершенствованных пользовательских интерфейсов.

При этом, в XAML не существует понятия наследования: нельзя создать элемент XAML, который наследуется от другого. Элементы Silverlight представляются в виде объектов, которые являются частью иерархии классов, но когда они записаны в файле XAML, они все представляются элементами, которые находятся на одном уровне иерархии.

С помощью XAML можно создать довольно сложный дизайн для программ. Также существует специальный инструмент Expression Blend для графических дизайнеров, который содержит расширенные средства для создания графических интерфейсов.

## Пример создания приложения Silverlight для Windows Phone

Для того чтобы приложение выполняло какие-то действия, нужно добавить код программы на языке C#, который выполнит необходимую работу. Когда Visual Studio создает файл XAML, который описывает страницу приложения Windows Phone, вместе с ним создается файл кода. В этот файл можно поместить код, который заставит приложение работать. Созданный файл уже содержит начальный код, который может выглядеть так:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

using Microsoft.Phone.Controls;

namespace PhoneApp1

{

 public partial class MainPage : PhoneApplicationPage

 {

 // Конструктор

 public MainPage()

 {

 InitializeComponent();

 }

 }

}

Большую часть файла занимают инструкции using, которые позволяют программе напрямую обращаться к классам без необходимости указывать полный путь к ним. Например, вместо того чтобы записать System.Windows.Controls.Button, можно написать просто Button, потому что файл содержит строку

using System.Windows.Controls;

Единственный метод в программе — конструктор класса MainPage, который вызывается при создании класса. В конструкторе вызывается метод InitializeComponent, который создает экземпляры визуальных элементов. Этот код автоматически создает Visual Studio на основе XAML-файла, который описывает структуру страницы. Код этого метода не рекомендуется изменять напрямую, поскольку это может привести к ошибкам при выполнении программы.

Не нужно беспокоиться о том, как создаются и выводятся на экран визуальные объекты. Достаточно использовать высокоуровневые инструменты или научиться писать XAML-код для создания интерфейса страниц.


### Создание приложения

Рассмотрим процесс создания интерфейс для нашей программы для сложения чисел. После того как мы добавим все необходимые элементы и запустим приложение, мы можем выполнить в нем некоторые действия. Мы можем ввести числа в текстовые поля и нажать на кнопку Cумма ([рис. 5](http://www.intuit.ru/department/se/devmobwphn/3/devmobwphn_3.html#image.3.5)).

Если прикоснуться к элементу TextBox, появится экранная клавиатура, с помощью которой можно ввести числа в поля. Если мы прикоснемся к другой области экрана, клавиатура исчезнет. Однако, при нажатии на кнопку сумма мы не получим результат, поскольку приложение "не знает", что оно должно выполнять вычисления.

**Рис. 5.** Экран программы Сумматор с открытой экранной клавиатурой

### Вычисление результата

Необходимо создать программный код для выполнения необходимых вычислений и вывода на экран результата:

private void calculateResult()

{

 float v1 = float.Parse(firstNumberTextBox.Text);

 float v2 = float.Parse(secondNumberTextBox.Text);

 float result = v1 + v2;

 resultTextBlock.Text = result.ToString();

}

Объекты TextBox работают со значением свойства Text, доступным для чтения и записи. При записи значения в свойство Text текст в текстовом поле изменится, а при считывании значения этого свойства программа получает значение, которое было введено в это поле.

Текст имеет строковый тип, который нужно преобразовать в числовое значение, для того чтобы можно было выполнить операцию сложения. Для этого используется метод Parse, который принимает в качестве параметра строку и возвращает записанное в строке число. Метод Parse есть у каждого из числовых типов (int, float, double и т.д.). Наша программа Сумматор может работать как с целыми, так и с дробными числами. Метод Parse класса float преобразует введенные в текстовые поля значения в численный тип и складывает их.

После этого, метод преобразует полученное значение в строку и записывает ее в свойство Text элемента resultTextBlock. Метод ToString противоположен Parse. Он возвращает текст, который описывает содержание объекта. Для типа float метод возвращает численное значение, записанное в строке.

В результате, мы получили код, который выполняет вычисления и получает результат. Теперь нужно, чтобы этот код выполнялся, когда пользователь нажимает на кнопку сумма.

### События и программы

Во времена консольных приложений программа всегда запускалась с самого начала, работала некоторое время и затем завершала свою работу. Но теперь в нашем распоряжении есть богатые пользовательские интерфейсы с большим количеством кнопок и других элементов для взаимодействия с пользователями. В этой ситуации очень трудно написать программу, которая проверяла бы последовательно каждый элемент интерфейса, чтобы увидеть, попытался ли пользователь его использовать. Вместо этого, система ожидает от этих визуальных элементов каких то действий.

Использование событий существенно облегчает создание программ. Программа не должна проверять каждый элемент на экране, чтобы узнать, выполнил ли пользователь над ним какие-то действия — вместо этого программа просто отвечает на генерируемые события.

Чтобы заставить события работать, языку программирования нужен способ представления ссылки на метод объекта, для чего в C# используется механизм делегатов. Можно создать тип делегата, который может обратиться к определенному методу, и создать экземпляр этого делегата, который обращается к методу в объекте. Делегаты являются мощным и полезным механизмом.

### События в Silverlight

В C# событие отправляется объекту посредством вызова метода этого объекта. В нашей программе Сумматор когда пользователь нажимает на кнопку сумма, программа должна вызвать нужный метод, например, метод calculateResult.

Можно использовать редактор Visual Studio: чтобы связать метод с событием щелчка кнопки мыши, нужно дважды щелкнуть по этой кнопке в окне дизайнера. При этом, Visual Studio устанавливает свойства кнопки в XAML, для того чтобы связать кнопку с методом страницы. Кроме этого, создается пустая версия этого метода-обработчика, и среда переносит курсор к этому методу, чтобы можно было сразу начать добавлять код, который должен выполняться при нажатии кнопки.

Теперь нужно просто добавить следующий код, который выполнит необходимые действия:

private void calculateResult()

{

 float v1 = float.Parse(firstNumberTextBox.Text);

 float v2 = float.Parse(secondNumberTextBox.Text);

 float result = v1 + v2;

 resultTextBlock.Text = result.ToString();

}

private void equalsButton\_Click(object sender, RoutedEventArgs e)

{

 calculateResult();

}

Созданному обработчику событий Visual Studio автоматически присваивает имя, которое формируется из имени элемента, в конец которого добавляется текст \_Click. Таким образом, по этому имени можно легко определить, каким образом он вызывается.

Теперь программа работает. Однако, если пользователь введет в текстовое поле нечисловое значение, то программа сгенерирует исключение.

### Управление свойствами событий

Чтобы узнать, как происходит вызов событий, для начала рассмотрим свойства кнопки сумма нашего приложения. Для этого нужно нажать на кнопку в редакторе Visual Studio, и мы увидим информацию о ее свойствах ([рис. 6](http://www.intuit.ru/department/se/devmobwphn/3/devmobwphn_3.html#image.3.6)).

|  |  |
| --- | --- |
| **Рис. 6.** Область свойств элемента Button | **Рис. 7.** События элемента Button |

В области свойств есть две вкладки: Свойства и События. При нажатии на вкладку События на экран выводятся события, которые может сгенерировать этот элемент, и связанные с событиями методами ([рис. 7](http://www.intuit.ru/department/se/devmobwphn/3/devmobwphn_3.html#image.3.7)).

В этом окне событие Click связано с методом equalsButton\_Click. Если нужно прекратить связь метода с событием Click, то достаточно удалить название обработчика в этом окне. Если попытаться подставить вместо указанного имени метода произвольную строку, то программа не будет скомпилирована, если метода с указанным именем не существует.

До сих пор мы руководствовались принципом, что отображаемые свойства Visual Studio связывает с файлом XAML, который описывает компоненты на экране. Мы можем проверить, что это так, рассмотрев описание кнопки в файле XAML для страницы MainPage:

<Button Content="equals" Height="72" HorizontalAlignment="Left" Margin="158,275,0,0"

 Name="equalsButton" VerticalAlignment="Top" Width="160"

 Click="equalsButton\_Click" />

Описание кнопки теперь содержит элемент, который связывает событие Click с названием метода. Следует отметить, что, если в файле XAML указано, что кнопка связана с методом, которого не существует, программа не сможет правильно работать.

## Краткие итоги

1. Silverlight предоставляет возможность создавать пользовательские графические интерфейсы.
2. Пользовательский интерфейс Silverlight состоит из визуальных элементов: текстовые поля, кнопки и другие.
3. Интегрированная среда разработки Visual Studio содержит окно дизайнера, которое используется для добавления элементов пользовательского интерфейсаSilverlight на страницы приложения.
4. С точки зрения программного обеспечения, каждый элемент пользовательского интерфейса представлен особым типом объекта в иерархии классов, которая является частью Silverlight.
5. Дизайнеры могут изменять значения свойств элементов, изменяя эти элементы в окне дизайнера Visual Studio или изменяя значения свойств этих элементов.
6. Свойства используемых в программе элементов Silverlight хранятся в текстовых файлах в формате XAML. Эти файлы обновляются средствами проектирования и используются для создания программных объектов, используемых в решении.
7. XAML (расширяемый язык разметки приложений) основан на языке XML и определяет все свойства элементов дизайна на странице. Он позволяет отделить дизайн и свойства элементов от программного кода.
8. XML (расширяемый язык разметки) является основой для создания языков для описания сущностей.
9. Элементы могут генерировать события, которые могут быть связаны с методами программы на C#. Имя метода указывается в XML-описании для элемента.
10. Методы, связанные с событиями, содержат бизнес-логику приложения.

## Упражнения

### Упражнение 1. Пользовательский интерфейс программы Калькулятор времени

Ваша компания решила создать приложение для адвокатов. Приложение должно хранить и управлять информацией о времени, которое тратит их персонал на работу с клиентами. В настоящее время адвокаты записывают в отчет количество минут, которые они потратили на работу с клиентом, и эти отчеты обрабатываются вручную. После того как адвокатам выдали телефоны на платформе Windows Phone, было решено создать простой Калькулятор времени, который они смогут использовать для расчета времени.

Адвокаты будут вводить время запуска и время окончания (в часах и минутах), и программа должна выводит на экран количество минут между этими отметками времени.

Ваш менеджер попросил Вас разработать пользовательский интерфейс Silverlight для этого приложения:

1. Определите, какие элементы интерфейса Silverlight нужно разместить на главной странице приложения.
2. Нарисуйте схему расположения элементов на форме.
3. Создайте «Руководство пользователя» для приложения.
4. Попытайтесь определить все возможные случаи ввода недопустимых значений.
5. Добавьте описание действий приложения в случае ввода недопустимых данных.
6. Создайте набор тестовых данных и результатов, которые должна выдать программа, и оформите их в виде таблицы. Количество тестов должно быть не менее 10.

### Упражнение 2. Синхронизация данных калькулятора времени

Один из менеджеров по продукции считает, что телефон должен отправлять детализированную информацию о каждом временном интервале в корпоративную систему так, чтобы информация о проведенном времени могла обновляться автоматически. Менеджер решил, что нужно отправлять следующую информацию:

* дата и время начала работы;
* дата и время окончания работы;
* название компании-клиента;
* местоположение клиента, если встреча происходит не в помещении компании клиента;
* оценка качества работы, поставленная клиентом — целое число в диапазоне от одного до пяти.

На сервер нужно отправлять отчеты, содержащие указанную информацию о каждой рабочей встрече.

1. Создайте пример XML-файла, содержащий описания трех встреч.
2. Удостоверьтесь, что файл содержит записи о встречах в помещении компании-заказчика, а также записи о встречах за пределами компании.

## 6. Справочный материал

### Среда разработки

После установки средств разработки Windows Phone SDK в диалоговом окне Создать проект в Visual Studio появятся группы проектов для Silverlight for Windows Phone:

После установки разработчику доступны следующие шаблоны приложений Silverlight for Windows Phone:

* Windows Phone Application
* Windows Phone Databound Application
* Windows Phone Class Library
* Windows Phone Panorama Application
* Windows Phone Pivot Application
* Windows Phone Silverlight and XNA Application
* Windows Phone Audio Playback Agent
* Windows Phone Audio Streaming Agent
* Windows Phone Scheduled Task Agent

### Шаблоны приложений

Среди шаблонов проектов имеются три шаблона, представляющих собой три основных стиля приложения для Windows Phone:

* Windows Phone Application
* Windows Phone Pivot Application
* Windows Phone Panorama Application

**Рис. 8.** Шаблоны приложений для Windows Phone

Windows Phone Application — это аналог простого диалогового приложения, у которого один основной экран, через который происходит основное взаимодействие с пользователем.

Windows Phone Pivot Application — это некий аналог приложения с закладками, где заголовок каждой закладки определяет содержимое. Стандартный вариант использования — каждая закладка представляет собой одни и те же, в целом, данные, но в разных представлениях и/или с разной фильтрацией. Например, календарь, почтовый клиент и настройки телефона. Шаблон использует элемент управления Pivot.

Windows Phone Panorama Application — приложение-панорама, в котором зоны взаимодействия с пользователем также разделены на панели, но доступны они через горизонтальную прокрутку; фоновое изображение установлено сразу на всю панораму, она имеет общий заголовок, который прокручивается медленнее, чем панели; контент соседней панели справа виден при отображении текущей. Например, таким образом реализованы хабы в Windows Phone: Контакты, Marketplace, Фото, Музыка+видео и др. Шаблон использует элемент управления Panorama.

### Структура простого приложения

Рассмотрим структуру проекта в обозревателе решений Visual Studio:

По умолчанию новый проект содержит следующие файлы:

* AppManifest.xml — файл манифеста, необходимый для генерации XAP файла, в который упаковывается приложение для развёртывания на телефоне;
* AssemblyInfo.cs — ещё один конфигурационный файл, в котором определяются некоторые метаданные главной сборки приложения;
* WMAppManifest.xml — файл метаданных, который содержит разнообразные настройки приложения: заголовок, задание первой страницы, пути к значкам, определение необходимых системных возможностей и т.д.;

**Рис. 9.** Обозреватель решений Visual Studio

* App.xaml — файл ресурсов приложения. Здесь располагаются глобальные ресурсы или глобальные события (которые происходят при запуске приложения). Этот файл также является точкой входа приложения;
* App.xaml.cs — файл отделённого кода для App.xaml. Здесь можно обрабатывать события и ошибки уровня приложения;
* ApplicationIcon.png — картинка, которая будет значком приложения в телефоне. Этот файл является первым, что увидят пользователи при работе с приложением;
* Background.png — картинка, которая используется, когда приложение закреплено на стартовом экране телефона;
* MainPage.xaml — главная страница приложения. Эта страница представляет интерфейс, который видит пользователь при запуске приложения;
* MainPage.xaml.cs — файл кода страницы MainPage.xaml;
* SplashScreenImage.jpg — картинка, которая отображается во время загрузки приложения.

Файлы XAML определяют интерфейс приложения. На самом деле это просто XML файлы с языком разметки XAML.

Несмотря на то, что это самый простой проект, он содержит все ключевые элементы, которые содержат все остальные шаблоны и типы проектов.

Обратите внимание, что часть настроек, представленных в виде конфигурационных файлов, могут редактироваться в визуальном интерфейсе редактирования настроек приложения.

### Основные элементы управления

Чтобы создать простое приложение Silverlight для Windows Phone, необходимо разместить на странице приложения визуальные элементы пользовательского интерфейса. Визуальные элементы можно разделить на несколько групп:

1. Элементы для ввода данных пользователем:
* TextBox — используется для ввода как короткого, так и многострочного текста;
* PasswordBox — специальный элемент для ввода пароля, скрывает ввод пользователя;
* CheckBox — представляет собой элемент управления "флажок". Может быть установлен или снят, опционально можно включить поддержку "неопределённого" состояния;
* RadioButton — позволяет пользователю выбрать только один вариант из нескольких. Сгруппированные элементы управления (один GroupName) позволяют выбрать только один вариант из группы;
* Slider — позволяет пользователю выбрать из нескольких последовательных вариантов. Позиция соотносится со значением свойства Value;
1. Элементы для указания выполнения действий:
* Button — кнопка, при нажатии пользователем генерирует событие Click;
* HyperlinkButton — кнопка, отображающая ссылку. При нажатии переходит на ссылку, указанную в свойстве NavigateUri;
1. Элементы для вывода данных на страницу приложения:
* TextBlock — позволяет отобразить простой текст, без возможности редактирования;
* Image — позволяет вывести на экран изображение;
* ListBox — отображает список элементов, которые может выбрать пользователь;
* ProgressBar — отображает текущий прогресс пользователю;
1. Элементы для разметки страницы:
* ScrollViewer — добавляет возможность прокручивания дочерним элементам;
* Border — предоставляет другому элементу управления рамку и/или фоновое изображение;
* StackPanel — позволяет располагать дочерние элементы в один ряд по горизонтали или вертикали с автоматическим выравниванием;
* Grid — является контейнером для других элементов управления. Позволяет располагать и упорядочивать дочерние элементы управления с привязкой к сетке;
* Canvas — определяет область, в рамках которой можно явно расположить дочерние элементы с помощью координат относительно области;
1. Специализированные элементы:
* Map — отображает карту Bing;
* MediaElement — позволяет воспроизводить звуки или видео;
* WebBrowser — отображает веб-страницу на языке HTML.

## Дополнительные материалы

1. Windows Phone 7.5 Training Kit: Labs\YourFirstWP7Application\YourFirstWP7App.html\html\ DocSet\_386d610c-7ea0-473f-b375-e10429dfd675.html.
2. Сайт MSDN: <http://msdn.microsoft.com/ru-ru/library/hh202915>.

# Лабораторная работа № 2. Модернизация программы Silverlight для Windows Phone

## Улучшение приложения

### Управление свойствами элементов

Визуальные элементы Silverlight фактически являются классами в коде, и можно настроить их отображение, изменяя свойства отдельных экземпляров этих элементов. Свойства элементов Silverlight можно задать в области свойств в Visual Studio или в XAML-коде, в котором описываются все элементы страницы. Эти настройки можно задать при создании программы, а также можно устанавливать свойства элементов и во время работы программы, например:

resultTextBlock.Text = result.ToString();

В программе можно управлять любыми свойствами визуальных элементов. Рассмотрение всех существующих свойств элементов выходит за рамки данного курса, но рекомендуется изучить их самостоятельно.

***Обработка ошибок***. На текущий момент в разработанной на прошлых лекциях программе Сумматор не осуществляется обработка ошибок. Если пользователь введет произвольный текст в элемент TextBox, то программа попытается использовать метод Parse, чтобы преобразовать этот текст в число, и это может завершиться неудачей ([рис.](http://www.intuit.ru/department/se/devmobwphn/8/devmobwphn_8.html#image.8.1) 10).

**Рис.10.** Сообщение об ошибке при вводе недопустимого значения

Такая ошибка возникает, поскольку программный код не производит проверку на допустимость введенных значений:

float v1 = float.Parse(firstNumberTextBox.Text);

Если в элемент firstNumberTextBox ввести текст, который не может быть преобразован в число (например, "один"), то возникнет исключение. Можно изменить код, используя другую версию метода Parse:

float v1 = 0;

if (!int.TryParse(firstNumberTextBox.Text, out v1))

{

 // Код для обработки ввода недопустимого значения

}

Этот код использует метод TryParse, который возвращает значение false, если элемент firstNumberTextBox будет содержать недопустимый текст.

Изменение цвета текста. При возникновении ошибки можно выделить текст элемента TextBox красным цветом. Silverlight использует кисти для отрисовки элементов на экране. Этот мощный механизм позволяет нарисовать текст, используя изображения или текстуры, и применить некоторые дополнительные эффекты. В Silverlight есть набор стандартных цветов, и можно сделать текст элемента TextBox красным, сделав основную кисть непрерывной красной заливкой:

float v1 = 0;

if (!float.TryParse(firstNumberTextBox.Text, out v1))

{

 firstNumberTextBox.Foreground =

 new SolidColorBrush(Colors.Red);

 return;

}

Теперь, если пользователь введет недопустимое значение, основной цвет текстового поля станет красным ([рис.](http://www.intuit.ru/department/se/devmobwphn/8/devmobwphn_8.html#image.8.2) 11), и выполнение метода завершится без вычисления результата. Однако программа должна выдавать сообщение об ошибке, если пользователь введет недопустимый текст не только в первое, но и во второе текстовое поле.

Кроме этого, в приведенном примере есть еще один недостаток: после того как пользователь исправит ошибку ввода, цвет текста останется красным. Нужно добавить код для возврата цвета текста в первоначальный, когда оба числа будут введены правильно. Для этого нужно сохранить исходную кисть, чтобы ее можно было использовать повторно. Полная версия метода calculateResult может выглядеть так:

**Рис. 11.** Реакция программы на ошибку при вводе данных

private SolidColorBrush errorBrush = new SolidColorBrush(Colors.Red);

private Brush correctBrush = null;

private void calculateResult()

{

 bool errorFound = false;

 if (correctBrush == null)

 correctBrush = firstNumberTextBox.Foreground;

 float v1 = 0;

 if (!float.TryParse(firstNumberTextBox.Text, out v1))

 {

 firstNumberTextBox.Foreground = errorBrush;

 errorFound = true;

 }

 else

 firstNumberTextBox.Foreground = correctBrush;

 float v2 = 0;

 if (!float.TryParse(secondNumberTextBox.Text, out v2))

 {

 secondNumberTextBox.Foreground = errorBrush;

 errorFound = true;

 }

 else

 secondNumberTextBox.Foreground = correctBrush;

 if (errorFound)

 resultTextBlock.Text = "Ошибка при вводе!";

 else

 {

 float result = v1 + v2;

 resultTextBlock.Text = result.ToString();

 }

}

Этот код позволяет обработать возможные ошибки при вводе числовых данных. Часто создание программ, которые получают данные от пользователя, заканчивается созданием большого объема кода для вычисления результата и обработки ошибок при вводе данных.

### Редактирование XAML-кода элементов Silverlight

XAML является декларативным языком и используется для описания и настройки объектов Silverlight. Например, можно настроить режим ввода для элементов TextBox для ввода чисел.

Конфигурирование элемента TextBox для использования цифровой клавиатуры. На текущий момент программа Сумматор использует стандартную клавиатуру Windows Phone, которая появляется при выборе элемента TextBox и позволяет вводить в поле любой текст. В нашем случае предпочтительнее использовать вариант клавиатуры, предназначенный только для ввода чисел. Существует несколько конфигураций экранной клавиатуры Windows Phone.

Можно настроить все элементы TextBox, чтобы при их выборе отображалась клавиатура для ввода чисел. Для этого можно изменить соответствующее свойство элементов TextBox во время работы программы, но лучшее указать настройки элемента непосредственно в XAML-файле:

<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>

Это описание указывает Silverlight положение элемента TextBox на экране, размер элемента, его имя и некоторую информацию о выравнивании текста. Каждое из значений, заключенных в блок <TextBox ... /> является атрибутом текстового поля. Атрибуты — это простые элементы, которые не являются структурированными и представляют собой пары "имя—значение".

Чтобы указать элементу TextBox, какая форма текста должна быть введена, в него добавлен вложенный элемент TextBox.InputScope. Этот элемент является частью значения TextBox.

Простая информация о свойствах элемента TextBox представляется в виде атрибутов, в то время как значение InputScope является вложенным элементом, который содержит ряд свойств, которые описывают тип вводимого текста, например, вводимый текст должен содержать цифры и текст, но не знаки пунктуации. Нам достаточно только ввода цифр, для чего нужно добавить элемент:

<InputScopeName NameValue="Digits" />

При использовании этого описания в обоих текстовых полях в программе Сумматор при их вызове будет использоваться клавиатура для ввода числовых значений, которая является более удобной, чем клавиатура для ввода произвольного текста.

Visual Studio поддерживает технологию Intellisense, суть которой заключается в том, что редактор подсказывает, какое значение можно ввести в текущее положение курсора. Это существенно облегчает редактирование текста в программе и помогает предотвратить возможные ошибки при наборе текста.

Использование атрибутов и элементов в XAML. В языке XAML атрибуты и вложенные элементы являются взаимозаменяемыми. Значение свойства элемента может быть считано как из атрибута, так и из одноименного вложенного элемента. Можно указать следующее описание:

<TextBox InputScope="Number" Height="72" HorizontalAlignment="Left"

 Margin="144,44,0,0" Name="startHourTextBox" Text="00" VerticalAlignment="Top"

 Width="104" TextAlignment="Center" />

Это описание указывает, что вводимый текст должен быть числом. Такое описание не позволяет установить несколько значений свойства, но оказывается более простым для задания одного значения.

<TextBox HorizontalAlignment="Left" Margin="8,175,0,0" Name="secondNumberTextBox"

 Text="0" VerticalAlignment="Top" Width="460" TextAlignment="Center">

 <TextBox.Height>72</TextBox.Height>

</TextBox>

В этом описании высота элемента TextBox указана не как атрибут, а как вложенный элемент. Имя элемента должно содержать название типа элемента, частью которого является это свойство. При этом нельзя указать в качестве имени элемента просто Height, а нужно указать TextBox.Height. Это не ограничение XML, а требование XAML.

С точки зрения XAML не важно, какая из этих форм используется в описании. Для простых пар "имя—значение" можно использовать атрибуты, но для более сложных структур лучше использовать вложенные элементы.

***Настройка свойств C#.*** В коде на языке C# установка значений свойств выглядит так:

// 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 создает все элементы, представленные в описанном выше коде XAML.

### Вывод окна с сообщением

Silverlight содержит класс MessageBox, который можно использовать для сообщения пользователю об ошибке:

MessageBox.Show("Недопустимый ввод");

Эта команда выведет в верхнюю часть экрана сообщение и выдаст звуковой сигнал.

Если нужно отправить пользователю телефона сообщение с большим объемом текста, можно вывести на экран сообщение в несколько строк:

MessageBox.Show("Недопустимый ввод" + System.Environment.NewLine +

 "Пожалуйста, повторите ввод");

Элемент System.Environment.NewLine вставит символ новой строки, принятый в целевой платформе. Класс MessageBox можно использовать, если нужно убедиться в том, что пользователь принял сообщение перед продолжением работы. При выводе сообщения на экран выполнение программы будет приостановлено.

***Окно сообщения с выбором***. Можно предложить пользователю сделал выбор в окне MessageBox:

if (MessageBox.Show("Вы действительно хотите это сделать?", "Выполнение действия",

 MessageBoxButton.OKCancel) == MessageBoxResult.OK)

{

 // Действия, если пользователь нажмет "ok"

}

else

{

 // Действия, если пользователь нажмет "отмена"

}

Этот код выводит на экран сообщение с возможностью отмены действия, представленное на [рис.](http://www.intuit.ru/department/se/devmobwphn/8/devmobwphn_8.html#image.8.3) 12. Блок if позволяет выполнять разные действия в зависимости от ответа пользователя.


**Рис. 12.** Всплывающее окно с вопросом пользователю

### Добавление и использование ресурсов

В приложении можно использовать различные ресурсы, например, изображения и звуки. Все ресурсы размещаются вместе с другими файлами приложения в XAP-файл, который передается на устройство.

При использовании изображений в приложении важно помнить, что целевое устройство не содержит большой объем памяти или экран высокого разрешения. Самое высокое разрешение для Windows Phone составляет 800\*480 пикселей, поэтому нет необходимости использовать фоновое изображение большего размера.

***Ссылки на ресурсы***. Если нужно использовать ресурсы сразу в нескольких проектах, в них можно добавить не сами ресурсы, а ссылки на ресурсы. В этом случае при изменении файлов ресурсов их новая версия будет автоматически использоваться в этих проектах при их следующем построении.

Свойство ресурса "Действие при построении". При построении проекта в Visual Studio существует несколько способов добавления ресурсов к программе. Часто используются два способа: добавление ресурса как элемент контента или как части сборки.

По умолчанию при построении проекта ресурс встраивается в сборку. В этом случае свойство Действие при построении имеет значение Внедренный ресурс.

Если необходимо использовать ресурс в качестве элемента контента, нужно присвоить свойству Действие при построении значение Содержание. В этом случае при построении программы файл с ресурсом будет скопирован в каталог приложения.

***Использование в программе изображения как часть содержимого***. Ресурс, который является элементом контента, доступен в приложении в виде файла, который находится в одной папке с программой. Чтобы вывести на экран изображение из файла, нужно добавить строку в код XAML содержащую путь к этому файлу с изображением:

<Image Height="611" HorizontalAlignment="Left" Margin="8,0,0,0"

 Name="gearBackgroundImage" Stretch="Fill" VerticalAlignment="Top" Width="472"

 Source="AddingGears.jpg" />

Элемент Image выводит на экран страницы изображение. В нем описан атрибут Source, который определяет файл с изображением. Если ввести имя файла в качестве источника, то при запуске программы изображение будет загружаться из файла и выводиться на экран. Silverlight выводит элементы на экран в том порядке, в котором они описаны в XAML-файле. Если эту строку разместить перед объявлением элемента Grid, содержащего элементы страницы, то изображение будет выведено на заднем плане.

***Использование в программе изображения как внедренный ресурс***. Если в приложении используется изображение, встроенное в сборку, XAML-код для обращения к этому изображению может выглядеть так:

<Image Height="611" HorizontalAlignment="Left" Name="gearBackgrounds" Stretch="Fill"

 VerticalAlignment="Top" Width="472"

 Source="/AddingMachine;component/Images/AddingGears.jpg" />

Во время работы программы изображение будет выглядеть одинаково, независимо от способа его использования в программе. Разница заключается в расположении ресурса и в коде для обращения к нему.

***Выбор способа размещения ресурсов***. При создании приложений для Windows Phone важно понять разницу между элементами контента и внедренными ресурсами приложения.

Элемент контента — это обычный файл, который хранится в каталоге приложения. Программа может открывать и использовать файлы содержимого, когда в этом есть необходимость. Добавление файлов не влияет непосредственно на размер программы, хотя при загрузке содержимого файлов программа будет использовать больше памяти.

Внедренный ресурс сохраняется в сборке программы, и ее размер увеличивается. Когда устройство Windows Phone загружает программу, оно просматривает всю сборку программы и выполняет некоторые проверки кода. Чем больше размер программы, тем больше времени занимают проверки. Если загрузка программы занимает слишком много времени, ее выполнение будет прервано операционной системой Windows Phone. Поэтому нужно с осторожностью внедрять большие по объему ресурсы в сборку программы.

Добавление изображения как элемент содержимого позволяет уменьшить размер программы, и программа будет быстрее запускаться. Однако, элементы содержимого загружаются дольше, поскольку находятся в файловом хранилище. Добавление изображения как встроенный ресурс позволяет программе быстрее загружаться, но увеличивает размер программы, и она запускается дольше.

Рекомендуется придерживаться следующих рекомендаций:

* если программа использует много больших ресурсов, которые не должны быть загружены сразу все, то лучше хранить их в отдельных файлах. Например, можно разрешить пользователю выбирать фоновое изображение из нескольких доступных;
* если программа использует много небольших по размеру ресурсов, к которым программа часто обращается, то их лучше внедрять в сборку. Например, можно использовать собственные значки в меню настройки программы;
* если программа слишком долго запускается, рекомендуется перенести часть ресурсов из сборки в файлы и добавить в программу заставку, которая будет отображаться во время запуска программы. Также можно ускорить процесс запуска программы, разделив программу на несколько связанных сборок.

## Изменение и отображение данных

У созданной нами программы Сумматор имеется недостаток, связанный с тем, что для получения результата нужно каждый раз нажимать на кнопку. Было бы лучше, если бы результат обновлялся автоматически при изменении текста в одном из текстовых полей. Это можно осуществить с помощью событий.

### Событие TextChanged

Ранее мы использовали событие элемента Button, которое происходит при нажатии на кнопку. С точки зрения программы событие является вызовом метода. Код XAML, описывающий кнопку, может выглядеть так:

<Button Content="сумма" Height="72" HorizontalAlignment="Left"

 Margin="158,275,0,0" Name="equalsButton" VerticalAlignment="Top"

 Width="160" Click="equalsButton\_Click" />

При нажатии на кнопку вызывается указанный обработчик события. В нашем случае в нем вызывается метод calculateResult.

private void equalsButton\_Click(object sender, RoutedEventArgs e)

{

 calculateResult();

}

Можно узнать, какие события может вызывать элемент, щелкнув по нему в редакторе Visual Studio и затем нажав на кнопку События в области свойств. Там же можно назначить, и даже автоматически создать обработчики событий, которые требуется обработать в программе.

Для элемента firstNumberTextBox можно создать обработчик события TextChanged, дважды щелкнув в строке TextChanged в пустом поле справа, и вызвать в нем метод calculateResult:

private void firstNumberTextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

 calculateResult();

}

То же самое можно сделать для второго текстового поля. Теперь, при изменении текста в любом из этих полей результат будет обновляться. Следовательно, теперь можно удалить кнопку сумма.

Однако, возникает проблема с проверкой допустимости введенных значений. Если пользователь введет текст вместо числа, то программа это обнаружит и выведет на экран сообщение. Однако, это сообщение будет выводиться на экран по два раза при вводе недопустимого символа. То есть, событие происходит дважды — и это известная проблема для приложений Windows Phone. Обойти эту проблему можно, проверяя, были ли изменены данные при последовательных вызовах событий с теми же данными:

string oldFirstNumber = "";

private void firstNumberTextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

 if (firstNumberTextBox.Text == oldFirstNumber)

 return;

 oldFirstNumber = firstNumberTextBox.Text;

 calculateResult();

}

Этот обработчик вызывает метод calculateResult, только если текст в элементе TextBox действительно изменился.

### Привязка данных

Привязка данных позволяет связать визуальные элементы с методами, которые работают с этими элементами. Это позволяет связать свойство визуального элемента напрямую с объектом. Все что нужно для этого сделать — это создать особый объект, значения которого будут автоматически передаваться элементам Silverlight.

Привязка данных может работать в любом направлении. Можно связать текст элемента TextBox с приложением таким образом, чтобы при изменении текста в этом текстовом поле приложение было об этом уведомлено. Также можно связать элемент TextBlock с объектом так, чтобы при изменении свойства объекта содержимое элемента TextBlock соответственно обновлялось. Это пример двунаправленной привязки, когда программа может сама изменить элемент и отреагировать на изменения в этом элементе. Можно задать однонаправленную привязку, чтобы программа только выводила на экран результаты при изменении свойства объекта. Наконец, можно связать любые свойства объектов так, чтобы программа могла переместить элемент Silverlight по экрану при изменении свойств X и Y объекта игры.

***Создание объекта для привязки***. Попробуем создать версию программы Сумматор, использующую привязку данных. Начнем с класса, который сделает всю работу. Нужно определить свойства класса, которые будут связаны со свойствами визуальных элементов. Наш объект будет содержать три свойства:

* текст верхнего элемента TextBox;
* текст нижнего элемента TextBox;
* текст элемента TextBlock.

Можно создать класс AdderClass, имеющий следующую структуру:

public class AdderClass

{

 private int topValue;

 public int TopValue

 {

 get

 {

 return topValue;

 }

 set

 {

 topValue = value;

 }

 }

 private int bottomValue;

 public int BottomValue

 {

 get

 {

 return bottomValue;

 }

 set

 {

 bottomValue = value;

 }

 }

 public int AnswerValue

 {

 get

 {

 return topValue + bottomValue;

 }

 }

}

У этого класса три свойства. Первые два предназначены для чтения и записи, чтобы можно было получить текст верхнего и нижнего текстового поля и занести в них новые значения. Третье свойство предназначено только для получения последнего результата без возможности его изменения напрямую. Программа может создать экземпляр этого класса, установить значения свойств TopValue и BottomValue и затем получить результат.

Добавление функций уведомления. Для того чтобы класс можно было связать с объектом Silverlight, в нем должен быть реализован интерфейс INotifyPropertyChanged:

public interface INotifyPropertyChanged

{

 // Событие происходит при изменении значений свойства

 event PropertyChangedEventHandler PropertyChanged;

}

Класс, который реализует этот интерфейс, должен содержать делегат события, которое будет использовать класс AdderClass для сообщения об изменении значения свойства.

public event PropertyChangedEventHandler PropertyChanged;

Класс PropertyChangedEventHandler используется элементами Silverlight для управления сообщениями событий. Он описан в пространстве имен System.ComponentModel. Если компонент Silverlight нужно связать с каким-либо свойством класса, можно добавить к этому событию делегаты.

Класс AdderClass будет использовать этот делегат для того, чтобы сообщить об изменении одного из свойств в классе. При этом произойдет обновление экрана. Связанные с этими свойствами объекты Silverlight связываются с делегатом, чтобы они могли получить уведомление в случае необходимости. Окончательная версия класса может выглядеть так:

public class AdderClass : INotifyPropertyChanged

{

 public event PropertyChangedEventHandler PropertyChanged;

 private int topValue;

 public int TopValue

 {

 get

 {

 return topValue;

 }

 set

 {

 topValue = value;

 if (PropertyChanged != null)

 {

 PropertyChanged(this, new PropertyChangedEventArgs

 ("AnswerValue"));

 }

 }

 }

 // то же самое для свойства BottomValue

 public int AnswerValue

 {

 get

 {

 return topValue + bottomValue;

 }

 }

}

Класс AdderClass теперь содержит свойство PropertyChanged. Каждый раз при изменении значения свойства он должен проверить значение свойства PropertyChanged. Если оно не равно null, то оно связано с делегатом. Можно рассматривать свойство PropertyChanged как список рассылки. Процесс может подписаться на рассылку и получать уведомления об интересующих событиях, которые происходят в AdderClass. Если никто не подписан на список рассылки (то есть значение PropertyChanged равно null), то нет точки вызова метода для описания изменений. Однако, если оно не равно null, это означает, что есть объект, который интересуют изменения, происходящие в AdderClass.

Вызов метода, который генерирует событие, выглядит следующим образом:

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

Первый параметр является ссылкой на объект, который генерирует событие. Вторым параметром является значение аргумента, которое содержит имя свойства, значение которого изменилось. Элемент Silverlight использует технологию отражения (т.е. он будет считывать публичные свойства класса AdderClass), чтобы узнать, какие свойства доступны. Если он получит уведомление, что значение свойства AnswerValue изменилось, то он будет обновлять все свойства визуальных элементов, связанные со свойством этого класса.

Когда программа сгенерирует событие, на экран будет выведено обновленное значение свойства AnswerValue. При вызове кода секции get свойства AnswerValue он вычисляет и возвращает результат.

public int AnswerValue

{

 get

 {

 return topValue + bottomValue;

 }

}

Следующим шагом будет соединение этого объекта с пользовательским интерфейсом. Пользовательский интерфейс сообщит классу AdderClass, когда значения двух текстовых полей будут изменены пользователем. Когда будут вводиться новые значения полей, экземпляр класса AdderClass должен сообщить пользовательскому интерфейсу, что результат изменился, и его нужно вывести на экран.

***Добавление пространства имен на страницу MainPage***. Для того чтобы связать созданный объект с данными, нужно связать код объекта с XAML-кодом, описывающим пользовательский интерфейс. Необходимо добавить в код XAML пространство имен, содержащее класс AdderClass:

xmlns:local="clr-namespace:AddingMachine"

Этот атрибут корневого элемента в XAML-файле для страницы MainPage.xaml делает доступными любые классы пространства имен AddingMachine для использования в XAML-коде. Как и в программе на C# в начале файла должны указываться все директивы using, так и в начале файла XAML также должны указываться все используемые пространства имен.

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

<phone:PhoneApplicationPage.Resources>

 <local:AdderClass x:Key="AdderClass" />

</phone:PhoneApplicationPage.Resources>

***Добавление класса к элементу на странице***. После того как пространство имен стало доступным, можно соединить нужный класс из этого пространства имен с элементами страницы. В нашем случае таким элементом будет элемент Grid, который содержит все визуальные элементы страницы. Добавление класса к элементу автоматически делает его доступным для любых вложенных элементов, и таким образом, элементы TextBox и TextBlock могут использовать этот класс.

<Grid x:Name="LayoutRoot" Background="Transparent"

 DataContext="{StaticResource AdderClass}">

Этот код указывает элементу Grid использовать класс AdderClass, указанный в значении атрибута DataContext. В нашем приложении он является статическим ресурсом, в котором класс является частью программы.

***Привязка элемента Silverlight к свойству объекта***. Теперь класс AdderClass доступен в контексте данных элементам на странице. При этом создается связь между страницей Silverlight и объектом, в котором описано поведение.

Теперь нужно привязать свойства каждого элемента к свойствам объекта. Для этого можно использовать область свойств для каждого элемента. Нужно связать текст элемента firstNumberTextBox со свойством TopValue класса AdderClass. Если щелкнуть по элементу TextBox в редакторе Visual Studio, откроется область свойств для этого элемента, в которой для свойства Text в контекстном меню нужно выбрать пункт Применить привязку данных…. После этого откроется окно, в котором нужно указать, с каким свойством какого объекта требуется связать выбранное свойство элемента.

Выбрав необходимый элемент из списка доступных элементов и их свойств, можно привязать свойство элемента Silverlight к свойству объекта. Обратите внимание, что в разделе Параметры выбран режим TwoWay, при котором при изменении пользователем значения в элементе TextBox объект автоматически был об этом уведомлен.

При связывании с элементом TextBlock доступен только режим OneWay, поскольку нельзя передать в программу данные из объекта TextBlock.

После установки привязки программа будет работать. При этом содержимое файла MainPage.xaml.cs с кодом программы не изменится. Вся работа теперь выполняется в классе, который соединен с визуальными элементами. Изменение содержимого элементов TextBox, используемых для ввода значений, приведет к возникновению события класса AdderClass, которые приведут к изменению содержимого класса AnswerValue, связанного с элементом ResultTextBlock.

### Привязка данных через свойство DataContext

Для установки привязки к данным потребовалось проделать большую работу. Однако, существует способ упростить выполнение этих действий.

***Установка привязки данных в XAML***. Можно просто связать свойство объекта со свойством элемента в XAML-коде, и Silverlight выполнит привязку.

<TextBox Height="72" HorizontalAlignment="Left" Margin="8,19,0,0"

 Name="firstNumberTextBox" Text="{Binding TopValue, Mode=TwoWay}"

 VerticalAlignment="Top" Width="460" TextAlignment="Center" />

В этом коде указана привязка текста элемента к свойству TopValue объекта. Режим привязки установлен в значение TwoWay, чтобы изменения в программе (при вычислении результата) отображались на экране. Если нужно использовать визуальный элемент только для того, чтобы показать значение (например, результат вычисления), можно использовать режим привязки OneWay.

***Установка свойства DataContext***. Теперь остается создать экземпляр класса AdderClass и присвоить его свойству DataContext элемента, содержащего элементfirstNumberTextbox. Это можно сделать в конструкторе для основной страницы.

// Конструктор

public MainPage()

{

 InitializeComponent();

 AdderClass adder = new AdderClass();

 ContentGrid.DataContext = adder;

}

Свойство DataContext элемента Silverlight идентифицирует объект, который содержит все свойства, связанные с элементом и с теми элементами, которые в нем содержатся. Элемент ContentGrid содержит элементы TextBox и TextBlock, и теперь каждая привязка, которую эти элементы содержат, будет отображаться на созданный экземпляр класса AdderClass.

В этом примере вся тяжелая работа с областью свойств и ресурсами заменена на небольшое количество XAML-кода и одну строку кода.

## Управление ориентацией страницы приложения

### Альбомная и книжная ориентация

Существует два варианта ориентации устройства: альбомная (устройство расположено горизонтально) и книжная (устройство расположено вертикально). Телефон может определить ориентацию, и некоторые встроенные приложения правильно обрабатывают изменение ориентации телефона.

Можно создавать приложения, которые смогут корректно работать в обоих режимах ориентации. При этом, потребуется потратить вдвое больше времени на создание пользовательского интерфейса и на написание программного кода, обрабатывающего изменение ориентации телефона и изменяющего соответственно ориентацию приложений.

Указание ориентации в файле MainPage.xaml. Страница может сообщить Silverlight, какую ориентацию она поддерживает. Для этого используются свойства класса PhoneApplicationPage:

SupportedOrientations="Portrait" Orientation="Portrait"

Эти настройки Visual Studio использует по умолчанию при создании страницы. Они указывают на то, что страница использует только портретную ориентацию. В случае необходимости эти настройки можно изменить:

SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"

С этими настройками при изменении ориентации телефона приложение также будет изменять ориентацию ([рис.](http://www.intuit.ru/department/se/devmobwphn/8/devmobwphn_8.html#image.8.4) 13).


**Рис. 13.** Приложение Сумматор в альбомной ориентации

Теперь программа изменяет содержимое экрана при изменении ориентации устройства, но поскольку расположение всех элементов задано фиксированными значениями, некоторые из них находятся не в том месте, в каком хотелось бы их видеть, а поле для результата не видно на экране.

<TextBox Height="72" HorizontalAlignment="Left" Margin="8,19,0,0"

 Name="firstNumberTextBox" Text="0" VerticalAlignment="Top" Width="460"

 TextAlignment="Center" />

Этот элемент использует значения отступа от краев экрана, которые используются для точного указания расположения элементов на экране. При этом, если у элемента указана позиция, которая находится за пределами экрана, то Silverlight просто не будет отображать этот элемент на экране.

Существует два способа исправления подобной ошибки. Один способ заключается в изменении параметров расположения элементов при изменении ориентации телефона.

Событие OrientationChanged. Класс PhoneApplicationPage включает событие OrientationChanged, которое происходит при изменении ориентации страницы. Как и для других событий, для него можно создать метод-обработчик, в который нужно добавить код для обработки изменения ориентации экрана:

private void PhoneApplicationPage\_OrientationChanged(object sender,

 OrientationChangedEventArgs e)

{

 if (e.Orientation == PageOrientation.PortraitUp)

 {

 setPortrait();

 }

 else

 {

 setLandscape();

 }

}

Этот код использует параметр, значение которого определяет новую ориентацию, и вызывает соответствующий метод. Методы изменяют значения отступов от краев элементов, чтобы разместить их подходящим для определенной ориентации образом:

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);

}

Код метода setPortrait аналогичен методу setLandscape. Элементы располагаются в соответствии со значениями свойства Margin. Отступы от краев задаются значением свойства Thickness, которое содержит величину отступа и ширину границы. В нашем случае ширина границы установлена в 0, поэтому граница не будет отображаться. После добавления этих методов приложение будет работает в обоих режимах ориентации ([рис.](http://www.intuit.ru/department/se/devmobwphn/8/devmobwphn_8.html#image.8.5) 14).


**Рис. 14.** Приложение Сумматор, настроенное под разные режимы ориентации

Часто приложения при изменении ориентации экрана используют анимацию перемещения элементов. Для создания подобных эффектов рекомендуется использовать инструмент Expression Blend.

### Использование контейнеров для группировки элементов

Среди элементов Silverlight есть элементы-контейнеры, которые могут содержать другие элементы. Существует несколько различных типов контейнеров, среди которых есть элемент StackPanel.

Элемент StackPanel содержит последовательность элементов Silverlight. Он позволяет располагать вложенные в него элементы в определенном порядке. Элементы в контейнере могут располагаться вертикально (сверху вниз) или горизонтально (слева направо). Вместо того, чтобы настраивать для визуальных элементов отступы от краев экрана, можно поручить эту работу элементу StackPanel:

<StackPanel>

 <TextBox InputScope="Digits" Height="72" HorizontalAlignment="Center"

 Name="firstNumberTextBox" VerticalAlignment="Top" Width="460"

 TextAlignment="Center" />

 <TextBlock Height="56" HorizontalAlignment="Center" Name="plusTextBlock" Text="+"

 VerticalAlignment="Top" FontSize="32" Width="25" />

 <TextBox InputScope="Digits" Height="72" HorizontalAlignment="Center"

 Name="secondNumberTextBox" VerticalAlignment="Top" Width="460"

 TextAlignment="Center" />

 <TextBlock Height="46" HorizontalAlignment="Center" Name="resultTextBlock"

 VerticalAlignment="Top" FontSize="30" Width="160" TextAlignment="Center" />

</StackPanel>

Элементы StackPanel решают сами, как будут отображаться элементы на экране. В этом коде лишь указано выравнивание всех элементов по центру выделенной им области.

При таком подходе возникает очень полезный побочный эффект, который заключается в том, что контейнер StackPanel автоматически перестроит элементы при изменении ориентации экрана ([рис.](http://www.intuit.ru/department/se/devmobwphn/8/devmobwphn_8.html#image.8.6) 15).


**Рис. 15.** Приложение Сумматор, использующее элемент StackPanel

Элемент StackPanel можно добавить к проекту, перетащив его с панели инструментов Visual Studio в окно дизайнера или вводя текст XAML вручную. Если элементStackPanel должен группировать элементы горизонтально, нужно указать соответствующий атрибут:

<StackPanel Orientation="Horizontal">

Также можно поместить один StackPanel в другой, что позволяет использовать другую группировку вложенных элементов.

Существуют и другие типы контейнеров. Один из них — элемент Grid. Он позволяет создать прямоугольную сетку из элементов, в каждом из которых может размещаться один или несколько элементов. Фактически в приложении Сумматор экран содержит сетку, состоящую из одного элемента, который занимает всю доступную область экрана.

## Отображение списков данных

Часто в приложениях необходимо выводить списки каких-либо объектов. Для этого необходимы специальные средства, предусматривающие отображение списочных данных.

В качестве примера рассмотрим создание списка клиентов компании. Информация о клиентах включает: имя клиента, адрес и идентификатор. Класс Customer может выглядеть так:

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;

 }

}

Этот класс имеет единственный метод, который является конструктором, принимающим три параметра.

Теперь нужно сохранить список клиентов. Лучший способ для хранения списков объектов — использование класса List, который входит в состав языка C#. Можно создать класс Customers, содержащий имя списка клиентов и сам список:

public class Customers

{

 public string Name { get; set; }

 public List<Customer> CustomerList;

 public Customers(string inName)

 {

 Name = inName;

 CustomerList = new List<Customer>();

 }

}

Класс Customers содержит единственный метод, который является конструктором класса. Он определяет имя списка и создает пустой список. Используя этот класс, можно создать новый список таким образом:

Customers mailCustomers = new Customers("Список клиентов");

Теперь нужно заполнить созданный пустой список информацией о клиентах.

### Создание данных для списка

Для нашего примера не имеет значение, какие имена будут у клиентов — главное, чтобы информация о клиентах была разной. Обычно такую информацию удобно хранить в базе данных, но мы пока сгенерируем список на основе небольшого количества данных. Создадим метод, который будет возвращать готовый список:

public static Customers MakeTestCustomers()

{

 string[] firstNames = new string[] { "Rob", "Jim", "Joe", "Nigel", "Sally",

 "Tim"};

 string[] lastsNames = new string[] { "Smith", "Jones", "Bloggs", "Miles",

 "Wilkinson", "Brown" };

 Customers result = new Customers("Тестовый список");

 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));

 // увеличение значения идентификатора для следующего клиента

 id++;

 }

 }

 return result;

}

В этом коде создается 36 клиентов комбинированием всех заданных имен и фамилий. У каждого клиента свой уникальный идентификационный номер. МетодMakeTestCustomers является статическим, чтобы не нужно было создавать список перед получением тестовых данных. Теперь можно использовать полученный список в программе:

customerList = Customers.MakeTestCustomers();

### Использование элемента StackPanel для вывода списка на экран

Следующим шагом будет вывод полученного списка клиентов на экран Windows Phone. Для этого можно поместить имя каждого клиента в отдельный элемент TextBlock. Требуется, чтобы элементы могли размещаться на экране независимо от их количества. Для этого можно использовать элемент StackPanel:

<StackPanel HorizontalAlignment="Left" Margin="0,0,0,0" Name="customersStackPanel"

 VerticalAlignment="Top" />

Элементы TextBlock можно создавать в программе с помощью следующего кода:

customers = Customers.MakeTestCustomers();

foreach (Customer c in customers.CustomerList)

{

 TextBlock customerBlock = new TextBlock();

 customerBlock.Text = c.Name;

 customersStackPanel.Children.Add(customerBlock);

}

В этом коде для каждого клиента создается элемент TextBlock, в котором в текстовое поле записывается имя клиента, после чего этот элемент добавляется customersStackPanel. Как видно из примера, визуальные объекты можно создавать как в XAML-коде, так и в коде программы на C#.

Полученная программа может выводить на экран список любого размера ([рис.](http://www.intuit.ru/department/se/devmobwphn/8/devmobwphn_8.html#image.8.7) 16).


**Рис. 16.** Список клиентов

Однако, некоторые элементы не видны на экране, поскольку находятся за его пределами. Это является существенным недостатком. Решить проблему можно, добавив в программу возможность прокрутки экрана вверх и вниз, чтобы пользователь мог увидеть остальную часть списка. Такую возможность предоставляет элемент ScrollViewer, если в него поместить элемент StackPanel:

<ScrollViewer>

 <StackPanel HorizontalAlignment="Left" Margin="9,6,0,0"

 Name="customersStackPanel" VerticalAlignment="Top" />

</ScrollViewer>

Теперь в приложении отображается полоса прокрутки, и пользователь может прокручивать экран вверх и вниз, чтобы просмотреть список полностью.

### Использование элемента ListBox для отображения списка элементов

В представленном примере нам пришлось самостоятельно создавать элементы и присваивать им значения, после чего размещать эти элементы на странице. Вместо выполнения этих действий можно использовать привязку данных и специальные элементы для размещения на странице списков данных.

Создание шаблона данных. Необходимо связать класс Customer с визуальным элементом, но нельзя сделать это напрямую, как это было сделано в программе Сумматор, потому что экземпляр класса Customer содержит несколько значений: имя клиента, его адрес и идентификатор. Необходимо создать шаблон данных, который определяет, как информация о клиенте будет выглядеть на экране. Шаблон может иметь следующий вид:

<DataTemplate>

 <StackPanel>

 <TextBlock Text="{Binding Name}" />

 <TextBlock Text="{Binding Address}" />

 </StackPanel>

</DataTemplate>

Этот шаблон данных определяет, что информация должна выводиться в элемент StackPanel, который содержит два элемента TextBlock с именами Name и Address. В соответствии с этим шаблоном будут выводиться две строки: в первой строке указывается имя клиента, во второй — его адрес. При этом в шаблоне не указано значение идентификатора, поскольку нам не нужно выводить это значение на экран.

Использование шаблона DataTemplate для списка. Шаблон данных указывает Silverlight как нужно выводить на экран свойства объекта. Его можно использовать для вывода списка клиентов на экран. Для этого нужно поместить шаблон DataTemplate в секции ItemTemplate элемента ListBox, который выведет на экран информацию о клиентах:

<ListBox Name="customerList">

 <ListBox.ItemTemplate>

 <DataTemplate>

 <StackPanel>

 <TextBlock Text="{Binding Name}" />

 <TextBlock Text="{Binding Address}" />

 </StackPanel>

 </DataTemplate>

 </ListBox.ItemTemplate>

</ListBox>

Этот код настраивает внешний вид списка данных, вывод которого на экран осуществляет элемент customerList. Теперь нужно указать список данных, которые должны выводиться на экран:

customers = Customers.MakeTestCustomers();

customerList.ItemsSource = customers.CustomerList;

Этот код получает тестовый список клиентов, который присваивается свойству ItemSource элемента customerList. Элемент ListBox выведет на экран каждый объект в наборе как элемент списка ([рис. 17](http://www.intuit.ru/department/se/devmobwphn/8/devmobwphn_8.html#image.8.8)).


**Рис. 17.** Список клиентов с адресами

В коде программы необходимо указать свойству ItemsSource элемента ListBox список выводимых данных, а все остальные действия выполняются автоматически. Элемент ListBox также автоматически отображает полосы прокрутки, если список не помещается на экране целиком. Необходимо только разработать шаблон для отображения элементов списка и задать привязку к данным.

Добавление стиля отображения. В нашем примере в выводимом списке информация выводится не очень наглядно, так как трудно визуально отличить имя клиента от его адреса. Для повышения удобочитаемости списка можно задать разные размеры и стили шрифта для имен и адресов, но может быть удобнее использовать встроенные стили, которые предоставляются в виде ресурсов для приложений Windows Phone.

Преимущество использования встроенных стилей заключается в том, что они используют различные цветовые схемы Windows Phone. К элементам TextBlock в шаблоне DataTemplate можно добавить названия используемых стилей:

<DataTemplate>

 <StackPanel>

 <TextBlock Text="{Binding Name}"

 Style="{StaticResource PhoneTextExtraLargeStyle}" />

 <TextBlock Text="{Binding Address}"

 Style="{StaticResource PhoneTextSubtleStyle}" />

 </StackPanel>

</DataTemplate>

В этом коде используются встроенные стили ExtraLargeStyle для имени клиента и SubtleStyle для адреса. Теперь список визуально воспринимается гораздо лучше ([рис. 18](http://www.intuit.ru/department/se/devmobwphn/8/devmobwphn_8.html#image.8.9)).

Выбор элементов списка. Следующим шагом будет добавление в программу возможности редактирования данных. При выборе одного из элементов списка должен отображаться экран для редактирования выбранного элемента ([рис. 19](http://www.intuit.ru/department/se/devmobwphn/8/devmobwphn_8.html#image.8.10)).

**Рис. 18.** Список клиентов с адресами, использующий встроенные стили


**Рис. 19.** Экран приложения для редактирования информации о выбранном клиенте

При выборе элемента списка элемент ListBox генерирует событие SelectionChanged, для которого можно создать обработчик. Обработчик может выглядеть так:

private void customerList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

 // получение содержимого выбранного элемента

 Customer selectedCustomer = customerList.SelectedItem as Customer;

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

}

Класс customerList содержит свойство SelectedItem, который является ссылкой на выбранный объект. В этом коде выбранный объект приводится к типуCustomer, и программа выводит на экран окно с сообщением, в котором указано имя выбранного клиента.

Теперь необходимо добавить возможность перехода к новой странице приложения, которая позволит изменять информацию о выбранном клиенте. Для этого необходимо изучить, как в приложениях Silverlight выполняется переход к другой странице приложения.

##  Навигация по страницам приложения

До сих пор все рассмотренные примеры программ использовали только одну страницу Silverlight. Однако, большинство приложений содержат несколько страниц. В нашем примере необходима дополнительная страница для вывода и редактирования информации о выбранном клиенте.

Для добавление новой страницы в приложение нужно выбрать в главном меню Visual Studio пункт Проект -> Добавить новый элемент…. Откроется окно со списком шаблонов, которые можно добавить в проект. Выберем в списке пункт Страница Windows Phone в книжной ориентации и укажем имя новой страницы CustomerDetailPage.xaml. После нажатия на кнопку Добавить в проект будет добавлена новая страница приложения Windows Phone. Работа с новой страницей ничем не отличается от работы со страницей MainPage.

### Навигация по страницам

Модель перемещения по страницам Silverlight больше похожа на навигацию в Интернете, чем на приложение Windows Forms. У каждой страница есть свой адрес, который указывается в виде URI (универсальный идентификатор ресурса). Объект NavigationService предоставляет методы для навигации.

// перемещение к странице CustomerDetailPage.xaml

NavigationService.Navigate(new Uri("/CustomerDetailPage.xaml",

 UriKind.RelativeOrAbsolute));

Этот код вызовет переход приложения на страницу CustomerDetailPage. Обратите внимание, что метод Navigate принимает значение URI, которое содержит название страницы и тип URI.

При задании URI необходимо выбрать, будет указан относительный или абсолютный адрес. Установка типа в значение RelativeOrAbsolute обеспечивает гибкость, при загрузке ресурсов. Значение URI является обычной строкой, поэтому необходимо внимательно следить за правильностью указания ее значения, чтобы избежать ошибок во время выполнения программы.

Использование кнопки Назад. Пользователи Windows Phone ожидают, что при нажатии на аппаратную кнопку Назад произойдет переход к предыдущей странице. Эта функциональность автоматически поддерживается во всех приложениях для Windows Phone. Если пользователь нажмет кнопку Назад на странице MainPage, то приложение завершит работу.

Иногда необходимо переопределить это поведение, например, чтобы пользователь не переходил на другую страницу, если не сохранил введенные данные. Переопределить поведение кнопки Назад можно в обработчике события BackKeyPress, в котором можно при необходимости отменить переход:

private void PhoneApplicationPage\_BackKeyPress(object sender,

 System.ComponentModel.CancelEventArgs e)

{

 e.Cancel = true;

}

Этот метод-обработчик события BackKeyPress отменяет переход к предыдущей странице при нажатии на кнопку Назад.

Если нужно, чтобы пользователь подтвердил намерение перейти на другую страницу, можно использовать класс 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 на экран окно сообщения и спрашивает пользователя, действительно ли он хочет покинуть страницу. Если пользователь нажмет кнопку отмена, переход будет отменен.

### Передача данных между страницами

Каждая страница Silverlight работает независимо других страниц, и данные каждой страницы приложения недоступны другим страницам. Но иногда нужно передавать данные из одной страницы в другую. Если данные являются простой строкой текста, самый простой способ передачи — добавить строку к URI целевой страницы. Этот метод похож на отправку запросов на веб-сервер.

В нашей программе для работы с клиентами при выборе пользователем клиента из списка нужно передать его имя и адрес на страницу редактирования.

private void customerList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

 // 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 метод вызывается, когда пользователь выбирает клиента. Метод получает информацию о выбранном клиенте и формирует URI целевой страницы, в который добавляет информацию о выбранном клиенте. Часть запроса после символа "?" состоит из двух записей вида имя=значение, разделенных символом "&".

Теперь рассмотрим, как целевая страница может получить и использовать переданную информацию. Для этого необходимо изучить события, возникающие при навигации по страницам Silverlight.

Использование событий при навигации по страницам. При переходе на страницу Silverlight происходит событие OnNavigatedTo, а при уходе со страницы — событиеOnNavigatingFrom. Для получения управления при возникновении этих событий нужно переопределить соответствующие методы в программном коде страницы.

Для того чтобы приложение получило управление при перемещении пользователя к странице, нужно переопределить OnNavigatedTo:

protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)

{

 string name, address;

 if (NavigationContext.QueryString.TryGetValue("name", out name))

 nameTextBlock.Text = name;

 if (NavigationContext.QueryString.TryGetValue("address", out address))

 addressTextBlock.Text = address;

}

Этот метод пытается получить из свойства QueryString объекта NavigationContext значения переданных в строке адреса параметров. Каждый раз, когда пользователь переходит к этой странице, этот метод вызывается и выводит на страницу переданные значения параметров. Можно использовать событие OnNavigatedTo для настройки страницы, когда пользователь переходит к ней. При переходе на страницу можно передавать любое количество параметров.

Если запрашиваемый на странице параметр отсутствует, то программа выполняется дальше без возникновения ошибки. Поскольку параметры передаются в виде обычного текста, нужно внимательно следить за тем, чтобы имена параметров при передаче данных и их считывании совпадали.

Приведенный выше код можно использовать при создании приложения, состоящего из двух страниц, которое позволяет пользователю при выборе клиента выполнить переход на страницу с подробной информацией об этом клиенте.

Однако в этом коде есть проблема, которая возникает при выполнении следующей последовательности действий:

1. Выбор элемента в списке.
2. Переход к странице с подробной информацией.
3. Нажатие кнопки Назад для возврата к списку.
4. Выбор того же самого элемента в списке.

При выполнении этой последовательности действий на странице не будет отображаться информация о том же самом клиенте. Для этого нужно выбрать другого клиента, и затем выбрать первого еще раз. Причина такого поведения состоит в том, что программа использует для перехода событие SelectionChanged, которое не происходит при повторном выборе того же самого клиента. Можно устранить эту проблему, сбрасывая выбор элемента при переходе к этой странице:

protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)

{

 customerList.SelectedItem = null;

}

Теперь при повторном выборе того же самого элемента он будет считаться "новым" выбранным элементом. Однако, установка значения выбранного элемента в null также расценивается как выбор элемента, и при попытке обращения к значению этого свойство возникнет исключение. Избежать этого можно, добавив в обработчик события проверку, был ли выбран элемент:

private void customerList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

 // отмена вызова, если элемент не выбран

 if (customerList.SelectedItem == null)

 return;

 // код метода

}

### Совместное использование объектов несколькими страницами

Часто требуется, чтобы страницы совместно использовали объекты, содержащие структурированные данные. В нашем приложении было бы удобнее передать на страницу подробной информации экземпляр объекта.

Можно легко предоставить совместный доступ к объектам для всех страниц приложения. Для этого используется страница App.xaml, которая автоматически создается при создании проекта приложения. Ее можно считать контейнером для приложения Windows Phone. Эта страница не выводится на экран, но файл App.xaml.cs содержит методы, которые выполняются при запуске приложения.

Можно изменить файл App.xaml.cs и добавить в него свой код. Если нужно хранить какие-либо глобальные данные, которые должны быть доступны всем страницам приложения, их объявления можно разместить здесь.

public partial class App : Application

{

 // объект используется другими страницами приложения

 public Customer ActiveCustomer;

}

Этот код содержит объявление объекта ActiveCustomer, который будет доступен всем страницам. Объект содержит ссылку на информацию о выбранном клиенте.

Любая страница Silverlight может получить ссылку на объект, размещенный на странице App:

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 является ссылкой, по которой можно обратиться к объектам в классе Application. При запуске приложения ей присваивается значение адреса текущей страницы.

Класс App.xaml.cs определяет дочерний класс App, который расширяет родительский класс Application. Чтобы получить объекты, определенные в классе App, нужно получить значение свойства Current класса Application и привести ее к типу App. Приведенный выше код запускается на странице customerDetailsPage, когда к ней происходит переход. При этом вызывается объект ActiveCustomer, который устанавливается в качестве контекста данных объекта customerDisplayGrid для обращения к выбранному клиенту.

Все объекты приложения, которые должны быть доступны другим страницам, имеет смысл помещать в класс App.xaml.cs. Здесь также можно добавить код, который при запуске программы будет загружать данные в эти объекты из хранилища.

## Использование классов ViewModel

На текущий момент наша программа может отображать информацию о клиенте, но не позволяет редактировать эту информацию. Пользователь может изменить текст в полях страницы, но изменения не будут сохраняться в программе.

Чтобы внесенные на странице изменения обновляли данные в списке клиентов, можно использовать привязку данных. При этом, привязка текста в элементе TextBox должна быть двусторонней, чтобы при изменении данных на экране приложения автоматически обновлялось связанное с элементом свойство:

<TextBox Name="nameTextBox" Text="{Binding Name, Mode=TwoWay}" />

После этого можно добавить код к классу Customer классу для управления связанными событиями. Этот код при изменении имени клиента должен изменяться текст на экране, и если текст элемента TextBox изменится, то связанное с элементом также должно изменяться автоматически. Но можно сделать это немного иначе, использовав модель, известную под названием Model—View—ViewModel.

### Создание класса ViewModel

Можно добавить привязку данных с классом Customer, но есть некоторые причины, почему это не является хорошей идеей. Существует правило проектирования программного обеспечения: обычно класс должен выполнять одно действие, и только одно. Класс Customer должен хранить данные о клиентах, но не должен участвовать в процессе редактирования. При хранении или передаче данных о клиентах мы не должны создавать методы, которые касаются редактирования данных. При передаче объекта, представляющего клиента, на другое устройство или при осуществлении привязки данных любое дополнительное свойство или метод будет лишней тратой пространства, так как не будет использоваться в этом контексте.

Возможность преобразования и проверки допустимости данных. В идеале, при вводе данных пользователем всегда должна осуществляться проверка соответствия вводимых данных определенным правилам. Например, должно соблюдаться правило: "Имя должно содержать только буквы и пробелы". Можно поместить эти правила в класс Customer, но будет намного проще указать их непосредственно в процессе ввода данных, так как они должны использоваться только при вводе данных. Также полезна возможность выполнять преобразование некоторых входных значений, например даты и времени, из текстового формата, в котором они вводятся и выводятся, в формат, в котором значения хранятся.

Возможность отмены действия. Практически всегда при проектировании формы для редактирования каких-либо значений рекомендуется предусмотреть возможность отмены сделанных пользователем, но еще не сохраненных изменений. Обычно в таких формах присутствуют кнопки сохранить и отмена. При использовании привязки данных это может оказаться проблемой, поскольку в этом случае объект данных сразу изменяется. Чтобы отследить изменения, внесенные пользователем, необходимо создать копию данных при открытии формы, и при отмене изменений использовать ее, что очень неудобно и требует создания дополнительного кода.

Создание класса ViewModel. Можно избежать этих проблем, создав класс ViewModel. Этот класс предназначен для представления данных и связывания с компонентами формы и также может выполнять необходимые проверки допустимости данных.

Создадим класс ViewModel для представления действий, которые нужно выполнить в пользовательском интерфейсе. Этот класс соединяет пользовательский интерфейс с объектами данных, содержит свойства, которые будут использоваться в пользовательском интерфейсе:

public class CustomerView : INotifyPropertyChanged

{

 private string name;

 public string Name

 {

 get

 {

 return name;

 }

 set

 {

 name = value;

 if (PropertyChanged != null)

 {

 PropertyChanged(this, new PropertyChangedEventArgs("name"));

 }

 }

 }

 // свойство Address создается таким же образом

 private int id;

 public int ID

 {

 get

 {

 return id;

 }

 }

 public event PropertyChangedEventHandler PropertyChanged;

 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;

 }

}

Этот класс генерирует события при изменении свойств в программе, и элементы Silverlight могут устанавливать новые значения свойств в классе при их изменении пользователем. Обратите внимание, что свойство ID доступно только для чтения, и пользователь не может изменить его значение. Это позволяет обеспечить уникальность идентификатора клиента.

Работа с данными в классе ViewModel. Класс CustomerView также содержит методы Load и Save, предназначенные для взаимодействия программы с содержимым класса ViewModel. Следующий код выполняется перед началом редактирования информации о клиенте:

// создание экземпляра класса ViewModel для страницы редактирования

CustomerView view = new CustomerView();

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;

}

Когда программа переходит на страницу для редактирования информации о клиенте, Silverlight вызывает метод OnNavigatedTo для настройки страницы. Этот метод получает информацию о выбранном клиенте и загружает ее в экземпляр класса CustomerView.

Когда пользователь нажимает кнопку сохранить, программа должна выгрузить содержимое класса ViewModel в редактируемое свойство Customer:

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();

}

### Навигация по страницам с использованием метода GoBack

Класс NavigationService содержит метод GoBack, который переносит пользователя к предыдущей странице. Это эффективнее, чем использование URI, потому что при этом страница не обновляется. Если перейти на страницу, используя метод Navigate класса NavigationService, будет создана новая страница, и она будет создаваться каждый раз при переходе на нее. Это может замедлить работу приложения и привести к созданию и освобождению большого количества элементов Silverlight. Кроме того, неудобно при каждом переходе на страницу со списком прокручивать ее вниз, поскольку в этом случае при переходе на страницу будет отображаться начало списка.

Однако, использование метода GoBack для перехода к предыдущей странице приводит к возникновению некоторых проблем. Трудность состоит в том, что предыдущая страница не будет содержать изменения данных, которые внесет пользователь. Например, пользователь выполняет следующие действия:

1. Выбор клиента из списка.
2. Изменение имени клиента.
3. Нажатие кнопки Сохранить для выполнения изменений.
4. Возврат к предыдущей странице.

После выполнения этих действий в списке будут содержаться первоначальные значения без учета сделанных пользователем изменений. Проблема состоит в том, что элемент ListBox не знает, что содержимое одной из записей в списке изменилось, поскольку при выводе списка на экран создается его копия, и при изменении данных эта копия обновляться не будет. Решение этой проблемы основано на использовании наблюдаемых коллекций ObservableCollection.

### Наблюдаемые коллекции

Класс ObservableCollection используется для хранения коллекции элементов и предоставляет поддержку уведомлений так, чтобы при изменении коллекции элемент ListBox можно было связать с событием изменения. Для создания наблюдаемой коллекции можно использовать имеющийся список:

ObservableCollection<Customer> observableCustomers;

private void PhoneApplicationPage\_Loaded(object sender, RoutedEventArgs e)

{

 // получить ссылку на страницу, содержащую список клиентов

 App thisApp = Application.Current as App;

 // создать наблюдаемую коллекцию этих клиентов

 observableCustomers = new ObservableCollection<Customer>(

 thisApp.ActiveCustomerList.CustomerList);

 // отображение элементов наблюдаемой коллекции

 customerList.ItemsSource = observableCustomers;

}

Этот код выполняется при запуске главной страницы приложения. Он создает новый список observableCustomers на основе имеющегося списка клиентов и устанавливает этот список в качестве источника данных элемента customerList. Теперь, при изменении значения элементов в списке эти изменения будут отображаться на экране.

Однако, если пользователь обновит имя клиента, это изменение не будет отражено на экране, потому что ObservableCollection реагирует на изменение содержимого списка, но не на изменение данных в элементе списка. Это является очень серьезным недостатком, приводящим к неправильной работе программы.

Чтобы исправить эту ошибку, нужно как-то внести изменения в список, чтобы он обновлялся на экране. Один из способов вызвать обновление заключается в том, чтобы удалить элемент из списка и снова его добавить. Изменения содержимого наблюдаемой коллекции генерируют события, которые обрабатывает элемент ListBox, который выводит список на экран.

protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)

{

 // получить ссылку на страницу, содержащую список клиентов

 App thisApp = Application.Current as App;

 if (thisApp.ActiveCustomer != null)

 {

 // найти нужного клиента в списке

 int pos = observableCustomers.IndexOf(thisApp.ActiveCustomer);

 // удалить клиента

 observableCustomers.RemoveAt(pos);

 // вернуть клиента назад

 observableCustomers.Insert(pos, thisApp.ActiveCustomer);

 }

}

Этот код выполняется, когда пользователь переходит назад к главной странице. Если выбран какой-нибудь клиент, метод удаляет этого клиента из списка и затем добавляет его на то же самое место. Это вызывает изменение списка, и пользователь будет видеть сделанные изменения. При обновлении всего одного элемента обновление будет выполняться довольно быстро. Это особенно важно, если список содержит несколько тысяч элементов.

При использовании класса ViewModel для редактирования информации о клиенте мы создавали методы для загрузки и сохранения информации, которые заносили информацию о клиенте в класс CustomerView и затем возвращали обновленные значения. При использовании класса ObservableCollection необходимо сделать что-то похожее. К счастью, существует много способов получения списка данных из класса ObservableCollection. Например, можно использовать расширяющий метод ToList:

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

Использование наблюдаемых коллекций не приводят к повышению расхода памяти, поскольку содержат набор ссылок на уже существующие объекты. Соответственно, требуется лишь небольшой объем памяти для хранения дополнительных ссылок на объекты. Однако наблюдаемые коллекции не имеет смысла использовать, если программа только выводит список на экран, но не изменяет его.

## Краткие итоги

1. Программы могут изменять значения свойств элементов Silverlight для управления отображением элементов на экране. К числу свойств относятся положение элемента на экране и его цвет.
2. Большинство свойств элементов удобнее задавать, редактируя напрямую XAML-код. XAML-описание элемента имеет определенную структуру, в которой одни свойства могут вкладываться в другие.
3. В элементе TextBox есть набор свойств, которые задают тип клавиатуры, которая будет использоваться для ввода данных. Можно использовать для ввода специальную клавиатуру для ввода чисел вместо клавиатуры для ввода произвольного текста.
4. Windows Phone может выводить на экран окно с сообщением для пользователя. Текст сообщения может состоять из нескольких строк. Сообщения могут также использоваться для подтверждения или отмены действий пользователя.
5. Ресурсы могут быть добавлены в приложение для Windows Phone как часть содержимого или как внедренные ресурсы. Элемент содержимого копируется в каталог приложения в отдельный файл, который может использоваться программой. Внедренный ресурс встраивается в файл сборки приложения. Элементы содержимого не замедляют загрузку сборки, но могут медленнее загружаться во время работы программы. Внедренные ресурсы загружаются быстрее, но увеличивают размер сборки программы.
6. Элементы Silverlight могут генерировать события в ответ на действия пользователя. Например, событие TextChanged элемента TextBox происходит при изменении текста элемента.
7. Silverlight предоставляет привязку данных, с помощью которой свойства программного объекта могут быть соединены со свойствами визуального элемента Silverlight. Привязка может быть однонаправленной, когда визуальный элемент используется только для вывода значения объекта программы, или двунаправленной, когда изменение свойства визуального элемента приводит к обновлению связанного свойства класса.
8. Можно связать коллекцию элементов с элементом ListBox для отображения списка элементов. Шаблон данных используется для описания того, как отдельные свойства каждого элемента должны выводиться на экран.
9. Приложения Silverlight могут состоять из нескольких страниц. Навигация по страницам выполняется в классе Navigate, который использует URI страницы. Простые строковые значения можно передавать между страницами, разместив их в строке запроса, присоединенной к URI.
10. Страницы могут получать события при перемещении к ним и от них. Событие OnNavigatedFrom предоставляет возможность отмены перехода на другую страницу, если нужно чтобы пользователь подтвердил переход.
11. Большие объекты данных могут совместно использовать несколько страниц приложения с помощью класса App, который является частью приложения Windows Phone. Любая страница приложения Silverlight может получить ссылку на объект приложения, который является частью этого приложения.
12. Программисты могут создать классы ViewModel, которые связывают редактируемые данные с элементами на странице Silverlight.
13. Механизм ObservableCollection позволяет отразить на экране изменения в коллекции элементов.