Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Комсомольский-на-Амуре государственный технический университет»

В. А. Тихомиров

РАЗРАБОТКА ПРОСТЕЙШИХ ПРИЛОЖЕНИЙ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ

Утверждено в качестве учебного пособия
Ученым советом Федерального государственного бюджетного
образовательного учреждения высшего профессионального образования
«Комсомольский-на-Амуре государственный технический университет»

Комсомольск-на-Амуре
2013

УДК 004.4(07)
ББК 32.973.2-018.217я7
Т462

Рецензенты:

А. Н. Тачалов, кандидат технических наук, доцент, ведущий инженер АСУ ТП ОАО «Дальневосточная генерирующая компания» филиал «Хабаровская генерация» СП «Комсомольская ТЭЦ-3»;
Кафедра информационных систем, компьютерных технологий и физики ФГБОУ ВПО «Амурский гуманитарно-педагогический государственный университет», руководитель научного коллектива кандидат физико-математических наук М. А. Савунов

Тихомиров, В. А.

Т462 Разработка простейших приложений для мобильных устройств : учеб. пособие / В. А. Тихомиров. – Комсомольск-на-Амуре : ФГБОУ ВПО «КнАГТУ», 2013. – 134 с.

ISBN 978-5-7765-1046-5

В учебном пособии рассмотрены вопросы разработки приложений для мобильных устройств, таких как сотовые телефоны, смартфоны и карманные персональные компьютеры. Проведены анализ и сравнение большинства современных операционных систем мобильных устройств и инструментария для разработки в них программных продуктов. Рассмотрены начальные вопросы создания программного обеспечения в операционной системе Windows Mobile на платформах Silverlight и XNA. Приведены задания для контрольных работ по дисциплине «Программирование мобильных устройств».

Учебное пособие предназначено для студентов направления 230100 – «Информатика и вычислительная техника», а также для студентов других направлений, изучающих дисциплины, связанные с разработкой программного обеспечения для мобильных устройств.

УДК 004.4(07)
ББК 32.973.2-018.217я7

ISBN 978-5-7765-1046-5

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Множество устройств можно отнести к мобильным программируемым устройствам. В принципе это любая программируемая компьютерная техника, которую человек может перетащить с собой в дипломате. В данном учебном пособии мы будем рассматривать:

- сотовые телефоны;
- смартфоны;
- коммуникаторы;
- КПК (карманные персональные компьютеры).

Сотовые телефоны.

Сотовый телефон оснащен прошивкой, выполняющей роль примитивной операционной системы (ОС), компоненты которой простой смертный, не знающий языка программирования, вряд ли сможет изменить. Однако большинство современных аппаратов помимо неизменяемой прошивки имеют в своем арсенале программную платформу Java2ME (Java 2 Micro Edition), которая позволяет закачивать на телефон приложения, написанные на языке Java, в том числе и игры. Скорее всего, по мере удешевления смартфонов сотовые телефоны, а вместе с ними и платформа Java2ME канут в лету. Однако на данный момент в Интернете можно найти довольно большое количество приложений, которые для неё подходят. Кроме этой платформы, существуют ещё две: Mophun (живет только в старых телефонах Sony Ericsson, например SonyEricssonT610) и BREW. Последняя увидела свет позже, чем Java2ME (в 2001 г.) и изначально предназначалась для CDMA-телефонов, затем была адаптирована и для телефонов стандарта GSM, но широкого распространения пока так и не получила, по крайней мере в России.

Смартфоны.

Название «смартфон» (рис. 1) произошло от двух английских слов: «*Smart*», что в переводе означает «умный», и «*Phone*» – телефон. То есть смартфон – это умный телефон. А какой телефон можно считать «умным»? Разумеется, клиента электронной почты, WAP-браузера и «продвинутого» редактора рингтонов тут недостаточно. Речь идет о других, ***компьютерных*** функциях аппарата. То есть у него должна быть настоящая операционная система, большой дисплей, Bluetooth и/или инфракрасный порт. Также «ум» телефона подчеркивают возможность синхронизации с ПК, наличие слотов расширения и достаточный объем памяти для установки приложений.

От стоящих ниже на лестнице эволюции телефонов смартфонам достались ***телефонная клавиатура*** и ***отсутствие сенсорного дисплея***.

Рис. 1. Разновидности смартфонов

Коммуникатор.

Коммуникатор – это карманный компьютер с функциями телефона. Это значит, что в устройстве зачастую не бывает телефонной клавиатуры (рис. 2), в обязательном порядке имеются полноценная ОС и сенсорный экран. Многие модели коммуникаторов делаются очень просто – берется серийная модель смартфона, добавляется GSM-модуль и выводится на рынок в качестве новой модели с ценой сотни на полторы больше, чем у «родителя».

Рис. 2. Разновидности коммуникаторов

От этого, правда, часто страдает эргономика, которая у большинства коммуникаторов и так не блещет. Либо получается начисто лишенная всех удобств, но крайне функциональная «коробка», либо непроизводительная и нефункциональная, но очень красивая поделка. «Золотая середина» все же иногда встречается. Общепризнанным эталоном коммуникатора является Sony Ericsson P900/910 (рис. 3). Но за удобство приходится платить.

Удобству использования коммуникаторов сильно способствуют разные аксессуары, количество которых намного больше, чем у смартфонов. При помощи двух-трех дополнительных аксессуаров можно превратить коммуникатор в машину, ничем по удобству не уступающую ноутбуку. Тем, кому необходимо писать большие тексты или заполнять огромные отчеты, наверняка поможет дополнительная клавиатура. Есть даже резиновые клавиатуры, которые при желании можно свернуть в трубочку. А если созданные на коммуникаторе работы захочется напечатать минуя настольный компьютер – тоже нет проблем. Благо для мобильных устройств выпущено немало компактных и удобных принтеров. Так что владелец коммуникатора вполне может начать работать, едва выйдя из дома.

Рис. 3. Разновидности коммуникаторов с клавиатурами

Карманный персональный компьютер (КПК).

КПК – это карманные персональные компьютеры со своей памятью, процессором, слотами расширения, звуковой системой, ну и, конечно же, дисплеем (рис. 4). КПК от своих старших собратьев отличаются меньшим дисплеем, который к тому же реагирует на прикосновение специальной палочки (стилуса) и flash-памятью, которая, в отличие от винчестера, занимает меньше физического места.

Чаще всего КПК сравнивают по функциональности с ноутбуками, хотя это совершенно разные устройства, предназначенные для использования в разных целях. Основное различие их в том, что для работы с ноутбу-

ком Вам, прежде всего, нужно где-то его поставить и самому принять удобное положение, а также дождаться загрузки операционной системы. С КПК все гораздо проще. В метро, трамвае, сидя или стоя, Вы запросто включаете его и, управляя стилусом либо цифровой клавиатурой (если есть), спокойно работатете. Процесс включения занимает считанные секунды.

Рис. 4. Разновидности КПК

Также немаловажно то, что при работе с ноутбуком Вы пользуетесь клавиатурой для набора текстов, а в КПК для этого используется стилус. Для набора текста используется «виртуальная клавиатура». Также можно писать слова от руки, как Вы это делаете ручкой на бумаге. С первого взгляда это не очень удобно, но, потренировавшись, Вы сможете вполне быстро набирать небольшие заметки или статьи. При желании никто не мешает купить дополнительную клавиатуру, которая будет соединяться с Вашим КПК по Bluetooth или через USB-порт.

Раньше считалось, что КПК из-за слабых процессоров и малого наличия памяти могут использоваться только для набора текстов, прослушивания музыки и простеньких игр. На сегодняшний день современные КПК обладают высокоскоростными процессорами по 400, 520 МГц и выше, что вполне достаточно для просмотра несжатого видео в формате DivX, 3D игр типа Doom, работы с базами данных и большими электронными таблицами. Память тоже можно расширить, купив дополнительную карту на 1, 2 ГБ или больше, благо такие карты дешевеют с каждым днем.

От смартфонов КПК отличаются только тем, что в них не вставляется SIM-карта и в них нет телефонной связи. В противовес аппаратные мощности КПК (процессор, память, размер экрана и т.д.) существенно выше.

1. ОБЗОР ОПЕРАЦИОННЫХ СИСТЕМ МОБИЛЬНЫХ УСТРОЙСТВ

В сотовых телефонах, как говорилось выше, нет ОС как таковой, там прошивка, на которую «натянута» виртуальная Java-машина, исполняющая программы, называемые «мидлетами».

Настоящие ОС начинаются со смартфонов. Смартфоны и коммуникаторы имеют возможность устанавливать дополнительное программное обеспечение (ПО), зачастую от сторонних разработчиков, для добавления новых возможностей и расширения функциональности.

В коммуникаторах и смартфонах широкое распространение получили операционные системы:

1. Symbian OS;
2. Windows Mobile;
3. Palm OS;
4. iPhone OS;
5. BlackBerry OS;
6. Samsung Bada;
7. Системы на базе Linux:

- Google Android;
- Palm webOS;
- Access Linux Platform;
- Nokia Maemo.

Кроме ОС существуют еще достаточно интересные приложения, дополняющие саму ОС, расширяющие ее функциональность и меняющие внешний вид. Как пример можно вспомнить TouchFLO 3D для коммуникаторов HTC или фирменный интерфейс TouchWIZ, используемый в мобильных устройствах Samsung.

1.1. Palm OS

Система Palm OS достаточно редкая. Из различных околокомпьютерных СМИ мы слышим, что у Palm «не все в порядке». И это не удивительно, учитывая, что сейчас не каждый сведущий в ОС человек даст однозначный ответ на вопрос: «Кому принадлежат права на Palm OS?»

Palm OS Garnet принадлежит ACCESS, но Palm Inc купила у ACCESS «пожизненное право» на исходный код Palm OS 5.4 Garnet, а это значит, что она имеет право разрабатывать свои продукты на этой основе. Также заявлена и шестая версия системы, но под ее «парусами» еще не работает ни одно устройство.

Рис. 1.1. Дизайн Palm OS

Несмотря на все проблемы, число «пальмоводов» более чем внушительно, а значит, эту ОС рано сбрасывать со счетов. Что большинству пользователей нужно от КПК? Максимальное использование дисплея, честная надежность, мультимедийность, безболезненная синхронизация с ПК, приличное время работы без подзарядки. Все это есть в устройствах на основе Palm OS (рис. 1.1). Плюсов много, а минусы…

Вообще разработчики сейчас стремятся вперед, множат плюсы, во многом забывая о минусах. Болезнь Palm OS еще с детства – это отсутствие нормальной многозадачности, т.е. одновременного выполнения нескольких приложений. Иными словами многозадачность здесь реализована по шаблону «почувствуй себя пользователем мобильного телефона»: запустив одно приложение Вы не сможете запустить параллельно другое. К тому же сложно положительно охарактеризовать такое собирательное понятие, как мультимедийность, говоря о его реализации в Palm OS.

Достоинства:

- нетребовательна к ресурсам;
- очень удобный интерфейс пользователя;
- удобная синхронизация с ПК;
- надежность.

Недостатки:

- отсутствует полноценная многозадачность;
- не развиты мультимедийные функции;
- система не развивается (хотя, возможно, компания HP сможет это преодолеть).

1.2. symbian Symbian OS

До последнего времени, это самая распространенная ОС для смартфонов. По прогнозу аналитиков из компании «Garnter», в 2012 г. Symbian все еще будет самой распространенной ОС для смартфонов, однако ее доля уменьшится с почти 50 до 39 % (на рис. 1.2 – Samsung i8910 Omnia HD).

Symbian OS изначально создавалась исключительно для смартфонов, прототипом для нее послужила операционная система EPOC 32. Впервые эта ОС от компании Psion – одного из пионеров рынка КПК – была

использована в КПК Psion Series 5 в 1997 г. Она была создана для работы с процессорами ARM, традиционными для мобильных устройств, и обеспечивала работу с клавиатуры и через сенсорный экран. Несомненным плюсом в пользу новой системы стало разграничение графического интерфейса и другого ПО – это позволило адаптировать систему под устройства с любыми характеристиками экрана и клавиатуры, а производителям – использовать разные интерфейсы. Программное обеспечение EPOC 32 также отличалось компактностью, что позволяло без проблем использовать его на устройствах с ограниченными ресурсами. EPOC32 Release 5u (Symbian OS v5) была оптимизирована для работы с коммуникационными протоколами (например, с протоколами сотовой связи). В ней появилась возможность работы с Интернетом и почтой, обработки файловых вложений и SMS. На ее основе работал популярный смартфон начала 2000-х гг. – Ericsson R380.

Рис. 1.2. Дизайн Symbian OS

Самыми распространенными платформами на базе Symbian сейчас являются: Series 60 (самая популярная платформа, используется на моделях Nokia, Panasonic, Samsung, Lenovo, LG, Sony Ericsson и др.), Series 80 (использовалась в некоторых моделях Nokia), Series 90 (сейчас используется только на Nokia 7710), UIQ (модели Nokia, Benq, Motorola, Arima, Sony Erisson) и MOAP (закрытая платформа, устанавливается на телефонах Fujitsu, Sony Ericsson, Mitsubishi и Sharp).

Series 60 создавалась для смартфонов с телефонной клавиатурой, в ОС закладывалась поддержка экранов высокого разрешения, управление приложениями производилость через сенсорный ввод.

Разработкой и продвижением данной ОС занимается некоммерческая организация Symbian Foundation, созданная в 1998 г., в ее состав входят 40 компаний, среди которых Samsung, Nokia, LG Electronics, Sharp, Sony Ericsson, Huawei, Motorola, Siemens, Panasonic, Fujitsu, Sony, Sanyo, Ericsson, AT&T, Psion, STMicroelectronics, Texas Instruments и др.

Программ, предназначенных для Symbian OS, существует огромное количество; их можно узнать по расширению SIS. Файлы в формате SIS представляют собой самораспаковывающийся архив. Работоспособность Symbian вызывает только положительные отклики. Даже на смартфонах с ограниченными ресурсами система работает без сбоев и зависаний. Очень хорошо реализована многозадачность.

Достоинства:

- низкие требования к памяти и процессору;

- функция освобождения неиспользуемой памяти;
- стабильность;
- малое количество вирусов для этой платформы;
- быстро выходят новые версии и исправляются нестабильности;
- большое количество программ.

Недостатки:

- для связи с ПК нужно устанавливать дополнительный софт;
- несовместимость программ для старых и новых версий.

1.3. Windows Mobile

Рис. 1.3. Дизайн Windows Mobile

Windows Mobile – общее название нескольких вариантов ОС для мобильных устройств – на сегодня является сильнейшим решением в своем роде, для которого выпущено немалое количество программного обеспечения (на рис. 1.3 – Samsung i8000 Witu AMOLED).

Главные преимущества Windows Mobile: привычный по настольным ПК интерфейс, хорошая реализация многозадачности, поддержка аппаратов с высоким разрешением экрана, большое разнообразие моделей смартфонов, обилие программного обеспечения на любой вкус и под любые задачи.

Сегодняшнее заполнение рынка коммуникаторов Windows Mobile позволяет пользователю практически в любом ценовом диапазоне найти достойные модели.

Среди программ для устройств в Windows Mobile есть и софт, хорошо знакомый по настольным ПК. С 5-й версии ОС появилась новая версия набора офисных программ Office Mobile: это знакомые Word, Excel, Outlook, Internet Explorer и Windows Media Player.

С 2012 г в устройствах используется версия Windows Mobile 6.5, но также имеется информация о Windows Mobile 7.0 и 7.1. Одной из основных характеристик новой системы является поддержка управления в режиме multitouch и интегрированное веб-приложение Silverlight для обработки мультимедиа. Доработан также режим блокировки устройства: теперь на экран телефона выводится намного больше информации: о новых SMS, вызовах, времени и дате. Также были оптимизированы выпадающие меню,

установлена более легкая в управлении обновленная версия Internet Explorer Mobile.

Стоит отметить, что компания Microsoft представила бета-версию Microsoft Office Mobile. Пакет включает в себя Word Mobile, Excel Mobile и PowerPoint Mobile. В нем реализована поддержка онлайн-приложений, кроме того, улучшены характеристики отдельных мобильных приложений: в Outlook Mobile реализована возможность отображения переписки в виде разговора и группировка связанных между собой писем, в Excel Mobile есть возможность масштабировать таблицы и разворачивать их на весь экран, также с помощью Mobile Document Viewer можно проще работать с файлами, расположенными на удаленных серверах.

Достоинства:

- схожесть с настольной версией;
- удобная синхронизация;
- в комплекте идут офисные программы;
- многозадачность.

Недостатки:

- высокие требования к оборудованию;
- наличие большого числа вирусов;
- нестабильности в работе.

1.4. Android

Android – одна из самых молодых мобильных ОС, основанная на операционной системе Linux, платформа для мобильных телефонов, разрабатываемая Open Handset Alliance (OHA). Разработка инициирована компанией Google (на рис. 1.4 – Samsung i5700 Galaxy Spica).

Рис. 1.4. Дизайн Android

Основным преимуществом Android по сравнению с другими ОС является практически полностью открытая архитектура и глубокая интеграция с сервисами Google. Уже сейчас существует достаточно большое количество программного обеспечения для этой ОС, чтобы практически любой пользователь не почувствовал себя обделенным, при этом рынок ПО развивается достаточно динамично.

Работать с этой ОС удобно, она отлично подходит именно для таких устройств, как коммуникаторы, самый большой плюс которых – использование онлайновых сервисов. Прогноз погоды, котировки акций, новости – со всем этим вы можете ознакомиться посредством вашего мобильного

устройства, плюс ко всему удобно реализована работа с социальными сетями, файловыми сервисами.

1.5. BlackBerry OS

Рис. 1.5. Дизайн BlackBerry OS

BlackBerry – это торговая марка беспроводного ручного устройства, которое было впервые представлено в 1997 г. компанией Research In Motion. Основная функция – мгновенное корпоративное общение. Главное отличие смартфона BlackBerry – это моментальная синхронизация с корпоративным почтовым сервером. Пользователь получает почту на смартфон непосредственно в момент ее поступления на корпоративный почтовый ящик. При этом обеспечена надежная защита данных с помощью уникальной системы шифрования. Объем передаваемого траффика минимален, что актуально в роуминге. Серверы, через которые предоставляется сервис защищенной почты, находятся в Америке и Англии. Этими смартфонами пользуются в основном крупные компании, т.к. удовольствие не дешевое. По поводу исключительной защищенности есть и ложка дегтя – несмотря на все усилия производителя, многие спецслужбы получили коды шифрования от этой системы.

Основной функционал операционной системы Blackberry OS заточен под офисного, бизнес-пользователя (рис. 1.5). Полная QWERTY-клавиатура, специально спроектированная для набора большими пальцами рук, «прокрутка» содержимого экрана и возможность копировать из других сообщений и из Интернета делают написание писем особенно быстрым и удобным. Стандартные приложения для смартфонов BlackBerry позволяют просматривать вложенные файлы большинства основных форматов и работать с ними.

В «Приложениях» находятся текстовый и графический редакторы, редактор презентаций, полностью совместимые с настольными офисными программами. Это ***весомое конкурентное преимущество***, т.к. в других ОС офисные программы необходимо приобретать, они занимают оперативную память (не встроены в ОС) и не всегда стабильно работают. Планшет BlackBerry PlayBook позиционируется как первая модель, созданная специально для корпоративных пользователей, сочетающая в себе полную многозадачность и высокую производительность при работе с мультимедиа. Операционная система BlackBerry Tablet OS была создана специально для планшетных компьютеров.

Новый браузер на основе Webkit открытого движка (как и конкуренты Safari, Google Chrome и др.) поддерживает увеличение отдельных участков страниц при помощи жестов и одновременную работу нескольких сессий (при помощи закладок), а также обладает высокой эффективностью, т.е. для его работы требуется меньший объем загружаемых данных.

При создании операционной системы BlackBerry были сохранены все преимущества платформы BlackBerry, и в дополнение к функционалу для работы в Интернет добавили в состав новой ОС множество новых мультимедийных приложений, простых и удобных в использовании и интегрированных с другими функциями смартфона. Кроме того, в новой версии ОС появились такие функции, как Ленты новостей социальных сетей (Social Feeds) и Универсальный поиск (Universal Search), которые дополнительно расширяют и без того богатый спектр возможностей для общения.

1.6. iPhone OS

Apple iOS (ранее называвшаяся iPhone OS**)** – ОС, разработанная компанией Apple на основе стационарной Mac OS X для мобильных устройств iPhone, iPod Touch, iPad.

Сегодня это лидер рынка во многих странах, но в Азии и Европе, как и в России, все еще сильны позиции устаревающей Symbian, а в США в спину дышит молодая поросль Android и офисная Blackberry.

iOS – это полностью закрытая платформа, которая заставляет пользователя приобретать ее продукцию (как софт, так и хард), включая многочисленные аксессуары, а также контент и предоставляющая взамен простоту пользования, дружелюбный интерфейс, работоспособность и прочие «добродетели» (рис. 1.6).

Рис. 1.6. Дизайн iPhone OS

Достоинства:

- удобство пользования;
- качественная служба поддержки;
- регулярные обновления, устраняющие многие проблемы в работе;
- возможность купить в App Store множество различных программ.

Недостатки:

- необходимость джайлбрейка для установки неофициальных приложений;
- заблокированный характер ОС;
- отсутствие многозадачности;
- нет встроенного редактора документов.

1.7. bada Bada

Рис. 1.7. Дизайн Bada

Bada – собственная система компании Samsung. Она была представлена в феврале 2010 г., а первое устройство на этой ОС – Samsung 8500 Wave – было очень успешным рынке (рис. 1.7). Легко различить у компании Samsung устройства с различными ОС: с операционной системой Bada (морская тематика) называются Wave (Волна), с Android (космическая тематика) – Galaxy (Галактика).

Bada – это скорее мобильная платформа, а не полноценная ОС.

В смысле развития собственной экосистемы Samsung идет по стопам Apple, копируя их решения, которые даже внешне похожи на «яблочные»:

- Книжный магазин содержит 60 000 книг и продолжает развиваться (клон Apple Bookstore);
- Сервис Dive позволяет найти телефон с помощью определения местоположения и закрыть к нему доступ или стереть информацию;
- Socilal Hub позволяет систематизировать работу с социальными сетями, обьединяя контакты, календарь и информацию, поступающую от них, в единый поток данных, который пользователь получает непрерывно с помощью push-технологий на свое мобильное устройство.

1.8. TouchWiz от Samsung

Рис. 1.8. Дизайн Samsung

Пользовательский интерфейс TouchWiz (модели Samsung SGH-F480 TouchWiz, Samsung s8000 Jet, Samsung WiTu, Samsung M8800 Pixon) появился в результате эволюции интерфейса Croix (на рис. 1.8 – Samsung s8000 Jet).

Последняя версия – 2.0 – более «объемна» по дизайну и унифицирует то, как выглядят на экране различные платформы (Windows Mobile, Symbian, Android), а также организует рабочий стол в так называемый мультимедийный куб (кубический шестисторонний рабочий стол). В последней версии есть три панели для виджетов, которые можно перетягивать по экрану простым перемещением и вытягивать из боковой панели простым движением пальца. Одним движением можно настроить и сам экран (например, выбрать

обои, раскрыв Home Screen Customizer), проскроллить основные пункты меню, создать сообщение и т.д.

В TouchWiz 2.0 также поддерживается акселерометр и приложение разблокировки, которое дает быстрый доступ к некоторым апплетам в заблокированном режиме.

1.9. Обзор инструментов разработчика приложений для мобильных устройств

Есть два принципиально различных типа программ для мобильных устройств: самостоятельные приложения и исполняемые файлы, которые запускаются только при наличии установленной в устройстве специальной среды – интерпретатора.

В первом случае для «перевода» текста программы на язык, понятный какой-либо платформе (операционной системе), необходим компилятор – специальное приложение, которое, как правило, входит в состав средств разработчика. Пропускаем написанный код через компилятор и на выходе получаем самостоятельное приложение для совместимой платформы. Достаточно скопировать его на соответствующий аппарат и элементарно запустить. Поясним: в случае с обычной Windows XP компилятор выдает EXE-файл. Все, что требуется от пользователя для запуска, это двойной клик. Компилируемые языки программирования в освоении сложны, зато творческих возможностей предоставляют больше. C++, например, это стандарт де-факто при разработке ПО, в том числе и для многих мобильных платформ.

Первый метод создания программ отличается инструментами (для каждой ОС – свои), и файлы, созданные в этих инструментах, запускаются только на тех платформах, для которых они созданы.

Во втором случае интерпретатор занимается тем, что объясняет данному устройству, как следует выполнять код программы. Пожалуй, самый известный пример интерпретатора – виртуальная машина Java, которая, кстати, по умолчанию наличествует не только в смартфонах, но и практически в любых современных телефонах. Интерпретатор Java универсален. Одна и та же Java-программа, как правило, выполняется и на коммуникаторе Windows Mobile, и на каком-нибудь музыкальном телефоне Sony Ericsson.

Существуют интерпретаторы для мобильных приложений, написанных на языках Python, mShell (создан фирмой infowing AG (www.mshell.net)) и Basic, хотя эти интерпретаторы скорее экзотика, чем норма.

Минусы интерпретаторов в относительно медленной скорости работы, а кроме того, они обладают изрядным аппетитом в плане потребления

ресурсов. Зато такие языки просты для изучения, и инструменты для их создания носят универсальный характер, и созданные программы работают на всех платформах одинаково.

Таким образом, в зависимости от типа (исполнения) программного обеспечения для мобильных устройств можно выделить следующие классы инструментария программиста:

1. инструменты для разработки «мидлетов» – программ, выполняемых на виртуальных Java-машинах мобильных устройств (или программ для других интерпретаторов);

2. инструменты для создания специализированного программного обеспечения под одну из мобильных ОС.

1.9.1. Инструменты для разработки «мидлетов»

На данный момент почти все выпускаемые мобильные устройства имеют предустановленную возможность для запуска Java-программ (мидлетов). Большая распространенность этой технологии привлекает внимание разработчиков коммерческих продуктов (особенно игр), но и обычный пользователь может сделать что-нибудь свое.

Базовый язык для разработки программ под Java-интерпретатор («мидлетов») – Java ME. Чтобы вести программирование по этой технологии, необходимо создать у себя на компьютере специальную среду разработки. Основу этой среды составляет Java ME SDK – специальный комплект средств разработки. В настоящее время существует несколько различных версий SDK от разных производителей, их использование позволяет создавать мобильные приложения, заточенные под определенные телефоны и мобильные платформы. Соответственно, доступные программисту JSR-расширения и функциональные возможности среды разработки будут сильно зависеть от выбранного SDK. Наиболее распространенные Java ME SDK программиста следующие:

- Sun Java ME SDK 3.0;
- NetBeans 6.5 IDE;
- MOTODEV Studio for Java ME;
- Nokia S60 SDK;
- Nokia S40 SDK;
- Nokia NFC SDK;
- BlackBerry JDE 4.7;
- Sony Ericsson SDK 2.5 for Java ME;
- LG SDK 1.2 for Java ME.

Кроме того, для разработки «мидлетов» применяются специальные интегрированные среды, например, MIDletPascal.

Sun Java ME SDK 3.0 стала де-факто стандартом на рынке мобильных программ. Java ME SDK – кульминация проекта Java Wireless Toolkit. J2ME SDK поддерживает следующие JVM платформы:

- CLDC/MIDP: общая JVM-конфигурация для мобильных телефонов.
- CDC/FP/PBP/AGUI: JVM-конфигурация для high-end смартфонов.
- CDC/FP/PBP/BD-J: JVM-конфигурация для Blu-ray Disc плееров.

Java ME SDK – одна из нескольких доступных SDK, ориентированных на некое гипотетические устройство, что дает возможность разрабатывать и отлаживать мобильные приложения перед «заточкой» их под конкретную мобильную платформу. SDK содержит Platform Manager, который позволяет эмулировать конкретную платформу. На рис. 1.9 показан Java ME SDK 3.0 с запущенным эмулятором JavaFX телефона.

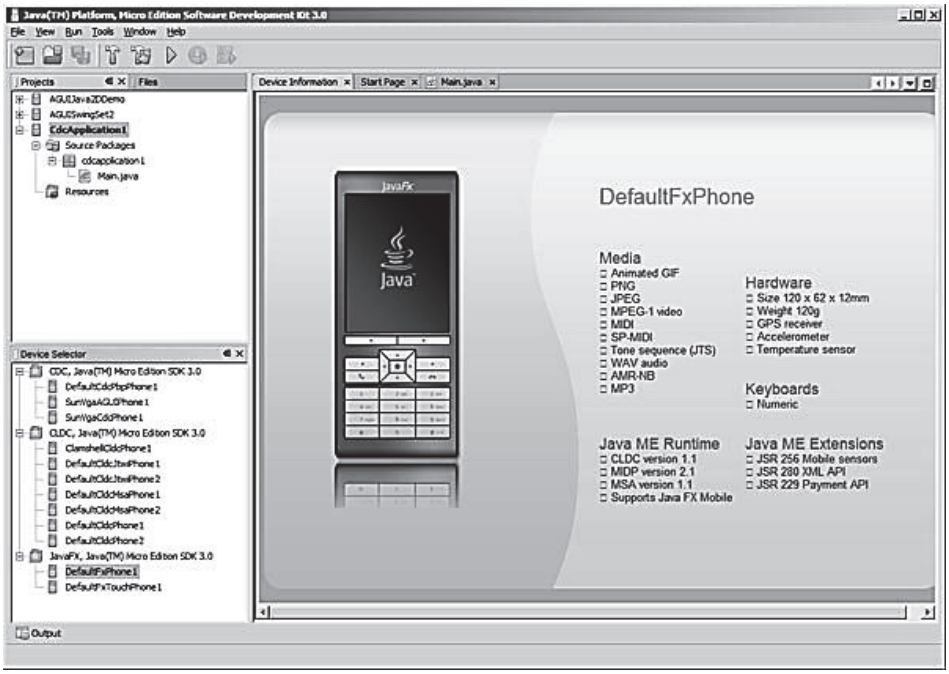


Рис. 1.9. Вид среды разработки Sun Java ME SDK 3.0

В отличие от Java Wireless Toolkit **Java ME SDK** содержит IDE, и Вы можете разрабатывать и тестировать свои приложения в этой среде. Нужно отметить, что Java ME SDK не поддерживает разработку JavaFX-приложений, однако он содержит несколько эмуляторов JavaFX 1.1 телефонов (один с тачскрином и один – без), которые позволяют запускать и тестировать JavaFX Mobile приложения. Для создания JavaFX Mobile приложений можно использовать NetBeans IDE. Основным отличием Java ME SDK 3.0 от предыдущих версий является процесс конфигурирования SDK для Blu-ray разработки. Последняя сборка содержит BD-J библиотеки. Таким образом устранены преграды, стоявшие перед разработчиками BD-J приложений.

Одной из главных особенностей Java ME SDK 3.0 является возможность пошагово отлаживать приложения на реальном мобильном устройстве. Данная возможность пока доступна только для Windows Mobile 6 устройств.

MOTODEV Studio for Java ME – еще одна Java ME SDK, ориентированная на Motorola устройства и имеющая ряд дополнительных сервисов (рис. 1.10):

- Bluetooth Service;
- Landmark Storage;
- Location Service;
- Remote Control (Bluetooth);
- SIM Configuration;
- SIP Proxy;
- WMA Server.

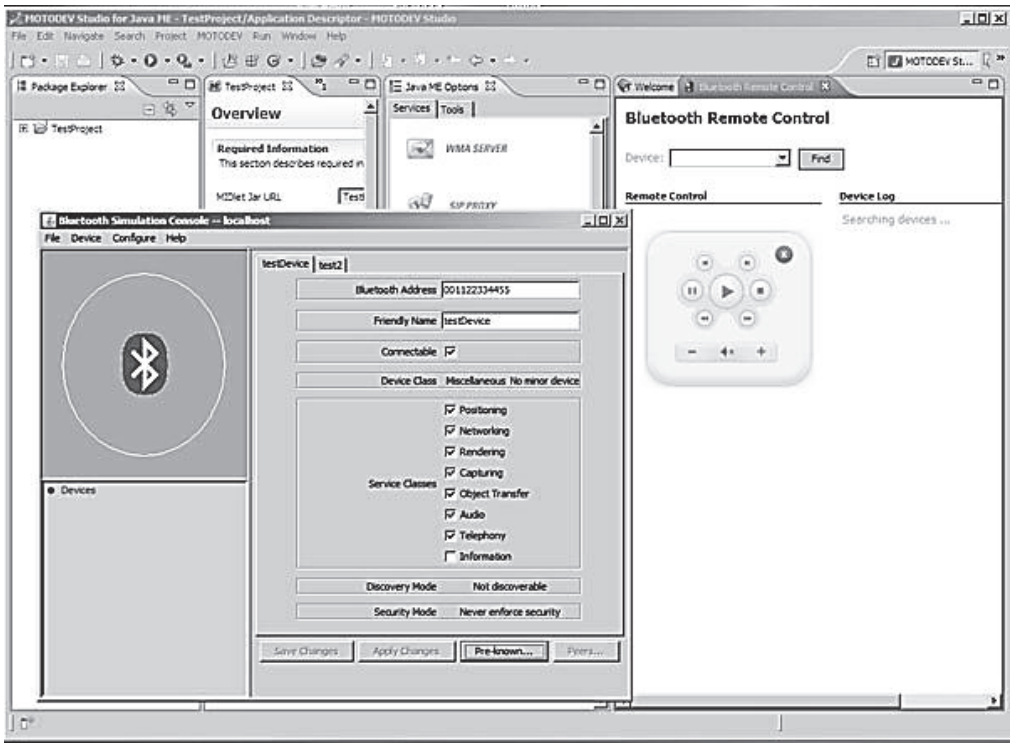


Рис. 1.10. Вид среды разработки MOTODEV Studio for Java ME

Эти сервисы позволяют Вам симулировать реальные события без необходимости отладки на реальном устройстве. Например, Bluetooth Service содержит Rococo Bluetooth симулятор, который позволяет симулировать Bluetooth устройства в MOTODEV Studio.

MOTODEV Studio отлично подходит для разработки приложений, ориентированных на Motorola устройства. Вы можете отлаживать приложения на реальных устройствах, подключив их по USB.

Nokia S60, S40, NFC SDK

Nokia предлагает программистам три SDK для разработки мобильных приложений. В состав **SDK** входят различные утилиты, например SVG (SVG-Tiny конвертер), который может быть очень полезным, если Вы планируете использовать JSR 226 API для отображения векторной графики. Как и рассмотренные выше SDK, **S60 SDK** позволяет проводить отладку приложений на реальных устройствах, однако он имеет особенность: позволяет перенаправлять System.out и System.err сообщения.

S40 SDK включает Nokia Connectivity Framework, который позволяет эмулировать Bluetoth и SMS-сообщения.

Если Вы хотите заняться разработкой для wireless smart card, Вам стоит задуматься над использованием инструментов **S40 Nokia 6212 NFC SDK.**

Этот SDK не только поддерживает JSR 257 API, но и позволяет симулировать наличие либо отсутствие виртуальной смарт-карты. SDK также поддерживает OMNIKEY и PEGODA карт-ридеры, которые подключены к Вашему настольному компьютеру, что позволяет быстро создавать и тестировать приложения на реальных NFC картах. Скриншот S40 Nokia 6212 NFC SDK показан на рис. 1.11.

Рис. 1.11. Вид среды разработки Nokia

BlackBerry JDE 4.7 – это полноценная среда для разработки и тестирования мобильных приложений для BlackBerry (рис. 1.12). Чтобы помочь разработчикам с их проектами, BlackBerry JDE 4.7 содержит справочную систему, включающую более 50 примеров проектов, которые используют Java ME JSR API и дополнительные BlackBerry API**.** JDE 4.7 содержит эмуляторы BlackBerry 9500/9530 с сенсорным экраном.

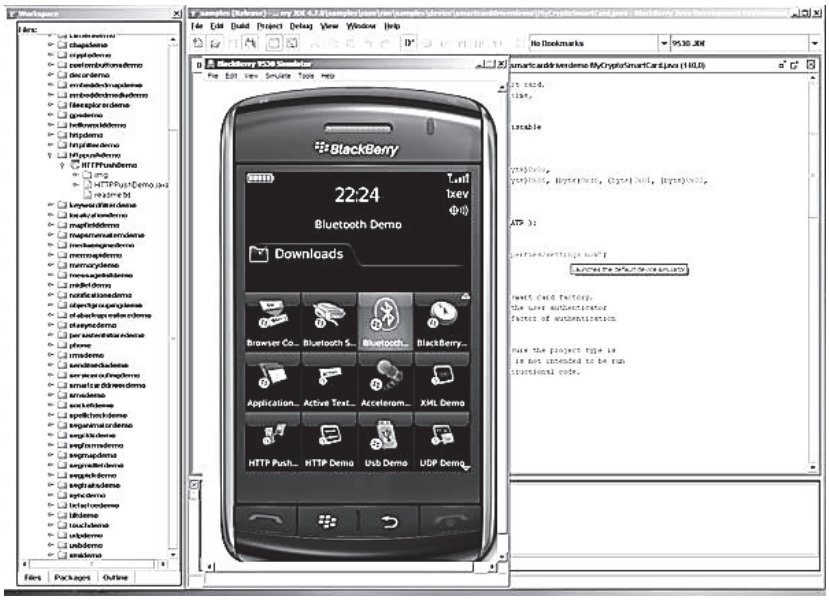


Рис. 1.12. Вид среды разработки BlackBerry JDE 4.7

Кроме того, эмулятор может реагировать на следующие события:

- наличие USB-соединения;
- наличие гарнитуры;
- эмуляция сенсорного скрина;
- изменение ориентации (тряска устройства);
- уровень батареи;
- установка или извлечение SD-карты;
- входящий звонок;
- изменение GPS-положения;
- использование камеры.

Sony Ericsson SDK 2.5 for Java ME.

Если Вы хотите сосредоточить свое внимание над экспериментами с JSR-расширениями, то можете поиграться с Sony Ericsson SDK 2.5 for Java ME. Особенно Вам следует обратить внимание на этот SDK, если Вы хотите использовать JSR 177 Security или Trust Services API (SATSA):

- SATSA APDU: базовые соединения с Java Card апплетами на SIM-карте;
- SATSA Crypto: для шифрования;
- SATSA PKI: цифровая подпись;
- SATSA JCRMI: для RMI-соединения с Java Card апплетами на SIM-карте.

Sony Ericsson SDK 2.5 for Java ME поддерживает 3D графику и анимацию: JSR 184 (Mobile 3D Graphics), JSR 239 (Java Binding for OpenGL ES) и Mascot Capsule API. Sony Ericsson SDK 2.5 for Java ME один из нескольких SDK, которые поддерживают JSR 229 Java Payment API. На рис. 1.13 показана интеграция **Sony Ericsson SDK 2.5** в NetBeans 6.5 IDE.

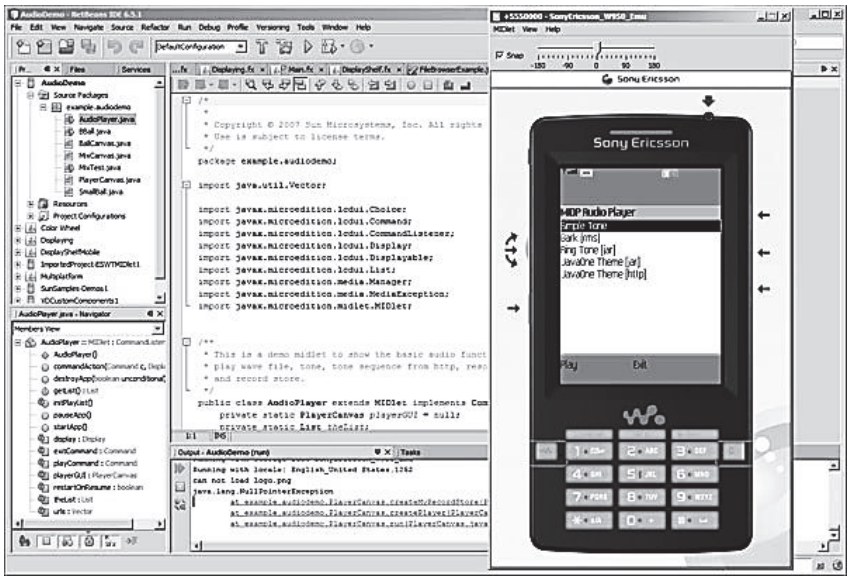


Рис. 1.13. Вид среды разработки Sony Ericsson SDK 2.5 for Java ME

LG SDK 1.2 for Java ME не блещет особой функциональностью и не очень хорошо поддерживает JSR-расширения. Однако это единственный SDK с поддержкой JSR 300 и DRM API, которые обеспечивают работу с защищенным цифровым контентом (графикой, звуком, видео) (рис. 1.14). LG SDK 1.2 for Java ME не содержит IDE, однако он, как, впрочем, и все другие SDK, может использоваться с NetBeans IDE.

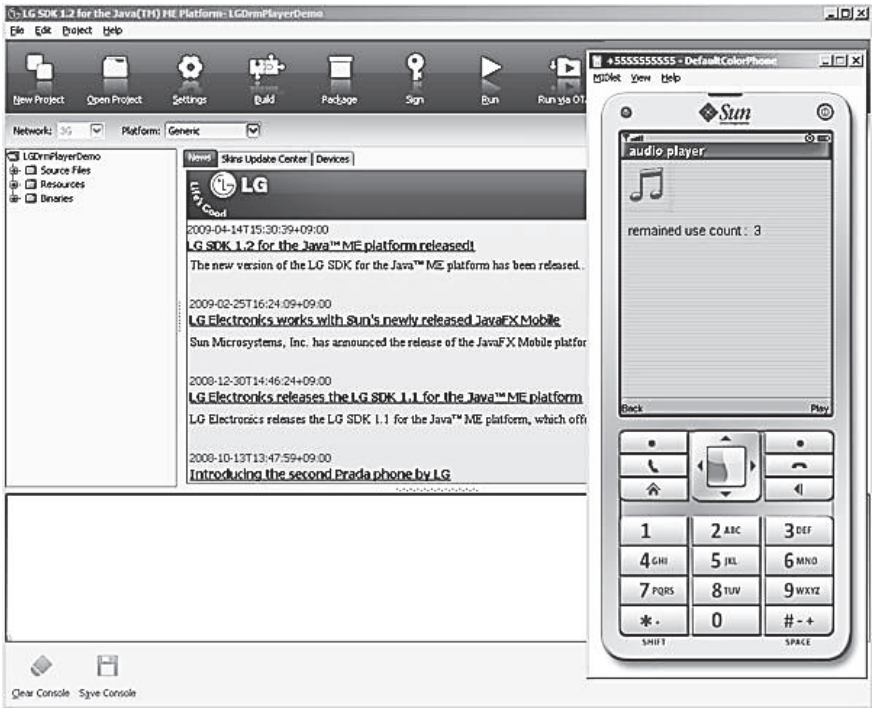


Рис. 1.14. Вид среды разработки LG SDK 1.2 for Java ME

LG SDK 1.2 может симулировать различные события:

- изменения в файловой системе;
- изменение местоположения;
- транзакция оплаты;
- изменение состояние подключенного устройства.

LG SDK 1.2 содержит также средства просмотра SVG-файлов.

Для более удобного программирования в указанных выше SDK удобно применять интегрированные среды разработчика (IDE), имеющие инструменты визуального программирования форм и встроенные отладчики. Наиболее применяемой IDE для создания «мидлетов» является среда NetBeans.

NetBeans 6.5 IDE.

Если вы хотите поработать над визуальным аспектом своего приложения, то вам следует воспользоваться NetBeans IDE. Эта среда наиболее подходит для разработки, проектирования и тестирования JavaFX-приложений (рис. 1.15). Основной принцип JavaFX – дать разработчикам возможность разрабатывать десктопные, веб-ориентированные и мобильные приложения используя один API framework.

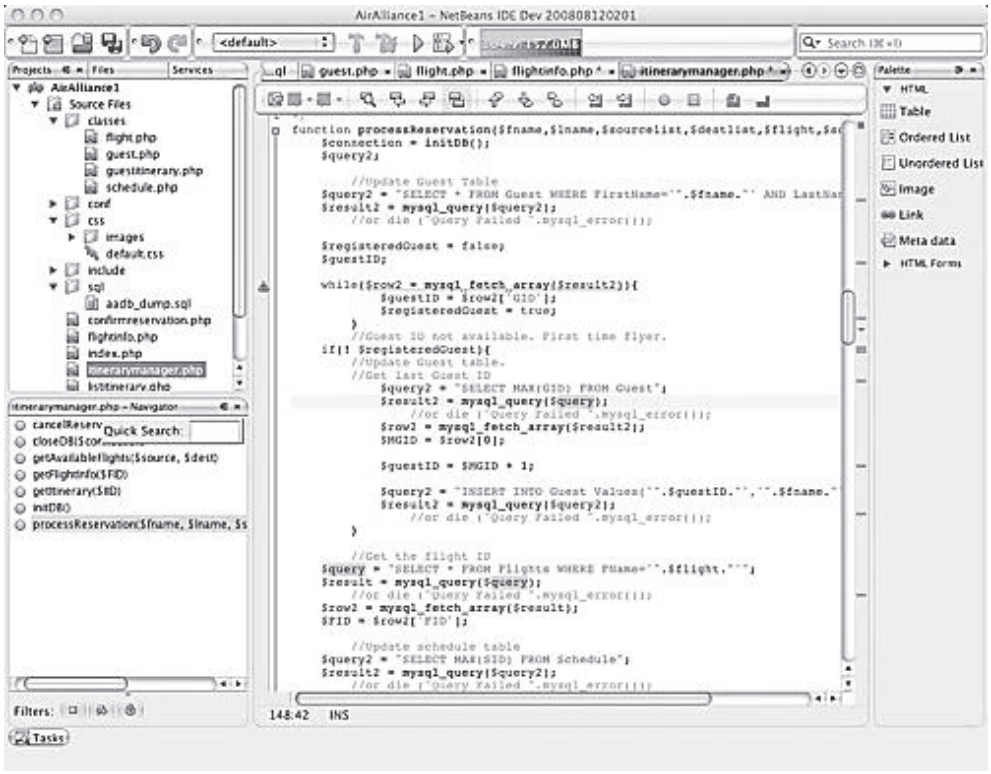


Рис. 1.15. Вид среды разработки NetBeans 6.5 IDE

В состав NetBeans 6.5 IDE входит Mobility Visual Designer – WYSIWYG утилита, позволяющая в визуальном режиме проектировать интерфейс Вашего приложения. В состав NetBeans 6.5 IDE

(org.netbeans.microedition) входят следующие визуальные компоненты: Alert, File Browser, Form, List, Login Screen, PIM Browser, SMS Composer, Splash Screen, Text Box, Wait Screen.

Mobility Visual Designer поддерживает векторную SVG-графику и анимацию.

Язык Java показался сложным? Тогда стоит использовать программы-посредники: «скармливайте» им программу, написанную на родственниках таких популярных языков, как Pascal и Basic, и на выходе получайте готовый Java-мидлет. Ярким примером такого «посредника» является IDE MIDletPascal.

MIDletPascal – это инструмент (IDE) для написания программ на языке Pascal для мобильных телефонов (рис. 1.16). Код транслируется в привычные для владельцев мобильников JAD и JAR-файлы. Поставляется MidletPascal с собственной, дружественной к пользователю средой разработки (IDE). Среда имеет встроенный компилятор, инспектор кода Java и обеспечивает построение архива JAR, что избавляет от установки Java SDK. В итоге компиляция и компоновка мидлетов проста, как нажатие на кнопку. Порадует Вас встроенная справка по доступным функциям: работа с графикой, SMS, звуками, файлами и т.д.

Таким образом, для компиляции не требуется ничего, кроме самого MidletPascal, что очень удобно при начальном обучении программированию для сотовых телефонов.

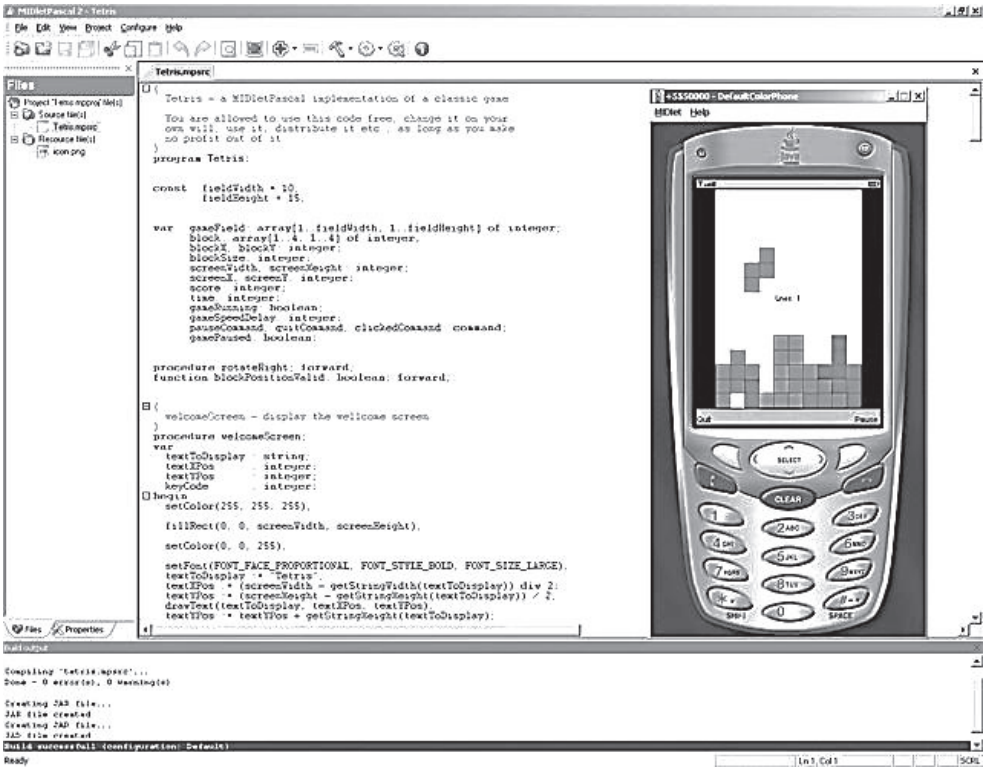


Рис. 1.16. Вид среды разработки MIDletPascal

OmegaBasic – специализированная среда разработки для создания программ и игр. Включает удобный редактор, поддерживает работу с проектами. Можно просматривать несколько файлов, ставить закладки, распечатывать справку по функциям. OmegaBasic позволяет работать с графикой, картами, звуком, музыкой, анимацией, видео и сетью. В качестве основного языка программирования используется Basic. Можно программировать и на Java, используя расширенный набор API OmegaBasic. Сайт разработчика – omegabasic.thegamecreators.com, оттуда можно скачать триальную версию, в которой программы ограничиваются 250 строками кода. Для функционирования OmegaBasic необходимо установить на ПК Java 1.4 SDK и Java Mobile 1.2 SDK.

MobileBasic – такая же специализированная среда. Ее особенность – наличие сервисов по так называемой немедленной OVER-THE-AIR («по воздуху») установке написанных мидлетов на телефоны. Написав программу, программист в MobileBasic может сохранить ее в виде JAD и JAR-файлов. Эти файлы с помощью MidletUploader выгружаются на сервер MobileBasic. Сервер создает WAP/WML-страницы, подключившись к которым с помощью WAP-браузера телефона можно установить мидлет. Кроме этого сервиса, в MobileBasic имеется графический редактор, редактор карт и плиточных изображений, а также редактор мелодий для телефонов Nokia. Скачать демо-версию MobileBasic можно со страницы www.mobilebasic.com/desktopedition.html. Стоимость продукта 24,99 фунтов стерлингов, ограничение триальной версии – максимум 1 Кб исходного кода. На сайте не стоит пренебрегать регистрацией, иначе запустить MobileBasic получится не более 30 раз. Как и в прошлом случае, необходимы установленные на компьютер Java SDK.

Итог: OmegaBasic и MobileBasic почти близнецы в плане подхода к написанию кода, к тому же обладают схожим функционалом. Недостатки: необходимость приобретать платную версию и устанавливать Java SDK. На этом фоне ярко выделяется MidletPascal – самый популярный, простой в установке, и главное – бесплатный! Именно поэтому среда MidletPascal в данном пособии будет рассмотрена более подробно.

1.9.2. Инструменты для создания программного обеспечения в операционных системах мобильных устройств

Инстументарий для программирования в Symbian.

Программная платформа Symbian Series 60 (или S60) (рис. 1.17) – самая популярная в мире смартфонов и коммуникаторов, если судить по продажам мобильных устройств. Поэтому приложения именно для этой платформы весьма актуальны.

C++ for Symbian – наилучший (и, по сути, единственный) язык для создания профессиональных и коммерческих приложений для смартфонов Symbian Series 60. Именно на нем пишется сама система и предустановленное ПО. Если вы полны решимости программировать на C++ for Symbian, то необходимо установить:

1. среду разработки – CodeWarrior (www.forum.nokia.com/codewarrior);
2. Carbide.c++ (www.forum.nokia.com/main/resources/tools_and_sdks/carbide_cpp/) или другое;
3. SDK для Symbian соответствующей Edition и Feature Pack под нужную среду разработки (www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html);
4. Java 2 Standard Edition;
5. Perl версии не ниже 5.003.07.

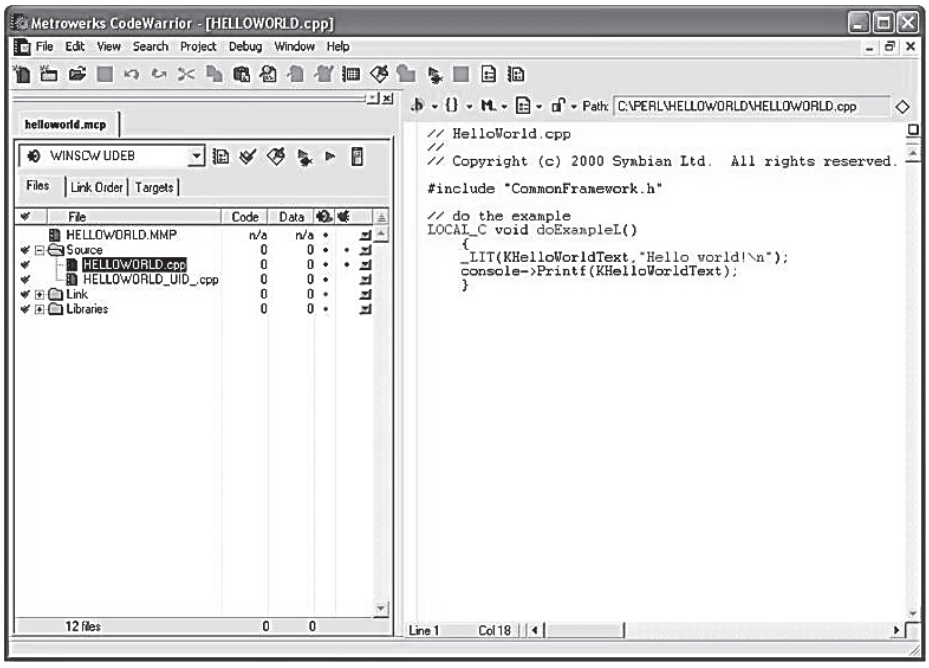


Рис. 1.17. Инстументарий для программирования в Symbian

При создании программ на C++ for Symbian можно получить доступ ко всем возможностям смартфона, что не идет ни в какое сравнение с Java и прочими интерпретируемыми языками. Полученные продукты будут потреблять минимум ресурсов и работать с максимальной скоростью, т.к. между программой и системой не будет посредников-интерпретаторов.

При компиляции программного кода создается приложение, а также иконка и необходимые файлы ресурсов. Все это в конце концов упаковывается в установочный файл SIS. После его подписи (если необходимо) разработчик может распространять и продавать программу как готовый продукт.

Инстументарий для программирования в Windows Mobile.

Для создания программ под Windows Mobile Microsoft предлагает среду разработки Visual Studio. Если у Вас уже есть установленная Visual Studio 2010 (Professional или Ultimate), то Вы можете использовать для разработки свою редакцию Visual Studio 2010 после установки Windows Phone Developer Tools.

Также существует Expression Blend for Windows Phone – программа для разработки дизайна, которая позволяет создавать и добавлять специальные визуальные возможности, такие как градиенты, анимации и переходы. Для некоторых задач Expression Blend проще в использовании, чем Visual Studio. Некоторые задачи, которые легко выполняются с помощью Expression Blend:

- визуальное создание шаблонов данных;
- использование во время разработки тестовых данных для визуализации шаблонов данных;
- визуальное создание стилей элементов управления;
- создание и просмотр анимации.

На рис. 1.18 показан внешний вид Expression Blend.

Рис. 1.18. Среда программирования Expression Blend

Имеется бесплатный пакет: Visual Studio 2010 Express for Windows Phone, который включает в себя drag-and-drop дизайнер, эмулятор телефона, редактор кода и отладчик. Если Вы работали с Visual Studio для разработки других видов приложений, то найдете среду для разработки мобильных приложений очень знакомой. На рис. 1.19 показан внешний вид Visual Studio 2010 Express for Windows Phone.

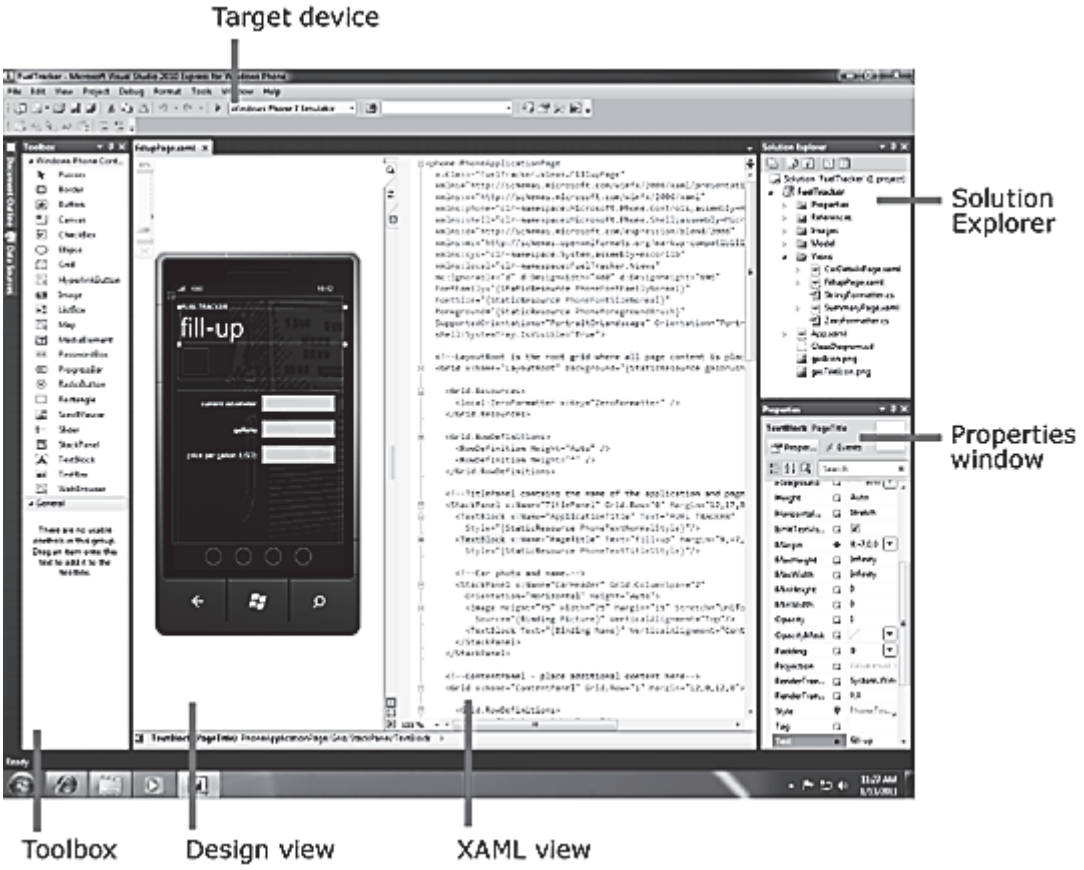


Рис. 1.19. Среда программирования Visual Studio Phone

Дизайнер для Windows Phone содержит панель инструметов (Toolbox), режим дизайна (Design view), режим XAML (XAML view), обозреватель решений (Solution Explorer) и окно Свойства (Properties window), похожие на стандартный дизайнер Visual Studio. Два ключевых различия:

1. в режиме дизайна поверхность выглядит как Windows Phone устройство;

2. появилось целевое устройство (Target device), которое позволит Вам выбрать, будет ли Вы отлаживать приложение на устройстве или эмуляторе.

Инстументарий для программирования в Android.

Самый простой способ приступить к разработке приложений для Android – это загрузить SDK Android и Eclipse IDE. Разработку Android-приложений можно вести на платформах Microsoft Windows, Mac OS X или Linux. Чаще всего используются Eclipse IDE и плагин Android Developer Tools для Eclipse.

Android-приложения пишутся на языке Java, но компилируются и выполняются в Dalvik VM (не в виртуальной машине Java). Кодирование на языке Java в рамках Eclipse – интуитивно понятный процесс. Eclipse предоставляет богатую среду Java, включая контекстно-зависимую справ-

ку и подсказки к коду. Когда ваш Java-код будет безошибочно скомпилирован, Android Developer Tools сам позаботится о том, чтобы приложение было надлежащим образом упаковано, в том числе снабдит его файлом AndroidManifest.xml.

Android-приложение можно написать и без Eclipse и плагина Android Developer Tools, но для этого нужно хорошо разбираться в Android SDK.

Android SDK распространяется в виде файла ZIP, который распаковывается в папку на жестком диске. Так как вышло несколько обновлений SDK, рекомендуется поддерживать среду разработки в порядке, чтобы можно было легко переключаться между разными установками SDK.

Android-приложения могут тестироваться как на реальном устройстве, так и на эмуляторе Android, который прилагается к SDK Android. На рис. 1.20 показан главный экран эмулятора Android.

Отладочный мост Android – утилита **adb** – поддерживает несколько дополнительных аргументов командной строки, которые обеспечивают мощные функции, такие как копирование файлов в устройство и из него. Аргумент оболочки командной строки позволяет подключаться к самому телефону и подавать простые команды оболочки. Рис. 1.20 иллюстрирует команду оболочки **adb**, подаваемую реальному устройству, подключенному к ноутбуку под Windows с помощью кабеля USB.

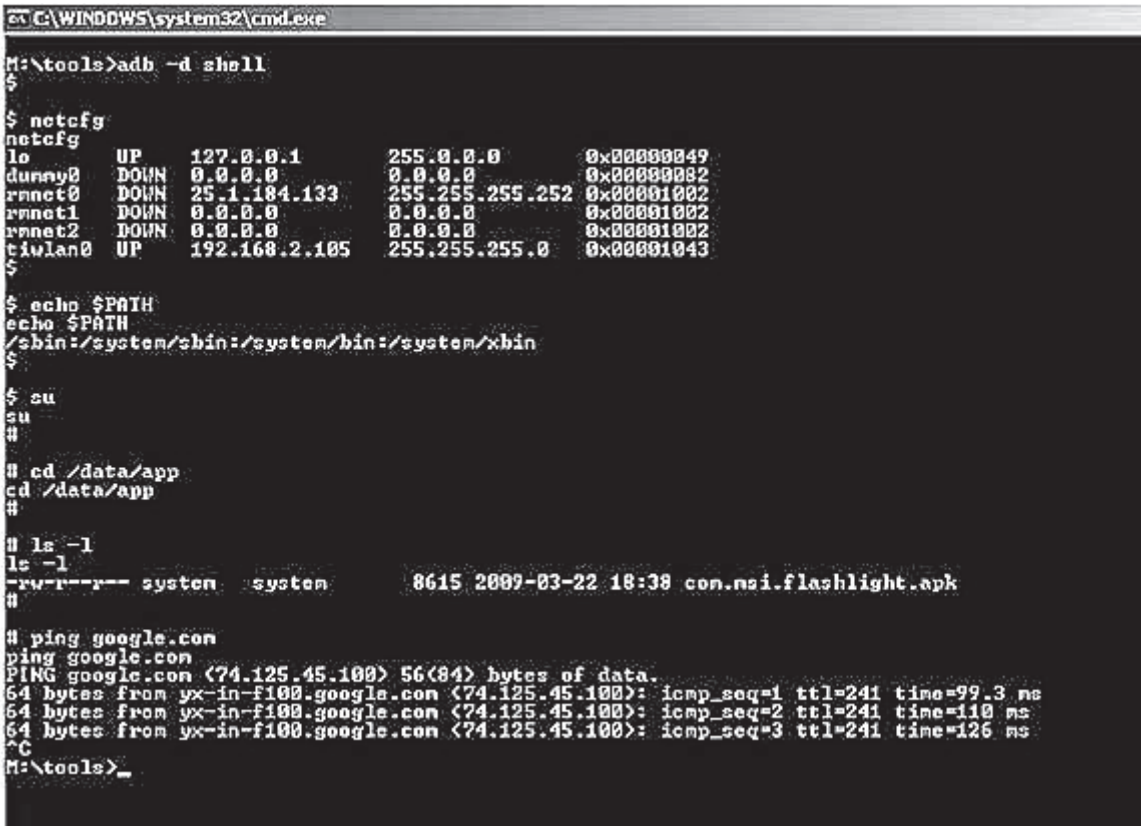


Рис. 1.20. Главный экран эмулятора Android

Инстументарий для программирования в BlackBerry.

Разработчики могут использовать как стандартные инструменты на основе веб-технологий, например HTML/HTML5, CSS, JavaScript или Java®, так и специальные средства разработки приложений для BlackBerry. Разработка приложений для BlackBerry может вестись также при помощи таких распространенных инструментов, как Eclipse и Microsoft Visual Studio. Таким образом, разработчики обладают свободой выбора наиболее подходящего им инструментария для создания приложений.

Инстументарий для программирования в Bada.

Предложенный разработчикам инструментарий позволяет писать код не для конкретной ОС, а работать с определенными функциями (например, оболочкой, контактами), а операционных систем, с которыми работает эта надстройка, может быть несколько. В рамках бета-версии SDK доступ осуществлялся только к функциям TouchWiz 3D и ряду системных функций, что позволяет назвать систему Bada обычной надстройкой к интерфейсу. В будущем развитие средств разработки программ позволит писать полнофункциональные программы, задействующие не только интерфейс, но и другие возможности телефона (не затрагивая ОС, лежащую в основе).

Вспомним, что компания Samsung стала первым производителем, кто транслировал интерфейс TouchWiz с собственных устройств на другие ОС, т.е. сделали этот интерфейс кросс-платформенным, чтобы приучить потребителей к нему. Не важно, какая ОС, важно, что везде потребитель видит один и тот же интерфейс и ассоциирует его с компанией Samsung. Bada позволяет быстро реагировать на расстановку сил на рынке операционных систем, не привязываясь ни к одной из них, что является достаточно гибкой стратегией.

Компания Samsung активно взялась за работу с разработчиками ПО, так что уже к моменту выхода операционной системы Bada на рынок для нее было доступно большое количество разнообразных программ, игр и виджетов, сконцентрированных на специализированном ресурсе Samsung Apps.

Инстументарий для программирования iPhone.

До официальной публикации SDK у разработчиков не было возможности легальной разработки native-приложений для iPhone и iPod Touch. Учитывая огромный интерес к iPhone, Apple пошла на компромисс: позволила сторонним разработчикам создавать так называемые виджеты – приложения, выполняемые в веб-браузере Safari, интегрированном в iPhone и iPod Touch. Основным отличием виджетов от native-приложений является необходимость написания кода не на Objective C, а с использованием стандартных веб-технологий вроде HTML, CSS, JavaScript и AJAX. С точки зрения пользователя, такое приложение отличается тем, что выполняется в веб-браузере и открывается не путем выбора иконки из главного меню

устройства, а при выборе закладки. Для ознакомления с процессом создания и развертывания виджетов для iPhone рекомендуется почитать книгу «Professional iPhone and iPod Touch Programming», а также заглянуть на сайт: http://developer.apple.com/webapps/.

Отсутствие официальной возможности создавать ПО не остановило энтузиастов. Они подготовили средства разработки, позволяющие создавать полноценный софт для JailBreaked iPhone. В процессе JailBreaking на аппарат устанавливается ПО с названием Installer. С его помощью пользователи могут скачивать и устанавливать необходимый софт из каталога, который формируется из репозиториев (их адреса прописываются вручную в Installer). Так что JailBreaking не только разлочка, но и процедура, позволяющая получить полный доступ к файловой системе iPhone. Описание процесса без труда можно найти в Интернете, поэтому мы не будем на этом останавливаться. Софт, распространяющийся через Installer, написан с использованием «неофициального» процесса разработки. До недавнего времени иного пути создания и даже установки стороннего ПО в iPhone не было.

Но в марте 2010 г. Apple осчастливили-таки общественность публикацией первой беты SDK. С тех пор на офсайте разработчиков Apple периодически публикуются новые версии беты SDK и документации. SDK представляет собой IDE XCode, набор необходимых библиотек, эмулятор и другие инструменты.

Чтобы программировать под iPhone, нужен Mac с установленной Mac OS X Leopard. Грустно, но это так. Вообще говоря, можно развернуть среду разработки на Unix и даже пытаться писать из-под VMWare, но это связано с рядом сложностей, которые бурно обсуждаются в Интернете. Кроме того, необходимо установить и сконфигурировать SDK. Описание процесса настройки рабочей станции для «неофициальной» разработки можно прочитать в замечательной книжке «iPhone Open Application Development», которую легко найти в сети Интернет.

При разработке приложений для iPhone OS, а также MacOS 10.5 и выше, используется язык программирования Objective C 2.0. Он является своеобразной надстройкой над ANSI C, предназначенной для гибкого объектно-ориентированного программирования. Не совсем понятно, чем Apple не угодил C++. Многие концепции Objective C заимствованы у одного из первых объектно-ориентированных языков Smalltalk. Тем не менее, программа для iPhone может содержать как код на Objective C, так и на C или C++.

При компиляции используются инструменты GNU Compilers Collection, которые распознают принадлежность кода к конкретному подвиду GNU C/C++ по расширению файла. В частности, C – код содержится в файлах с расширением *.c; C++ – код в *.mm; Objective C – в *.m.

Как видно из вышесказанного, одно перечисление (и то не всех) технологий и инструментов разработки программного обеспечения для мобильных устройств занимает достаточно много места, а для подробного освещения этих технологий требуется не один том учебной литературы. В нашем курсе мы познакомимся с наиболее общей технологией создания приложений мобильного мира – технологией создания мидлетов. С остальными технологиями (по необходимости) программист, имеющий навык создания мидлетов, сможет освоиться сам.

Успехов Вам в изучении курса!

2. РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ WINDOWS PHONE

Windows Phone 7 обеспечивает поддержку двух популярных (в настоящее время) платформ разработки: Silverlight и XNA. Это гарантирует, что в Windows Phone 7 найдется много интересного и для разработчиков.

Silverlight – браузерное развитие Windows Presentation Foundation (WPF) – уже обеспечила Веб-разработчиков беспрецедентными возможностями разработки сложных пользовательских интерфейсов, предоставляя традиционные элементы управления, высококачественный текст, векторную графику, мультимедиа, анимацию и привязку данных, которые могут выполняться во множестве браузеров и на разных платформах. Windows Phone 7 расширяет использование Silverlight на мобильные устройства.

XNA (три буквы, обозначающие XNA, не аббревиатура) – это игровая платформа Майкрософт, поддерживающая основанную на спрайтах 2D и 3D графику с традиционной архитектурой игрового цикла. Несмотря на то что главным предназначением XNA является написание игр для консоли Xbox 360, разработчики могут создавать на XNA программы и для ПК, и для стильного аудиоплеера Майкрософт Zune HD.

Для разработки приложений под Windows Phone желательно иметь Visual Studio 2010 с Service Pack 1 редакции Professional или выше и пакет разработчика Windows Phone SDK 7.1 (это инструментарий на начало 2012 г., в дальнейшем номера версий будут меняться). Взять Service Pack 1 для VS-2010 (бесплатно) можно по адресу: http://go.microsoft.com/ fwlink/?LinkId=210710, а Windows Phone SDK 7.1 (тоже бесплатно) – по адресу: http://go.microsoft.com/fwlink/ ?LinkID=226694.

Если у Вас нет Visual Studio 2010 Professional, то пакет Windows Phone SDK 7.1 Вы все-равно можете установить себе на ПК. В момент установки Вам автоматически будет установлена бесплатная версия Visual Studio 2010 Express for Windows Phone, на которой также можно разрабатывать приложения под Windows Phone (при этом, конечно, Ваш ПК должен быть связан с Интернетом).

Обе версии интегрированных средств разработки Visual Studio предоставляют разработчику полноценные возможности по отладке на устройстве и эмуляторе, такие же, какие есть у разработчиков приложений под настольную версию Windows.

Обратите внимание: для того чтобы отлаживаться на реальном устройстве помимо собственно устройства и кабеля для подключения его к компьютеру разработчика на компьютере со средствами разработки необходимо иметь установленное ПО Zune (http://zune.net). Также перед развертыванием приложения и отладкой требуется зарегистрировать устройство или «разлочить» с использованием утилиты Windows Phone Developer Registration Tool, которая устанавливается вместе с Windows Phone SDK.

2.1. Windows Phone SDK

Этот пакет содержит всё необходимое для того, чтобы начать разработку. Версия Windows Phone SDK 7.1 Release Candidate доступна в лицензии «Go Live» с возможностью разрабатывать свои приложения и публиковать их в Windows Phone Marketplace.

Windows Phone SDK 7.1 Release Candidate содержит следующие компоненты:

- Windows Phone SDK 7.1;
- Windows Phone Emulator;
- Windows Phone SDK 7.1 Assemblies;
- Silverlight 4 SDK and DRT;
- Windows Phone SDK 7.1 Extensions for XNA Game Studio 4.0;
- Expression Blend SDK for Windows Phone 7;
- Expression Blend SDK for Windows Phone OS 7.1;
- WCF Data Services Client for Windows Phone;
- Microsoft Advertising SDK for Windows Phone.

Если у Вас не установлена версия Visual Studio 2010 редакции Professional, Expression Bland 4 или XNA Game Studio 4.0, то в процессе установки также будут скачаны и установлены:

- Visual Studio 2010 Express for Windows Phone;
- Expression Blend 4 for Windows Phone;
- XNA Game Studio 4.0.

2.2. Expression Blend и Expression Blend for Windows Phone

Expression Blend – это интерактивный визуальный дизайнер для XAML-технологии описания интерфейса для приложений Silverlight и WPF. Это отличное средство разработки, которое позволяет просто манипулировать слоями, анимацией, стилями и шаблонами. Это базовое сред-

ство разработки на XAML. Собственно, программа Expression Blend не бесплатна, однако специальная версия для создания дизайнов приложений под Windows Phone под названием Expression Blend 4 for Windows Phone доступна для разработчиков бесплатно. Она закачается и установится в процессе установки Windows Phone SDK, если у вас на компьютере нет полной версии Expression Blend. Подробнее об Expression Blend 4 можно прочитать на MSDN: http://msdn.microsoft.com/ru-ru/library/cc296227.aspx.

2.3. XNA Game Studio 4.0

XNA Game Studio – это программное окружение, которое позволяет разрабатывать в Visual Studio игры для Windows Phone, консоли Xbox 360 и компьютеров на базе Windows. Включает в себя XNA Framework, представляющий собой набор библиотек на управляемом коде для разработки игр. Подробнее можно прочитать на MSDN: http://msdn.microsoft.com/ru-ru/library/bb200104.aspx.

2.4. Windows Phone Emulator

Несмотря на то что Windows Phone Emulator не содержит полного набора приложений, доступных на реальном устройстве, он предоставляет мощную среду, позволяющую практически полностью разработать приложение в эмуляторе.

Эмулятор Windows Phone Emulator не поддерживает проигрывание медиаконтента Zune. Эмулятор поставляется только с одним встроенным приложением Internet Explorer, но это Internet Explorer 9 с поддержкой HTML5. При этом эмулятор позволяет тестировать звонки и отсылку SMS-сообщений, поддерживает multitouch на мониторах с поддержкой multitouch, поддерживает симуляцию камеры, геолокационных сервисов и акселерометра, а также позволят делать снимки экрана.

Подробнее можно прочитать на MSDN: http://msdn.microsoft.com/ru-ru/library/ff402563(v=VS.92).aspx.

2.5. Windows Phone Developer Registration Tool

Перед тем как разработчик сможет развернуть своё приложение на реальном устройстве, его необходимо зарегистрировать как устройство разработчика – «разлочить». Это делается один раз для определенного телефона. Зарегистрированный на Marketplace разработчик может зарегистрировать до трех устройств (для разработчика, зарегистрированного как студент, количество устройств ограничено до одного). Подробнее: http://create.msdn.com.

2.6. Windows Phone Profiler

Windows Phone Profiler доступен в меню Debug Visual Studio с установленным инструментарием Windows Phone SDK (рис. 2.1).

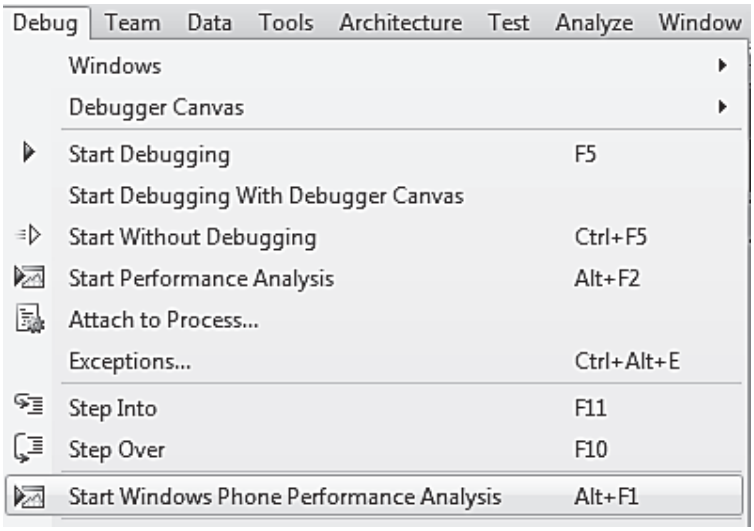


Рис. 2.1. Меню запуска Windows Phone Analysis

Windows Phone Profiler анализирует работу программы во время исполнения, идентифицирует возможные проблемы с производительностью. Подробнее можно прочитать на MSDN: http://msdn.microsoft.com/ru-ru/library/hh202934(v=VS.92).aspx.

2.7. Silverlight Toolkit for Windows Phone

Silverlight Toolkit for Windows Phone – набор полезных элементов управления Silverlight для Windows Phone с поддержкой режима дизайна от команды разработчиков Silverlight. Доступен весь исходный код, примеры и документация. Обновляется приблизительно раз в три месяца, доступен по адресу http://silverlight.codeplex.com или через NuGet.

Текущий релиз включает в себя такие элементы управления, как ContextMenu, DatePicker и TimePicker, ToggleSwitch, WrapPanel и GestureHelper.

2.8. Среда разработки

После установки средств разработки Windows Phone SDK в диалоге New Project в Visual Studio появятся группы проектов для Silverlight for Windows Phone (рис. 2.2).

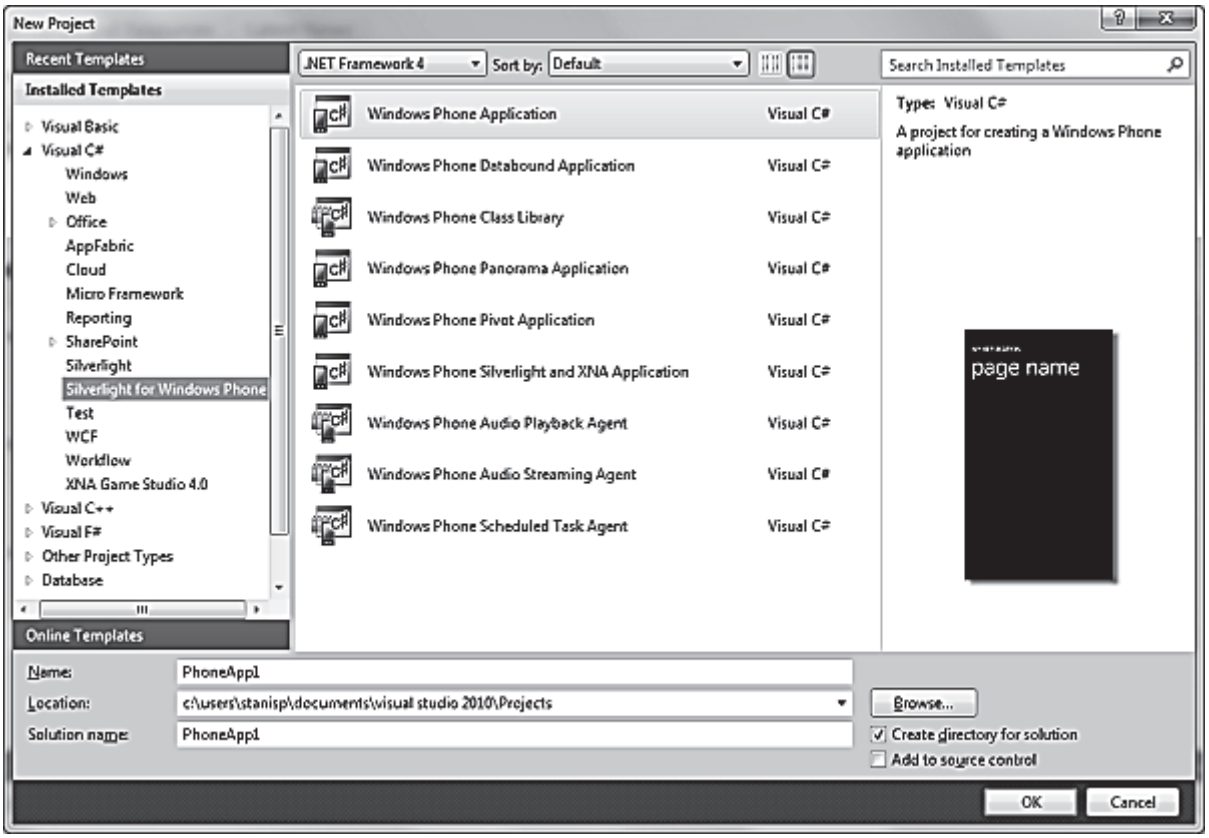


Рис. 2.2. Группы проектов в меню Silverlight for Windows Phone

Материал данного методического пособия сфокусирован на разработке под Windows Phone на Silverlight, поэтому рассмотрим доступные разработчику приложений шаблоны несколько более подробно.

После установки разработчику доступны следующие шаблоны приложений Silverlight for Windows Phone (рис. 2.3):

- 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.

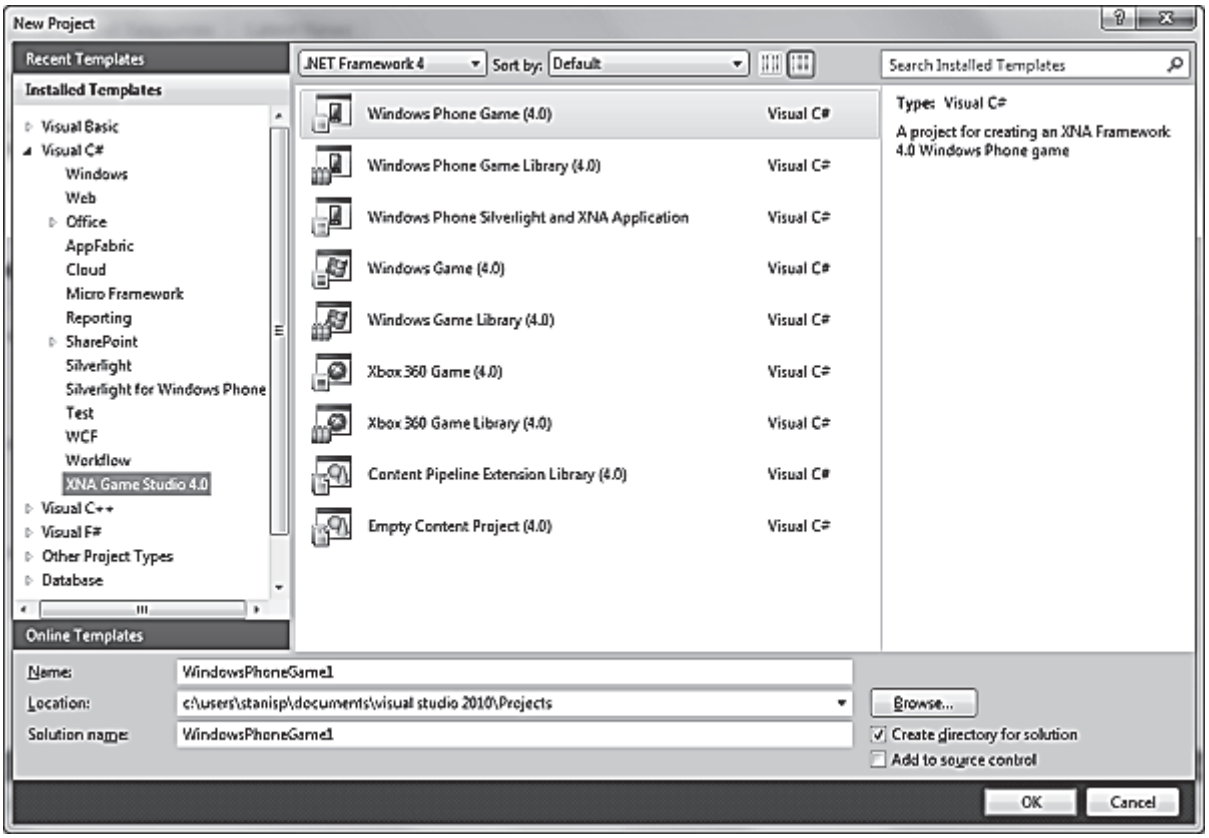


Рис. 2.3. Шаблоны приложений Silverlight for Windows Phone

Перед тем как перейти непосредственно к шаблонам приложений, надо сказать несколько слов по поводу Windows Phone и Metro-дизайна.

2.9. Windows Phone и Metro-дизайн

Платформа Windows Phone не просто очередная платформа для мобильных устройств. Она содержит в себе не только технологическую составляющую, но и полностью проработанную концепцию дизайна интерфейса и взаимодействия с пользователем под названием Metro-дизайн или стиль Metro.

Если Вы дизайнер или в Вашей команде есть выделенный дизайнер, Вы можете воспользоваться всей мощью инструментария Expression Blend 4 или Expression Blend for Windows Phone, которая поставляется вместе с Windows Phone SDK.

Что же делать, если Вы разработчик и не хотите заниматься визуальным дизайном приложения, например, Вы разрабатываете бизнес-приложение, и всё, что от него требуется, – соответствовать общему дизайну и стилю Windows Phone?

Всё очень просто. Во-первых, Silverlight для телефона разработан с учётом требований Metro-дизайна, поэтому все встроенные элементы

управления выполнены в Metro-дизайне. Во-вторых, по умолчанию приложения, созданные из шаблонов из поставки Windows Phone SDK, работают, выглядят и используют стили и шрифты в соответствии с Metro-дизайном.

С другой стороны, возможностей стилизации элементов управления и приложений, основанных на XAML, которые представляет Silverlight, вполне достаточно, чтобы сделать своё приложение неповторимым и узнаваемым, оставаясь в рамках стиля Metro.

Руководство по дизайну интерфейсов и взаимодействию с пользователем для Windows Phone можно найти по следующей ссылке: http://msdn.microsoft.com/ru-ru/library/hh202915.aspx.

Всё, что было сказано выше, относится, конечно, к дизайну обычных приложений, т.к. требования к дизайну игровых приложений и их интерфейсу могут существенно отличаться. При этом не надо забывать об общих принципах взаимодействия с пользователем, заложенных в концепции Windows Phone.

2.10. Шаблоны приложений

Сначала давайте рассмотрим три шаблона, представляющих собой три основных стиля приложения для Windows Phone (рис. 2.4):

- Windows Phone Application;
- Windows Phone Pivot Application;
- Windows Phone Panorama Application.

Рис. 2.4. Шаблоны приложений

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: People, Marketplace, Pictures, Music+Videos и др. Шаблон использует элемент управления Panorama.

Шаблоны, заканчивающиеся на Agent – это шаблоны библиотек, для выполнения соответствующих фоновых задач:

- Windows Phone Audio Playback Agent;
- Windows Phone Audio Streaming Agent;
- Windows Phone Scheduled Task Agent.

Шаблон Windows Phone Databound Application – простой шаблон приложения в виде списка – детальное представление с реализацией навигации между страницами с передачей параметров и хранением данных в глобальном VeiwModel.

Шаблон Windows Phone Class Library – шаблон библиотеки классов для Windows Phone.

Шаблон Windows Phone Silverlight and XNA Application – для Silverlight-приложения, которое может использовать XNA для рендеринга графического контента.

2.11. Создаем первый проект на Silverlight

Прежде чем приступить к приемам программирования для Windows Phone 7, необходимо познакомиться с базовыми понятиями. Те, кто имел опыт программирования на Windows Mobile 6, уже обладают некоторыми знаниями в этой области (использование эмуляторов, отличия от настольной .NET Framework и т.д.). Тем не менее и им также придется учиться заново, т.к. Microsoft в очередной раз поменяла правила и все прежние навыки теперь считаются устаревшими и выброшены на свалку истории. Гонка за новыми технологиями продолжается.

В этом разделе мы создадим традиционное приложение «**Здравствуй, мир!**», чтобы понять основные принципы создания программ для Windows Phone 7. Наше первое приложение будет построено при помощи технологии Silverlight, которая является удобной платформой для бизнес-приложений и игр.

2.11.1. Терминология

Прежде чем приступить к написанию приложений для Windows Phone, необходимо познакомиться с некоторой терминологией. Рассмотрим некоторые элементы Windows Phone (рис. 2.5).

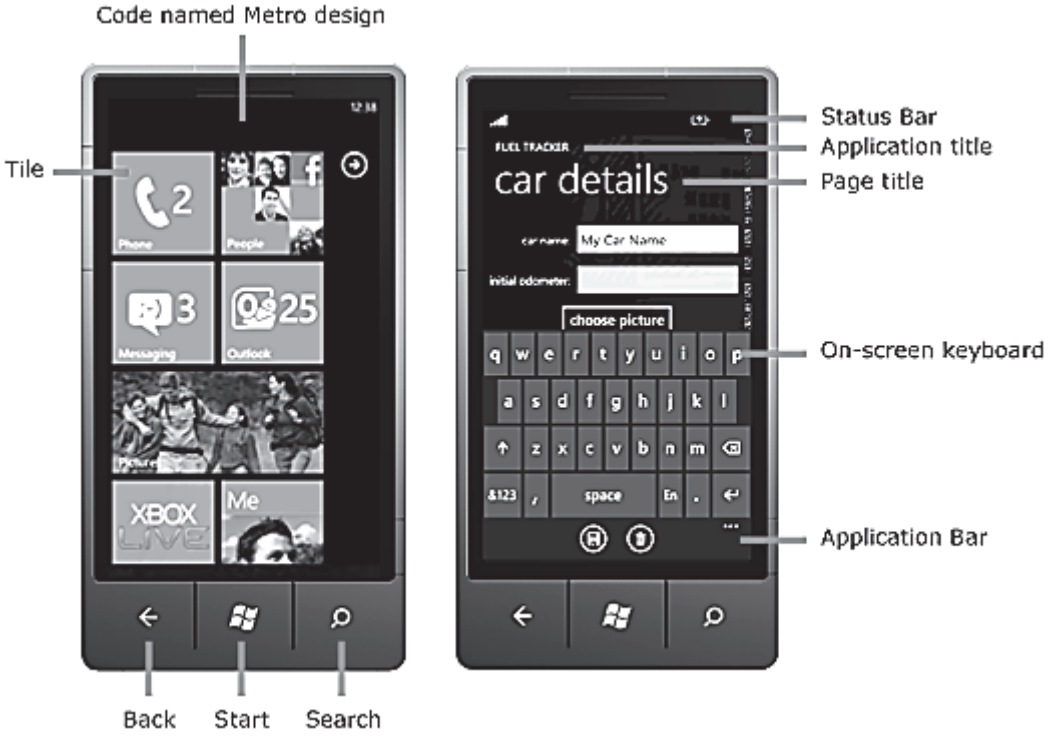


Рис. 2.5. Элементы интерфейса смартфона

Tile (плитка) – значок приложения на стартовом экране. Плитка может быть динамической и отображать некоторую информацию для пользователя.

Application Title – название приложения. Обычно в верхнем регистре.

Page Title – заголовок страницы. Обычно в нижем регистре.

Status Bar – состояние работы телефонной части, например, уровень сигнала.

On-screen keyboard – экранная клавиатура. Появляется при получении фокуса текстовым полем. Иногда используется термин SIP (soft input panel).

Application Bar – дополнительная всплывающая панель для навигации по приложению. Содержит кнопки и/или пункты меню.

Кнопки **Back, Start, Search** – стандартные кнопки на любом устройстве с Windows Phone.

Специально для Windows Phone 7 был разработан новый пользовательский дизайн под кодовым названием «Metro». Рекомендуется следовать этому дизайну в своём приложении, чтобы оно интегрировалось с ОС и другими приложениями. Дизайн обеспечивает простой в использовании интерфейс, предназначенный для уменьшения потребления энергии на телефоне.

2.11.2. Создание нового проекта

Запустите Visual Studio 2010 Express For Windows Phone. В меню **File** выберите пункт **New Project**. У вас откроется диалоговое окно **New Project** (рис. 2.6).

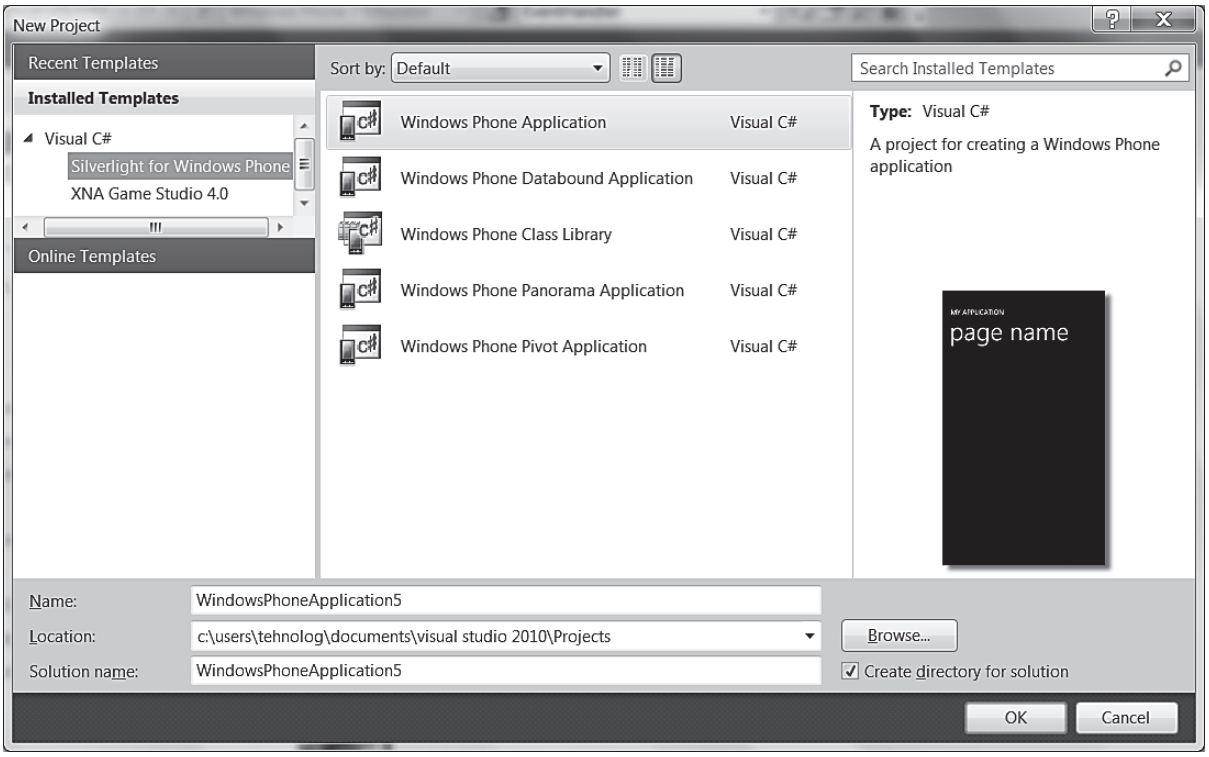


Рис. 2.6. Окно выбора типа проекта

Далее слева выберите пункт **Silverlight for Windows Phone**. Как видите, для данного типа проекта доступны несколько шаблонов: Windows Phone Application, Windows Phone Databound Application, Windows Phone Class Library, Windows Phone Panorama Application, Windows Phone Pivot Application. Для нашего учебного примера выберем первый вариант.

Постарайтесь сразу выработать привычку задавать понятные имена для своих проектов. Поэтому присваиваем проекту имя **WP7HelloWorld** и нажимаем на кнопку **OK**.

Спустя несколько секунд Visual Studio создаст новый проект. Вы увидите несколько окон на экране. Оставим пока в покое окна в центральной части экрана с изображением телефона и кодом XAML, а посмотрим на окно **Solution Explorer**. В Solution Explorer хорошо видна структура решения, созданного на основе выбранного шаблона Windows Phone Application. В нашем случае в этом окне содержится один проект **WP7HelloWorld**.

Проект содержит следующие файлы (табл. 2.1).

Таблица 2.1

Список файлов проекта

Имя файла	Содержимое файла
App.xaml/App.xaml.cs	Содержит точку входа программы, инициализирует ресурсы програмы и выводит программу на экран
MainPage.xaml/MainPage.xaml.cs	Содержит страницу с пользовательским интерфейсом
ApplicationIcon.png	Файл значка в формает PNG, который выводится в списке приложений телефона
Background.png	Файл изображения в формате PNG, который выводится на стартовой странице
SplashScreenImage.jpg	Файл изображения, которое выводится во время загрузки приложения. Вы можете заменить на свою картинку
Properties/AppManifest.xml	Манифест, необходимый для создания сборки
Properties/AssemblyInfo.cs	Содержит метаданные о имени и версии приложения, которые встраиваются в сборку
Properties/WMAppManifest.xml	Манифест, который содержит специальные данные о приложении для Windows Phone Silverlight
Папка References	Различные библиотеки (сборки), которые обеспечивают работоспособность приложения

Подробнее о файлах проекта можно почитать в статье [1].

2.11.3. Сборка и тестирование программы

Хотя наша программа еще бесполезна, тем не менее давайте проверим ее работу – скомпилируем и протестируем в эмуляторе.

В меню **View** выберите пункт **Output** (возможно потребуется настроить это меню), чтобы открыть окно **Output**. Далее в меню **Debug** выберите команду **Build Solution** (SHIFT + F6) для компиляции.

Посмотрите в окно **Output** (рис. 2.7) и изучите сообщения, генерируемые во время компиляции приложения, включая финальное сообщение, в котором подводится окончательный итог и количество предупреждений и ошибок.

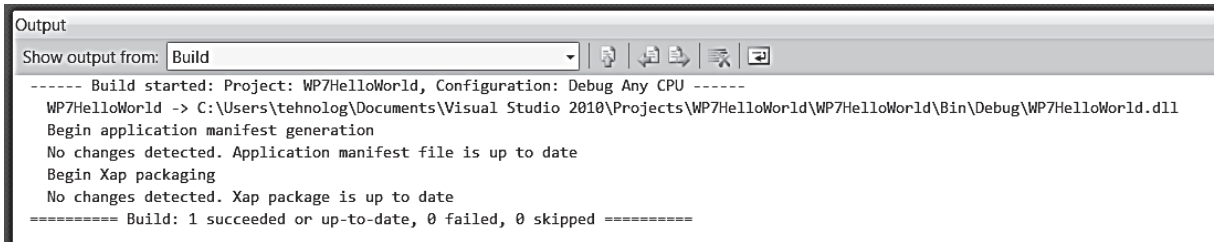


Рис. 2.7. Вид окна Output

Также Вы можете использовать окно **Error List** (View → Other Windows → Error List), которое показывает ошибки, предупреждения и сообщения, выдаваемые компилятором (рис. 2.8). Вы можете сделать двойной щелчок на описании ошибки, чтобы автоматически оказаться в нужном месте исходного кода.

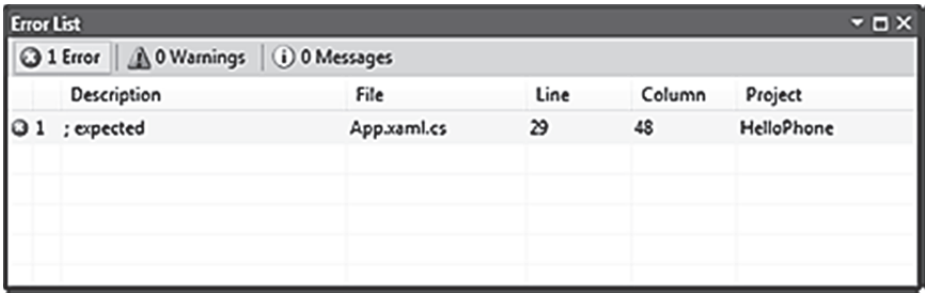


Рис. 2.8. Вид окна Error List

2.11.4. Запуск программы в эмуляторе

Убедитесь, что у вас установлен **Windows Phone Emulator** в выпадающем списке устройств **Select Device**, который расположен рядом с кнопкой **Start Debugging** на панели инструментов (рис. 2.9).

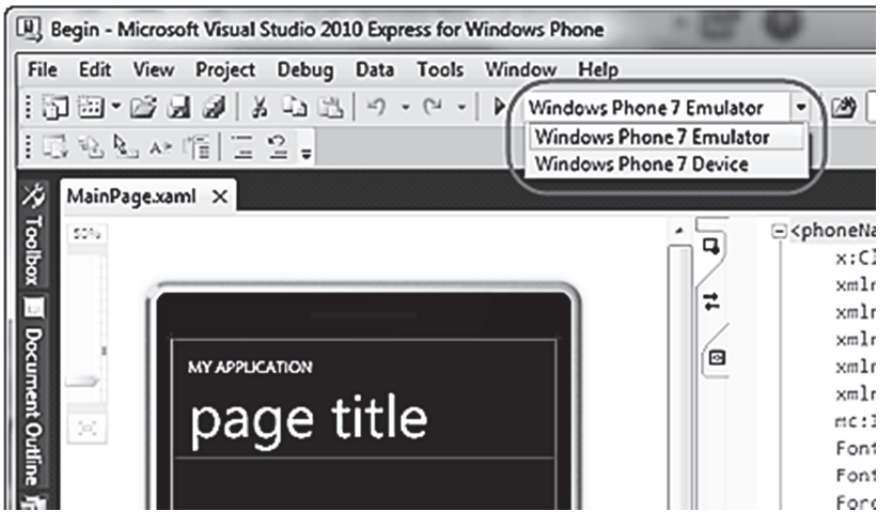


Рис. 2.9. Список выбора устройств отладки

Нажмите F5 или щелкните по зеленому треугольнику для запуска программы в Windows Phone Emulator. На экране появится эмулятор устройства, и начнется процесс установки приложения на эмулятор. Наберитесь терпения и ждите полной загрузки.

Через некоторое время Вы увидите свое приложение в эмуляторе.

Пока программа слишком проста для изучения, поэтому давайте закроем ее – нажмите SHIFT + F5 или щелкните на кнопке Stop на панели инструментов (рис. 2.10), чтобы остановить отладчик и закончить сеанс отладки. Но не закрывайте окно эмулятора.

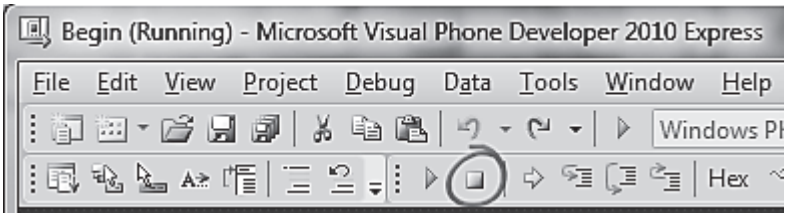


Рис. 2.10. Кнопка Stop на панели инструментов

После начала сеанса отладки значительную часть времени занимают настройка среды эмулятора и запуск приложения. Чтобы упростить отладку, не закрывайте эмулятор во время работы с исходным кодом в Visual Studio. Пока эмулятор работает, остановка текущего сеанса, редактирование исходного кода и последующее создание и развертывание нового образа приложения для запуска нового сеанса отладки выполняются очень быстро.

2.11.5. Проектирование интерфейса пользователя

Теперь, когда мы поняли, как создавать приложения, попробуем создать более сложную программу с элементами пользовательского интерфейса. Мы добавим такие элементы, как заголовок, текстовое поле и кнопка. При использовании приложения нужно ввести какой-либо текст в текстовое поле, и после нажатия кнопки появится сообщение с введенным текстом. Он будет выглядеть примерно, как на рис. 2.11.

Рис. 2.11. Результат работы программы в окне эмулятора

В обозревателе решений дважды щелкните файл MainPage.xaml, чтобы открыть его в конструкторе.

Хотя интегрированная среда разработки поддерживает графические манипуляции с объектами (как обычный визуальный конструктор интерфейса), мы вручную отредактируем код XAML. Для перевода режима редактора в представление XAML и увеличения области обзора дважды щелкните вкладку XAML с правого края окна конструктора.

В разметке XAML найдите элемент контейнера **Grid** с именем **LayoutRoot**. Он предназначен для упорядочивания элементов на странице. Внутри его свойства **RowDefinition** вставьте дополнительную строку между двумя существующими и установите значение свойства **Height** равным **Auto**. В этой строке вскоре появится текстовое поле и кнопка.

```
<Grid x:Name="LayoutRoot" Background="Transparent">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
  </Grid.RowDefinitions>
    ...
 </Grid>
</phone:PhoneApplicationPage>
```

Grid – это элемент разметки, который выступает в качестве контейнера для других элементов управления. Его основная задача – расположение и упорядочение дочерних элементов управления. Существует и другие элементы управления разметкой: Canvas, StackPanel.

Обратите внимание, что корневой элемент **Grid** содержит вложенные элементы, каждый из которых сопоставлен отдельной строке внешней сетки путем определения свойства **Grid.Row**. Найдите элемент **Grid** с именем **TitlePanel**. Задайте свойству **Text** первого элемента **TextBlock** в пределах внутренней сетки строковое значение **Windows Phone 7**. Аналогичным образом задайте свойству **Text** второго элемента **TextBlock** строковое значение **Hello Phone**.

Теперь найдите элемент **Grid** с именем **ContentPanel**, который должен быть пустым, и вставьте в него следующую разметку XAML:

```
<Grid x:Name="LayoutRoot" Background="Transparent">
 ...
 <!--ContentPanel - place additional content here-->
 <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
  <Grid.ColumnDefinitions>
       <ColumnDefinition Width="*" />
       <ColumnDefinition Width="Auto"/>
  </Grid.ColumnDefinitions>
  <TextBox Grid.Column="0" Name="MessageTextBox"
       FontSize="{StaticResource PhoneFontSizeExtraLarge}"
       Margin="20,20,10,20"/>
  <Button Grid.Column="1"
       Name="ClickMeButton"
       Content="Click Me"
       HorizontalAlignment="Right"
       Padding="4"
       Margin="10,20,20,20" />
 </Grid>
</Grid>
...
```

Описание текстового поля можно было также получить просто перетащив мышкой заготовку текстового поля из панели **TOOLS** на изображение эмулятора телефона.

Элемент **Grid** упорядочивает свои дочерние элементы управления на странице на основе ширины каждого столбца, как указано в коллекции ColumnDefinitions. Обратите внимание, что ширина первого столбца указывается как *. Это позволяет столбцу растягиваться и заполнять неиспользуемое пространство в строке после размещения всех остальных столбцов. Ширина второго столбца указывается как Auto, что позволяет изменять размер столбца в соответствии с размером его содержимого.

Чтобы завершить конструирование страницы, добавьте третью строку, которая будет содержать баннер с сообщением, вводимым пользователем. Для создания этой строки вставьте следующую разметку XAML сразу перед конечным тегом внешней сетки:

```
...
<Grid x:Name="LayoutRoot" Background="Transparent">
 ...
        <Grid Grid.Row="2">
                <TextBlock Name="BannerTextBlock"
                        Style="{StaticResource PhoneTextExtraLargeStyle}"
                        Foreground="#FFFF9A00"
                        HorizontalAlignment="Stretch"
                        TextWrapping="Wrap"
                        TextAlignment="Center"
                        FontWeight="Bold" />
         </Grid>
</Grid>
...
```

Щелкните вкладку **Design** с правого края окна, чтобы перейти в режим конструирования.

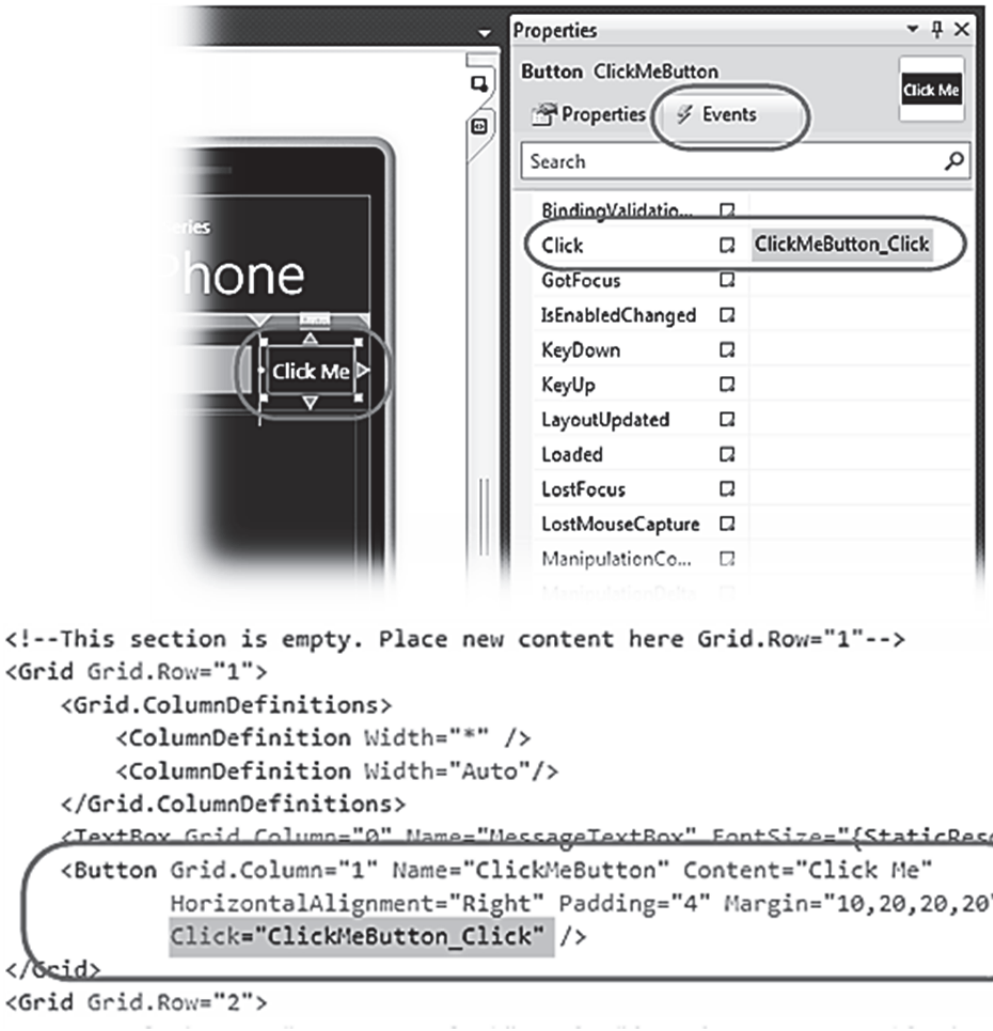
2.11.6. Обработка событий

Теперь необходимо определить обработчики событий, которые отвечают на действия из интерфейса пользователя, в частности событие нажатия кнопки. Включите в конструкторе режим Design. Для этого дважды щелкните вкладку **Design** с правого края окна конструктора. Щелкните кнопку **Click Me** на поверхности конструктора, чтобы выбрать ее, а затем нажмите клавишу F4, чтобы открыть окно свойств этой кнопки. На панели **Properties** щелкните вкладку **Events**, чтобы отобразить окно со списком доступных событий. Найдите в этом списке событие **Click** и введите **ClickMeButton_Click** в текстовом поле рядом с этим событием. Нажмите клавишу Enter, чтобы создать обработчик событий с этим именем, и откройте файл с выделенным кодом для отображения заглушки метода, созданной Visual Studio (рис. 2.12).

Существует альтернативный механизм создания обработчика событий. В Visual Studio можно дважды щелкнуть элемент управления в конструкторе, чтобы создать обработчик для его события по умолчанию: для элементов управления в виде кнопок это событие **Click**.

Реализация метода (который пока является пустым) находится в файле MainPage.xaml.cs. Вставьте следующий код в тело метода ClickMeButton_Click.

```
private void ClickMeButton_Click(object sender, RoutedEventArgs e)
{
   BannerTextBlock.Text = MessageTextBox.Text;
   MessageTextBox.Text = String.Empty;
}
```



Этот код считывает текст, введенный пользователем в текстовом поле, а затем создает баннер с этим текстом.

2.11.7. Проверка приложения

Проверим, работает ли приложение ожидаемым образом. Кроме того, мы зададим точку останова и с помощью отладчика пройдем по исходному коду, анализируя значения переменных, чтобы составить краткое представление о том, как с помощью Visual Studio выполнять отладку приложений, запущенных в эмуляторе.

При необходимости повторно откройте файл с выделенным кодом для страницы MainPage.xaml. Для этого щелкните данный файл правой кнопкой мыши в обозревателе решений и выберите команду **View Code**.

Теперь определите точку останова для прекращения выполнения в обработчике событий для кнопки **Click Me**. Для задания точки останова

найдите первую строку метода **ClickMeButton_Click** исходного файла и щелкните в области серого поля, расположенного на левой стороне окна редактора рядом с этой строкой. Красный кружок (●) указывает положение вставленной точки останова (рис. 2.13).

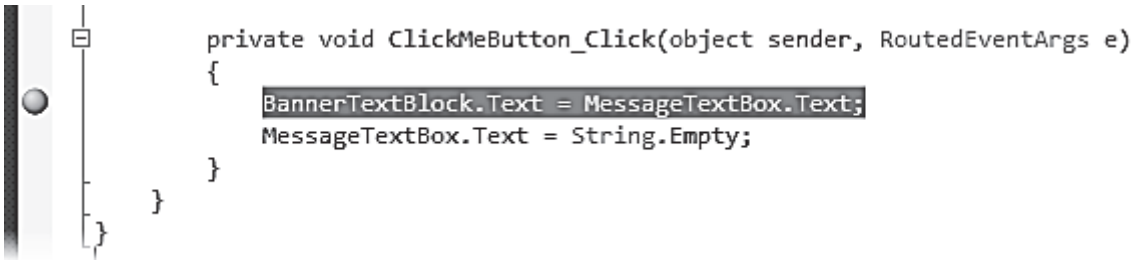


Рис. 2.13. Установка точки останова

Чтобы включить или выключить точку останова, щелкните ● в области поля или щелкните строку, содержащую точку останова, и нажмите клавишу F9.

Для создания и развертывания приложения в эмуляторе Windows Phone нажмите клавишу F5 и начните сеанс отладки. Подождите, пока приложение запустится и появится его главная страница.

В окне эмулятора щелкните текстовое поле, чтобы активировать его. После этого появится экранная панель ввода (SIP). Введя тот или иной текст в текстовое поле, нажмите кнопку рядом с ним. Введенный текст должен отобразиться в верхней части программы.

Вернитесь в Visual Studio. Обратите внимание, что выполнение прекращается в заданной ранее точке останова и следующий выполняемый оператор выделяется желтым (рис. 2.14).

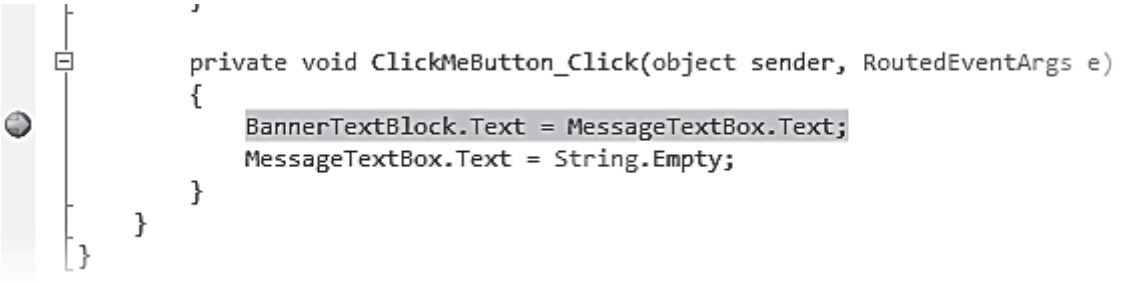


Рис. 2.14. Срабатывание точки останова

Изучите текущее содержимое текстового поля в отладчике. Для этого в окне исходного кода наведите указатель мыши на свойство **MessageTextBox.Text**. Появится окно подсказки (совета) с текущим значением свойства, которое должно совпадать с текстом, введенным в окно

эмулятора. Убедитесь, что указатель находится над частью **Text**. Иначе в подсказке будут отображаться сведения об объекте MessageTextBox (рис. 2.15).

alt="Проверка значений переменных в отладчике">

Нажмите клавишу F10, чтобы в пошаговом режиме выполнить текущую команду и отобразить в баннере текст, соответствующий содержимому текстового поля. Отобразите подсказку для свойства **BannerTextBlock.Text**, чтобы убедиться, что его значение соответствует значению текстового поля (см. рис. 2.15).

```
private void ClickMeButton_Click(object sender, RoutedEventArgs e)
{
    BannerTextBlock.Text = MessageTextBox.Text;
    MessageTextBox.Text = String.Empty;
}
}
```

MessageTextBox.Text "Hello Windows Phone"

Рис. 2.15. Проверка текущих значений переменной

При нажатии клавиши F10 отладчик выполняет текущую команду. А клавиша F11 обеспечивает пошаговое выполнение с заходом в вызываемые методы и функции. В этом случае, если команда включает вызов метода, отладчик выполняет заход в соответствующий метод для его отладки.

Нажмите клавишу F10 еще раз, чтобы выполнить следующий оператор и очистить содержимое текстового поля. Вновь отобразите совет для свойства **MessageTextBox.Text**, которое по-прежнему доступно, и убедитесь, что теперь оно пустое. Нажмите клавишу F5, чтобы возобновить выполнение приложения. Вернитесь в эмулятор Windows Phone.

Нажмите в эмуляторе кнопку **Back**, чтобы перейти на предыдущую страницу. Обратите внимание, что при этом сеанс отладки завершается и в отладчике отображается главное меню: Вы уходите с первой (и единственной) страницы приложения (закрываете ее), а других активных приложений нет.

Чтобы возобновить отладку после окончания текущего сеанса, нажмите клавишу F5 для повторного запуска приложения и присоединения отладчика. Однако обратите внимание, что при этом приложение запустится заново, а предыдущее состояние будет недоступно.

При закрытии эмулятора приложение останавливается, а отладчик отсоединяется. При отсоединении отладчик Visual Studio отображает сообщение о разрыве подключения к устройству.

Вы также можете посмотреть видео о том, как создать первое приложение Windows Phone 7 [2].

2.12. Создание страницы с навигацией

Теперь мы узнаем, как можно сделать навигацию между страницами. В предыдущем подразделе приложение состояло всего лишь из одной страницы. Иногда этого недостаточно. В Silverlight навигация между страницами осуществляется очень просто. В этом Вы сейчас сами убедитесь. Заодно Вы увидите, как работает кнопка **Back** (Назад) в подобных приложениях.

2.12.1. Создание приложения с навигацией

Запустите Visual Studio и создайте новый проект **PageNavigation** (см. подраздел 2.11). Далее необходимо добавить еще несколько новых страниц. Щелкните правой кнопкой мыши на имени проекта в Solution Explorer и выберите команду **Add→New Item...** и в диалоговом окне выберите элемент **Windows Phone Portrait Page** (портретная ориентация) (рис. 2.16).

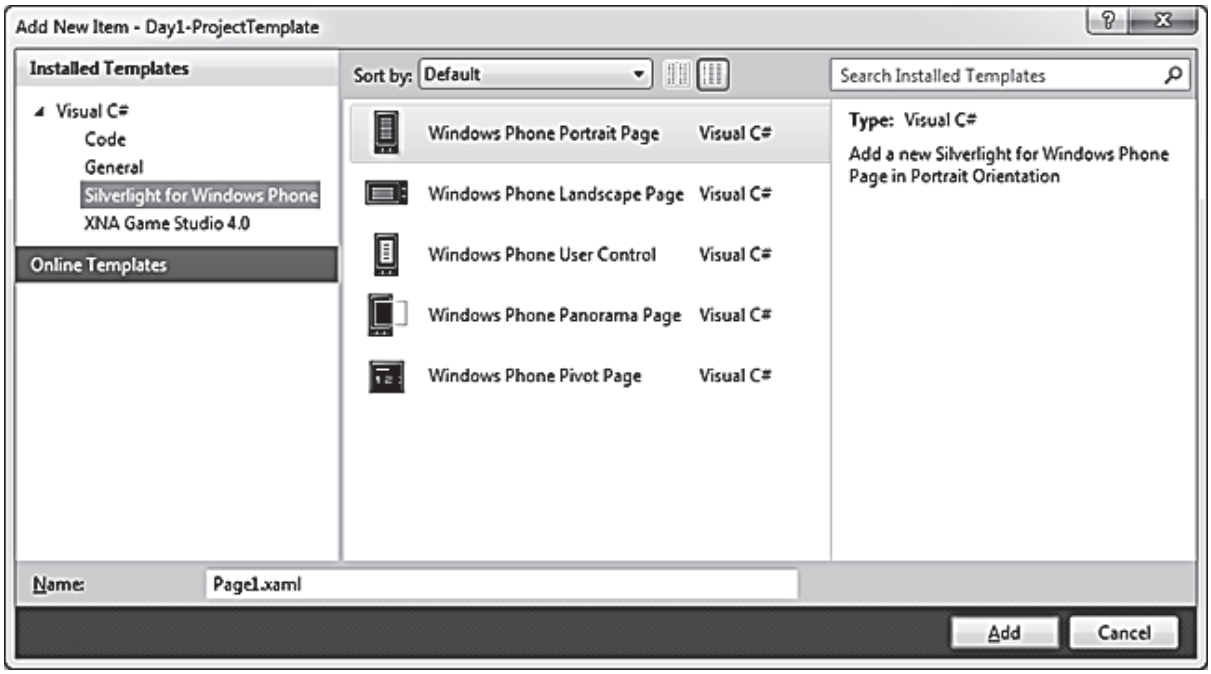


Рис. 2.16. Выбор нового проекта

Сейчас мы пока не будем обсуждать вопросы, связанные с ориентацией экрана. Создайте три новых страницы: Page1.xaml, Page2.xaml, Page3.xaml. Так как они выглядят совершенно одинаково, нам будет трудно ориентироваться среди них. Предлагается сделать следующее. Откройте каждую созданную страницу и измените текст в них. Вместо слов **PAGE NAME** используйте имена котов: Рыжик, Барсик, Васька. Теперь ошибиться будет сложно.

2.12.2. Создание гиперссылок на другие страницы

Навигацию между страницами будем делать при помощи гиперссылок. Найдите на панели инструментов элемент **HyperlinkButton** и добавьте трижды данный элемент на панель эмулятора. После добавления измените код XAML следующим образом:

```
<HyperlinkButton Content="Рыжик" NavigateUri="/page1.xaml"
                 Name="hiperlinkbutton1" Height="30" Width="200"
                 HorizontalAlignment="Left" VerticalAlignment="Top"
                 Margin="0,6,0,0" />

<HyperlinkButton Content="Барсик" NavigateUri="/page2.xaml"
                 Name="hiperlinkbutton2" Height="30" Width="200"
                 HorizontalAlignment="Left" VerticalAlignment="Top"
                 Margin="0,6,0,0" />

<HyperlinkButton Content="Васька" NavigateUri="/page3.xaml"
                 Name="hiperlinkbutton3" Height="30" Width="200"
                 HorizontalAlignment="Left" VerticalAlignment="Top"
                 Margin="0,6,0,0" />
```

Мы поменяли у гиперссылок текст, а также установили размеры и расположение на странице. При желании этого же результата можно добиться изменяя соответствующие свойства в окне свойств. И самое главное – мы указали в атрибуте **NavigateUri** нужные имена страниц.

Удивительно, мы не написали еще ни одной строчки кода на C#, но тем не менее приложение уже работает. Убедитесь сами. Запустите приложение и попробуйте нажимать на ссылки. Вы будете переходить на первую, вторую или третью страницу в зависимости от выбранной ссылки. Обратите внимание, что для возврата на основную страницу Вы можете использовать аппаратную кнопку **Back**. При этом если Вы находитесь на главной странице и нажмете на кнопку **Back**, то тем самым Вы закроете приложение.

2.12.3. Навигация через код

Мы осуществили навигацию при помощи XAML-кода. Такого же результата можно добиться и через код на C#. Для этого добавим в проект три новых элемента **Button** (кнопка). Чтобы не писать одинаковый код для каждой кнопки, создадим общий обработчик событий для них. Для этого нам нужно знать имена кнопок.

[XAML]

```
<Button Content="Рыжик" Height="72" HorizontalAlignment="Left" Margin="12,40,0,0"
     Name="button1" VerticalAlignment="Top" Width="160"
```

```
        Click="Button_Click" />
    <Button Content="Барсик" Height="72" HorizontalAlignment="Left" Margin="12,120,0,0"
        Name="button2" VerticalAlignment="Top" Width="160"
        Click="Button_Click" />
    <Button Content="Васька" Height="72" HorizontalAlignment="Left" Margin="12,200,0,0"
        Name="button3" VerticalAlignment="Top" Width="160"
        Click="Button_Click" />
```

[C#]

```
private void Button_Click(object sender, RoutedEventArgs e)
{
   Button clickedbutton = sender as Button;
   switch (clickedbutton.Name)
   {
      case "button1": NavigationService.Navigate(new Uri("/page1.xaml",
UriKind.Relative));
         break;
      case "button2":
         NavigationService.Navigate(new Uri("/page2.xaml", UriKind.Relative));
         break;
      case "button3":
         NavigationService.Navigate(new Uri("/page3.xaml", UriKind.Relative));
         break;
   }
}
```

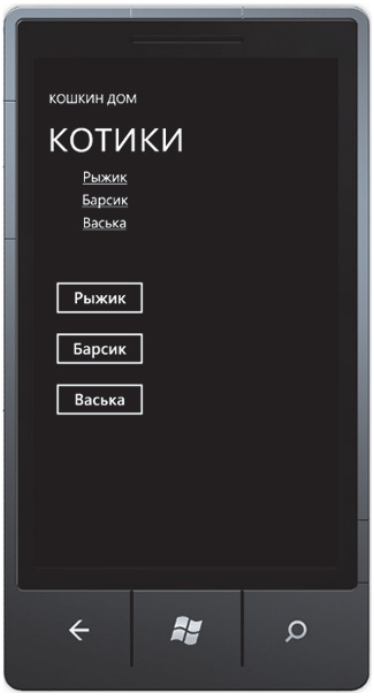


Рис. 2.17. Вид программы в окне эмулятора

Снова запустите проект и убедитесь, что навигация при помощи кнопок работает так же, как и в примере с гиперссылками (рис. 2.17).

Для перехода назад вы можете использовать метод **NavigationService.GoBack**, который возвращает к экземпляру предыдущей страницы. Конечно, это дублирует функциональность кнопки **Back**, так что Вы, скорее всего, будете вызывать этот метод как часть какой-либо другой функциональности.

```
private void button1_Click(object sender,
RoutedEventArgs e)
{
   NavigationService.GoBack();
}
```

2.12.4. Передача параметров

Иногда требуется не просто перейти на другую страницу, но и передать ей некоторые данные с предыдущей страницы. Добавьте на первую страницу текстовое поле и кнопку под именем **passParam**.

Добавьте код для обработчика щелчка кнопки:

```
private void passParam_Click(object sender, RoutedEventArgs e)
        {
NavigationService.Navigate(new Uri("/SecondPage.xaml?msg=" + textBox1.Text,
UriKind.Relative));
        }
```

Вы видите, что при навигации на вторую страницу мы передаем строковые данные, которые берутся из текстового поля. Чтобы получить передаваемые данные, добавьте на второй странице текстовый блок (TextBlock) под именем **textBlock1**. В файле SecondPage.xaml.cs создайте следующий метод:

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
        {
           base.OnNavigatedTo(e);
           string msg = "";
        if (NavigationContext.QueryString.TryGetValue("msg", out msg)) textBlock1.Text = msg;
        }
```

Запустите приложение, введите текст на первой странице и убедитесь, что он отображается на второй странице.

2.12.5. Аппаратная кнопка Back

Одним из требований к устройствам под управлением Windows Phone 7 является наличие аппаратной кнопки **Back**. По умолчанию кнопка **Back** работает как в обычном браузере. Она автоматически запоминает посещаемые страницы приложения, а также закрывает само приложение, если приложение состоит из одной страницы или пользователь вернулся к основной странице из других страниц и больше некуда возвращаться.

Переопределяем работу кнопки Back.

Если Вас не устраивает логика работы кнопки **Back** по умолчанию, то Вы можете переопределить ее работу в нужном Вам направлении. Делается это очень просто:

```
protected override void OnBackKeyPress(System.ComponentModel.CancelEventArgs e)
        {
           // Ваш код здесь
           e.Cancel = true;  //Cancels the default behavior.
        }
```

Давайте попробуем посмотреть работу переопределенной кнопки на примере. Возьмем предыдущий проект с навигацией по страницам **PageNavigation** и откроем редактор кода для страницы **Page3.xaml.cs**. Напишем следующий код:

```
protected override void OnBackKeyPress(System.ComponentModel.CancelEventArgs e)
    {
      // Ваш код здесь
      NavigationService.Navigate(new Uri("/page1.xaml", UriKind.Relative));
      e.Cancel = true;  //Cancels the default behavior.
    }
```

Запустите проект, нажмите на кнопку или ссылку **Васька**. Далее нажмите аппаратную кнопку **Back** и убедитесь, что теперь возвращаетесь не на основную страницу MainPage.xaml, а на страницу Page1.xaml.

Как видите, код работает – мы переопределили работу аппаратной кнопки под свои нужды. Но не спешите радоваться. С переопределением кнопки **Back** нужно быть очень осторожным. Если Вы еще не закрыли пример и находитесь на странице **Page1.xaml** (страница «Рыжик»), то нажмите на **Back** еще раз. Как и ожидалось, Вы вернетесь обратно на Page3.xaml, с которой Вы пришли. Нажмите кнопку **Back** и заметьте, что снова попали на страницу «Рыжик». Получился замкнутый круг – Вы больше не можете никуда попасть и попеременно оказываетесь на страницах Page1.xaml и Page3.xaml.

Поэтому тщательно тестируйте свою программу, если решили изменить поведение кнопки **Back**. Иначе, Вы рискуете попасть в неприятную ситуацию.

Кнопка Back для XNA-приложений.

Описанный нами пример в основном применяется для Silverlight-приложений. Мы еще не знакомы с разработкой приложений на основе XNA, но в XNA-программах есть класс **GamePad**, имеющий свойство **Buttons.Back**. Поэтому вам придется встречаться с такой конструкцией:

```
// выйти, если нажата кнопка Back
if(GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
```

2.13. Ориентация дисплея

Вам необходимо запомнить, что существуют два положения экрана. Вертикальное расположение называется портретным (Portrait), а горизонтальное – альбомным (Landscape). По умолчанию приложения на Silverlight запускаются в портретном режиме, а XNA-приложения запускаются в альбомном, т.к. игры лучше смотрятся именно в этом режиме (посмотрите на ваш монитор). Но Вы можете управлять режимами исходя

из Ваших задач. Жесткой привязки к ориентации нет. Более того, часто необходимо поддерживать два режима одновременно.

2.13.1. Рекомендация по проектированию интерфейса

Если приложение поддерживает ввод текста, Вы должны установить альбомную ориентацию из-за возможности существования аппаратной клавиатуры.

Существуют различные способы, которыми можно гарантировать правильное отображение содержимого как в портретной, так и в альбомной ориентации. Два основных метода – это прокрутка (scrolling) и сетка (grid layout). Эти методы могут использоваться отдельно или в сочетании друг с другом.

Scrolling использует элемент управления **StackPanel**, который находится в элементе управления **ScrollViewer**. Используйте этот метод, если содержимое отображается в виде списка или если различные элементы управления следуют один за другим на странице. **StackPanel** позволяет установить порядок расположения дочерних элементов одного за другим, а элемент управления **ScrollViewer** позволяет прокручивать содержимое **StackPanel**, если элементы пользовательского интерфейса не помещаются на экране.

2.13.2. Управление ориентацией экрана в Silverlight

Если Вы откроете предыдущие примеры на Silverlight и повернете устройство набок, то обнаружите, что изображение на экране не изменило свое расположение при смене ориентации. Это легко исправить. Посмотрите код в файле MainPage.xaml и найдите строчку:

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

Данный код говорит о том, что проект поддерживает только портретный режим и в этом же режиме запускается. Свойство **SupportedOrientations** поддерживает следующие режимы (рис. 2.18):

- Portrait (по умолчанию);
- Landscape;
- PortraitOrLandscape.

Достаточно заменить указанную выше строчку на **SupportedOrientations="PortraitOrLandscape"**, и ваше приложение будет реагировать на смену режима автоматически.

Вероятно, вы уже догадались, что свойство **Orientation** отвечает за начальный режим во время запуска приложения. Только Вы не должны забывать о поддержке выбранного режима. Иными словами, если Вы хоти-

те, чтобы приложение запускалось в альбомном режиме, то у свойства **SupportedOrientation** должно быть значение **Landscape** или **PortraitOrLandscape**. Возможно, Вас удивит, что свойство поддерживает целых шесть различных значений:

- Landscape;
- LandscapeLeft;
- LandscapeRight;
- Portrait;
- PortraitDown;
- PortraitUp.

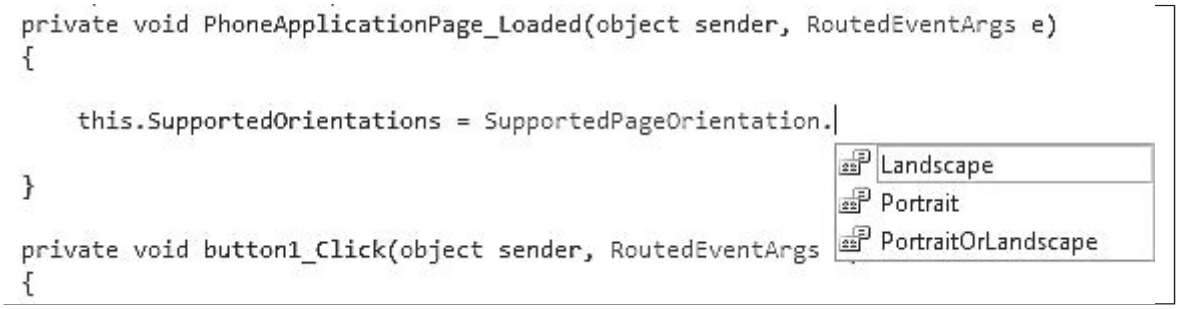


Рис. 2.18. Набор свойств ориентации экрана

Используя эти значения, Вы можете даже задать портретный режим вверх тормашками (PortraitUp). PortraitDown – это обычный портретный режим, а Portrait – общее свойство для двух портретных режимов, когда уточнение не требуется. Аналогично справедливо и для альбомных режимов.

2.13.3. События изменения ориентации

Итак, Вы можете настроить ориентацию экрана еще во время разработки дизайна приложения через свойство **SupportedOrientation**. В этом случае система сама позаботится о нужном режиме. Тут нужно проявить аккуратность. Как правило, разработчик не препятствует пользователю вращать устройством во время работы с приложением. Но здесь может возникнуть одна небольшая неприятность (рис. 2.19). Предположим, Вы написали программу-калькулятор, который красиво смотрится в портретном режиме. Вы ввели поддержку альбомного режима, и пользователь развернул телефон горизонтально. Как видите, нижняя часть интерфейса программы стала невидимой, уйдя за пределы экрана.

Рис. 2.19. Вид программы при разной ориентации экрана

Если рассматривать данный конкретный случай, то можно посоветовать временно убрать заголовок программы при работе в альбомном режиме. В этом случае мы освободим место для кнопок калькулятора. Также можно поиграться с размерами кнопок и их расположением, что они более красиво смотрелись в другом режиме. Вот как это выглядит в коде:

```
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 Day4_DeviceOrientation
{
        public partial class MainPage : PhoneApplicationPage
        {
                // Constructor
                public MainPage()
                {
                        InitializeComponent();
this.OrientationChanged +=  new
                EventHandler<OrientationChangedEventArgs>(MainPage_OrientationChanged);
```

```
                }

        void MainPage_OrientationChanged(object sender, OrientationChangedEventArgs e)
                {
if((e.Orientation==PageOrientation.LandscapeRight)||(e.Orientation==
                                                        PageOrientation.LandscapeLeft))
                        {
                                TitlePanel.Visibility = Visibility.Collapsed; // скрываем
                        }
else if((e.Orientation==PageOrientation.PortraitDown)||(e.Orientation == PageOrienta-
tion.PortraitUp))
                        {
                                TitlePanel.Visibility = Visibility.Visible; // показываем
                        }
                }
        }
}
```

В этом примере при повороте экрана наступает событие **OrientationChanged**. В обработчике данного события проходит проверка текущего режима. Если мы видим, что устройство находится в альбомном режиме, то прячем заголовок приложения, и наоборот, если приложение стало работать в портретном режиме, то заголовок выводим на экран (рис. 2.20).

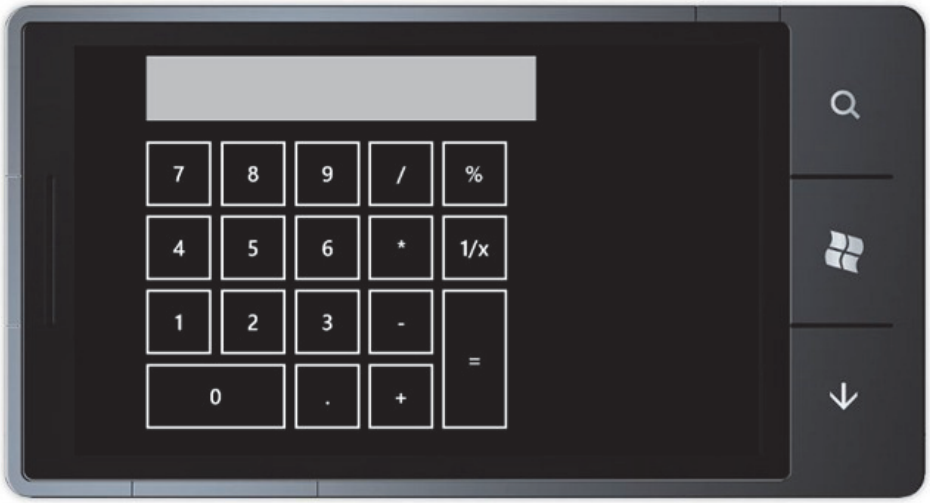


Рис. 2.20. Правильная компоновка изображения на экране эмулятора

Однако вернемся к ориентации. Итак, чтобы узнать текущее состояние ориентации, нам нужно использовать в обработчике события **OrientationChanged** перечисление **Orientation**:

```
private void PhoneApplicationPage_OrientationChanged(object sender,
                                                        OrientationChangedEventArgs e)
```

```
{
    MessageBox.Show(this.Orientation.ToString());
}
```

2.14. Темы и расцветка

По умолчанию все телефоны под управлением Windows Phone 7 используют черные цвета. Это не прихоть разработчиков, а необходимая мера для продления работы телефона. Дело в том, что AMOLED-экраны, используемые в телефонах Windows Phone 7, имеют такую особенность: при черном цвете потребление заряда батареи гораздо ниже, чем при светлом. Но пользователь может изменить по своему желанию используемую тему на другую, более подходящему собственному вкусу.

Рис. 2.21. Примеры основных цветов смартфона

Согласованный внешний вид и соблюдение рекомендаций дизайна «Metro» крайне важны для приложений Windows Phone. На рис. 2.21 показаны примеры различных основных цветов (accent colors), а также светлая и тёмная темы.

2.14.1. Цветовые темы

Сначала несколько слов о том, как поменять цветовую тему. На стартовом экране коснитесь правой стрелки в верхнем правом углу и перейдите к странице **Settings** (Настройки). Здесь Вы можете выбрать визуальную

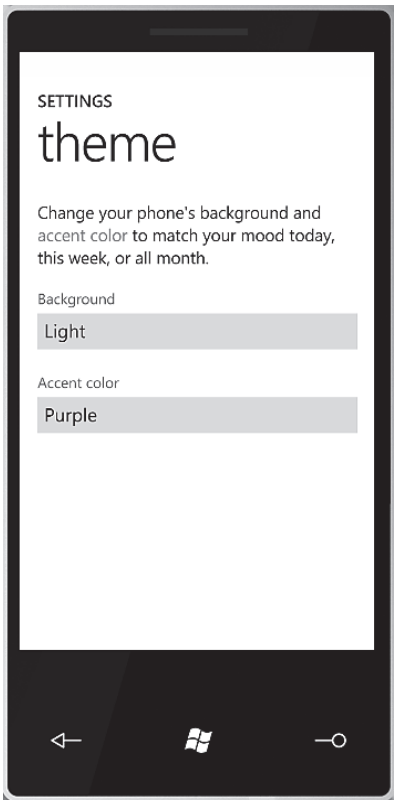


Рис. 2.22. Схема White

тему: темную (светлый текст на темном фоне, используется по умолчанию) и светлую (темный текст на светлом фоне – рис. 2.22). Если выбрать новую тему и запустить примеры из прошлых подразделов, то увидим, что цвета темы изменяются автоматически. Обратите внимание, что при смене темы меняется только цвет. Другие ресурсы (шрифты, размеры элементов) не меняются.

По своей сути тема – это набор ресурсов, используемых для персонализации визуальных элементов Windows Phone. Вы можете создавать свои приложения, которые будут выглядеть как родные системные приложения. Стилевые свойства включают в себя фоновые цвета и расцветку.

Таким образом, тема Windows Phone является комбинацией фона и расцветки. Цвет фона – это общий цвет для всей поверхности страницы, а цвет расцветки – это цвет, используемый в элементах управления и других визуальных элементах. Для фона есть два варианта: **Dark** и **Light**, для расцветки используются 10 вариантов.

2.14.2. Расцветка

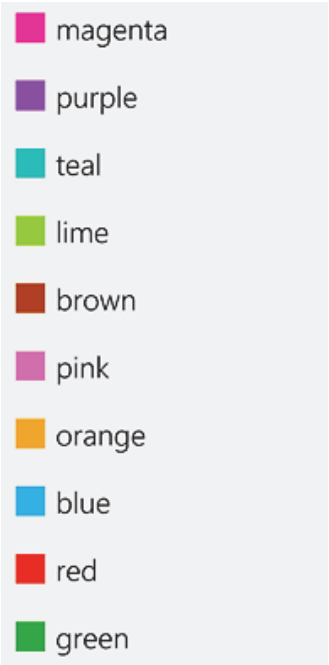


Рис. 2.23. Типовые расцветки

Кроме выбора темы (темной или светлой) система позволяет также выбрать расцветку из десяти различных цветов, которые будут использоваться в учебных темах. Выбранная расцветка позволит раскрасить нужным цветом значки, плитки, ссылки и т.д. Список возможных цветов: Magenta, Purple, Teal, Lime, Brown, Pink, Orange, Blue, Red, Green (рис. 2.23).

Производитель телефона может добавить еще свой одиннадцатый цвет, но Вам стоит избегать использования такого цвета в коде, т.к. нет возможности убедиться, что данный цвет используется у других пользователей, что вызовет ошибку приложения.

2.14.3. Использование системных цветов

Вы, конечно, можете использовать свои цвета в приложении, чтобы подчеркнуть свою индивидуальность. Но когда Вы меняете цвета, подумайте, как будет выглядеть интерфейс в другой теме и при другой расцветке. Для серьезных программ предпочтительнее использовать системные цвета, не прописывая жестко цветовые значения. Обратите внимание, что встроенные элементы управления по умолчанию подстраиваются под тему и расцветку. Вот как меняет свой цвет элемент Slider (Слайдер) при двух разных выбранных расцветках (рис. 2.24).

Рис. 2.24. Типовые расцветки

Вот как выглядит приложение в разных вариантах расцветки (выбраны светлая и темная тема, а также цвета Brown, Teal и Purple (рис. 2.25)).

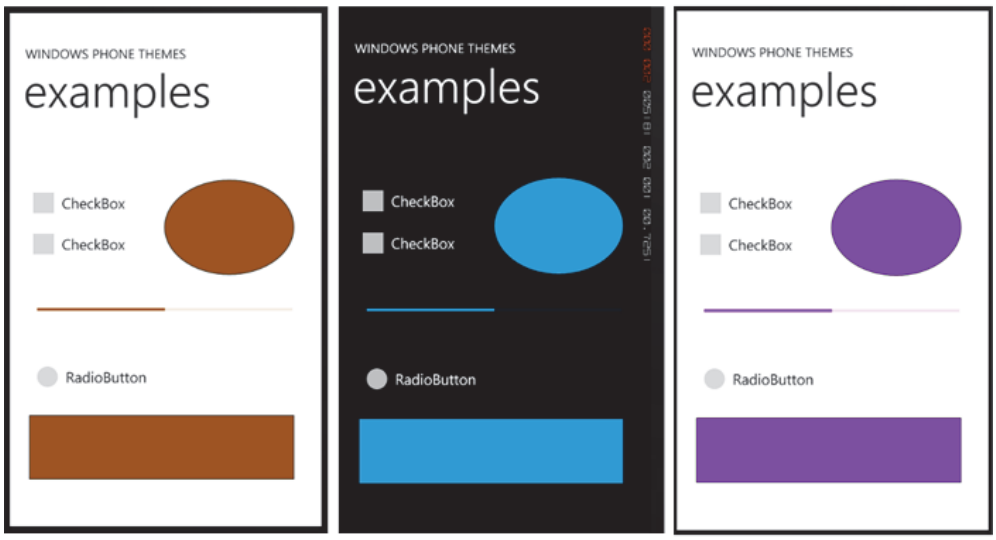


Рис. 2.25. Вид приложений в типовых расцветках

2.14.4. Использование встроенных стилей

Так же, как и каскадные таблицы стилей (CSS) совместно с HTML, XAML позволяет Вам применять те же настройки для свойств элементов управления, используя специальный синтаксис, называемый расширением разметки. С помощью стилей и ресурсов Вы можете повторно использо-

вать настройки и создать для Вашего приложения согласованный внешний вид.

Существует множество встроенных стилей и ресурсов для использования в проектах Windows Phone, которые соответствуют требованиям дизайна «Metro» и подходят как для светлой, так и для тёмной темы. Эти ресурсы включают кисти, цвета, шрифты, стили текста и темы.

Необходимые ресурсы подключаются к приложению при запуске. Вы можете установить ресурсы в дизайнере при помощи свойств, а также в XAML при помощи элемента разметки **{StaticResource}**.

В следующем примере показано, как привязать к фону (background) элемента управления **Button** кисть, являющуюся встроенным в Windows Phone ресурсом, с помощью расширения разметки:

```
<Button Content="Button" Height="72" Background="{StaticResource PhoneAccentBrush}" Width="160" />
```

В данном коде мы присвоили цвет кнопки через разметку **StaticResource**, задав кисть с именем **PhoneAccentBrush**. Если пользователь выберет в настройках телефона другую расцветку, то цвет кнопки в Вашем приложении также изменится на выбранный цвет. Таким образом Ваше приложение будет соответствовать цветовым предпочтениям пользователя.

На рис. 2.26 показано, как привязать к фону (background) элемента управления **Button** кисть, являющуюся встроенным в Windows Phone ресурсом, с помощью окна **Properties**.

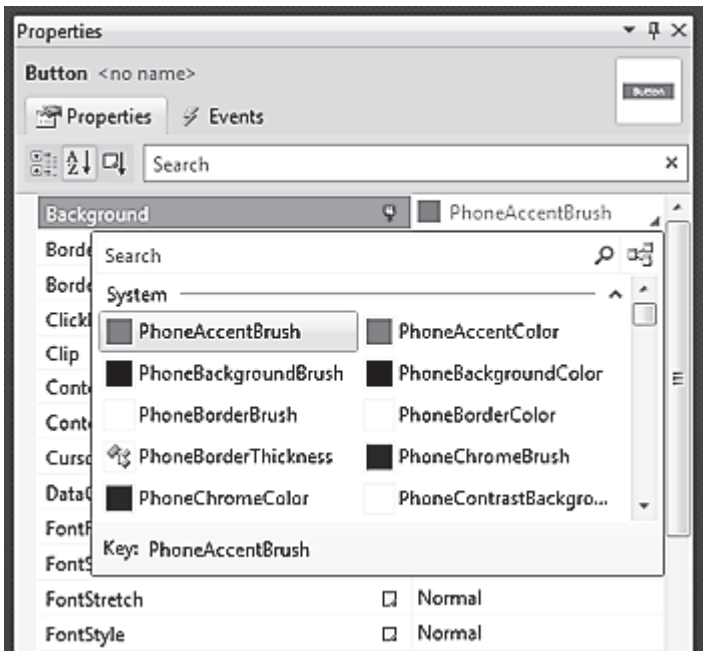


Рис. 2.26. Назначение элементу управления новой кисти

Поскольку фон предыдущего примера имеет значение **PhoneAccentBrush**, цвет кнопки будет основываться на текущем основном цвете (accent color), выбранном пользователем. На рис. 2.27 показано, как кнопка выглядит, когда пользователь выбирает в качестве основного цвета синий или зеленый.

Рис. 2.27. Вид кнопки при стандартных расцветках

Рекомендация по проектированию интерфейса: если основной или фоновый цвет элемента управления задан явно, убедитесь, что его содержание одинаково хорошо видно как при темной, так и при светлой теме оформления. Если указанного цвета не видно, также явно задайте фон или основной цвет, чтобы он был достаточно контрастным, или выберите более подходящий цвет.

Также можно использовать ресурсы, связанные со шрифтами: имя шрифта, его размер и т.п.

```
<TextBlock Height="45"
   HorizontalAlignment="Left"
        Margin="20,154,0,0"
        Name="textBlock1"
        Text="TextBlock"
        VerticalAlignment="Top"
        Width="213"
        FontFamily="{StaticResource PhoneFontFamilySemiLight}"
        FontSize="{StaticResource PhoneFontSizeLarge}"/>
```

Если Вам нужно динамически изменять стили, то можно это сделать через код следующим образом:

```
Color backgroundColor = (Color)Application.Current.Resources["PhoneBackgroundColor"];
```

Список доступных встроенных стилей и ресурсов для Windows Phone Вы можете увидеть в [3]. Дополнительные сведения о расширениях разметки Вы можете получить в [4].

Если Вам интересно посмотреть, как устроена реализация тем, то найдите в папке **%programfiles%\Microsoft SDKs\Windows Phone\v7.0\Design** файл **ThemeResources.xaml** и откройте его текстовым редактором.

Кстати, дизайн приложений, использующих другие цвета, удобнее делать в программе Expression Blend 4, которая специально предназначена для дизайнеров. Откройте Ваш проект в Expression Blend. Для этого щелкните правой кнопкой мыши на имени проекта в Solution Explorer и выберите команду **Open In Expression Blend…** (рис. 2.28).

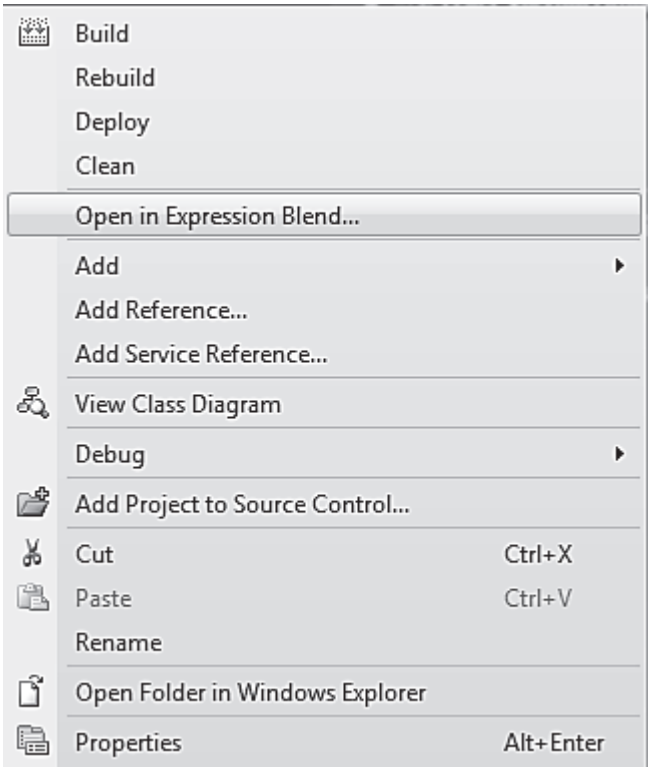


Рис. 2.28. Меню открытия инструмента Expression Blend

В открывшейся программе найдите вкладку **Device** (в верхней части окна) и перейдите на нее (рис. 2.29).

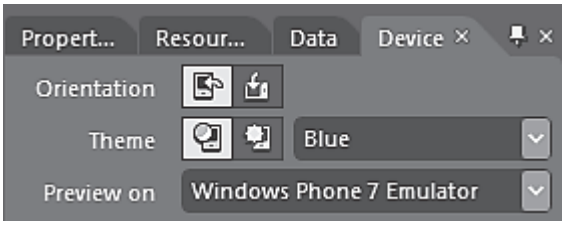


Рис. 2.29. Вид вкладки Device

Вкладка позволяет увидеть, как будет выглядеть приложение с различными темами и расцветкой.

2.14.5. Color Resources

Перейдем теперь на вкладку **Color Resources**, которую можно найти внутри вкладки **Properties**, выбрав какой-либо элемент управления. Window Phone 7 имеет множество стандартных цветов, которые Вы можете использовать в своих приложениях. Например, на картинке можно увидеть, что предлагаемые Expression Blend стандартные цвета **PhoneAccentColor** и **PhoneBackgroundColor** меняются в зависимости от темы, выбранной на вкладке **Device**. На рис. 2.30, *а* выбрана тёмная тема и синяя расцветка, а на рис. 2.30, *б* светлая тема и оранжевая расцветка.

а) б)

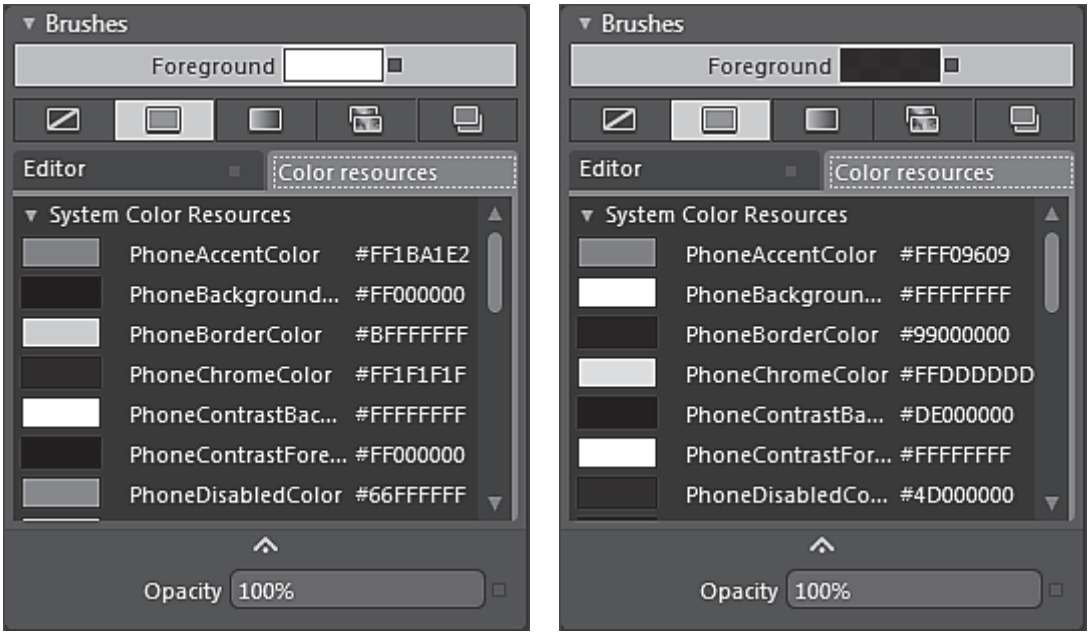


Рис. 2.30. Окна назначения цветовых тем и расцветок

Выбирая эти цвета для приложения, Вы подключаете предопределенные системные значения цветов, и когда пользователь будет менять тему или расцветку, то приложение также подстроится под новые настройки. Пример такого подхода продемонстрирован ниже. Добавим в приложение из прошлых подразделов прямоугольник с градиентной заливкой, начиная с цвета **PhoneBackgroundColor** и заканчивая цветом **PhoneAccentColor**. Кроме того, цвет заголовка страницы также использует выбранную расцветку.

```
<!--LayoutRoot is the root grid where all page content is placed-->
<Grid x:Name="LayoutRoot" Background="Transparent">
<Grid.RowDefinitions>
   <RowDefinition Height="Auto"/>
```

```
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <Rectangle Stroke="Black" Grid.RowSpan="2">
        <Rectangle.Fill>
            <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
            <GradientStop Color="{StaticResource PhoneBackgroundColor}" Offset="0"/>
            <GradientStop Color="{StaticResource PhoneAccentColor}" Offset="1"/>
            </LinearGradientBrush>
        </Rectangle.Fill>
    </Rectangle>

    <!--TitlePanel contains the name of the application and page title-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="КОШКИН ДОМ" Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="котики" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}">
            <TextBlock.Foreground>
                <SolidColorBrush Color="{StaticResource PhoneAccentColor}"/>
            </TextBlock.Foreground>
        </TextBlock>
```

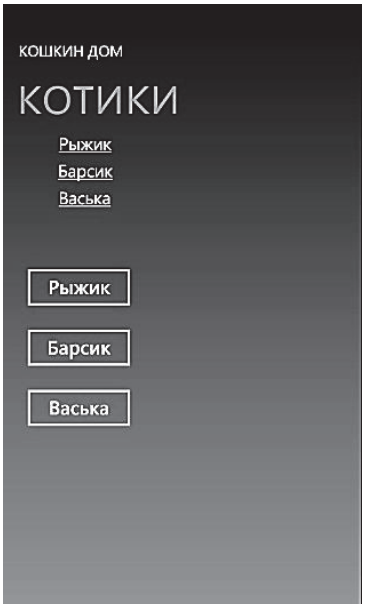


Рис. 2.31. Вид программы в эмуляторе

Теперь когда пользователь будет менять тему или расцветку на своем устройстве, приложение также будет использовать выбранные цвета. Обратите внимание, что заголовок приложения (слово «Котики») выводится цветом Lime, который был установлен в настройках телефона (рис. 2.31).

При разработке собственных элементов также придерживайтесь подобного стиля – используйте стандартные значения цветов PhoneBackgroundColor, PhoneAccentColor и т.п.

2.14.6. Создание собственных стилей

Если Вы хотите создать свой собственный стиль, то как правило, Вы должны объявить стиль как ресурс страницы или панели и применить его в качестве статического ресурса с помощью расширения разметки. Каждый стиль, как правило, имеет ключ (key), который используется для ссылки на него в дальнейшем, и целевой тип (target type), который указывает, к какому типу элементов управления он может быть применен. Основная часть стиля – это коллекция объектов Setter, которые содержат параметры Property (свойство) и Value (значение). Вы можете создавать стили в Visual Studio, указывая их прямо в коде XAML, или использовать Expression Blend, который позволяет создавать стили более визуализированным способом. При создании ресурсов, которые устанавливают цвета, Вы должны убедиться, что Ваш выбор цветов одинаково хорошо выглядит как в светлой, так и в тёмной темах.

2.15. Application Bar

В Windows Phone 7 есть специальный элемент – Application Bar (панель приложения), который представляет собой набор круглых значков (рис. 2.33) в нижней части приложения (рис. 2.32).

Рис. 2.32. Работа программы в эмуляторе

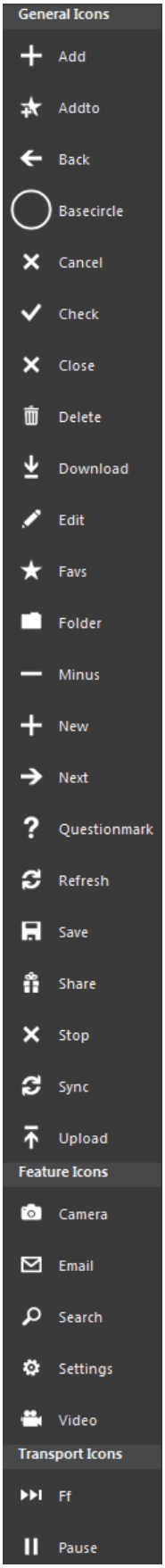


Рис. 2.33. Панель

Коснувшись значка, Вы можете перейти на нужную страницу. При этом сами значки остаются на месте. По сути это схоже с полоской меню в обычных настольных приложениях.

2.15.1. Добавление Application Bar (XAML)

Если Вы создаете новый проект, то там уже есть заготовка для добавления Application Bar в приложение. Вам нужно снять комментарии с кода в файле MainPage.xaml. Вернемся к нашему примеру и уберем комментарии:

```
<phone:PhoneApplicationPage.ApplicationBar>
        <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
            <shell:ApplicationBarIconButton IconUri="/Images/appbar_button1.png"
                        Text="Button 1"/>
            <shell:ApplicationBarIconButton IconUri="/Images/appbar_button2.png"
                        Text="Button 2"/>
            <shell:ApplicationBar.MenuItems>
                <shell:ApplicationBarMenuItem Text="MenuItem 1"/>
                <shell:ApplicationBarMenuItem Text="MenuItem 2"/>
            </shell:ApplicationBar.MenuItems>
        </shell:ApplicationBar>
    </phone:PhoneApplicationPage.ApplicationBar>
```

Если Вы создаете проект с нуля, то кроме указанного выше кода необходимо добавить пространства имен (не забудьте установить также ссылки):

```
xmlns:phone="clr-namespace:Microsoft.Phone.Controls;
assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
```

После того, как Вы уберете комментарии, в программе появятся два значка на панели приложения. Запустите проект, чтобы убедиться в этом. Всего в Application Bar может содержаться от 1 до 4 значков. Высота панели равна 72 пикселам.

Кстати, обратите внимание, что в коде для значков указаны пути к изображениям (например, /Images/appbar_button1.png), которые не являются частью проекта. Если Вы запустите проект, то в Application Bar будут выводиться значки X для ApplicationBarIconButtons. Безусловно, Вы можете создать собственные значки, добавить их в проект и использовать их в своей программе. Но можно пойти другим путем и использовать уже готовые значки для этих целей.

Откроем проект в Expression Blend и найдем объекты **ApplicationBarIconButton** в дереве объектов **Objects and Timeline** (рис. 2.34).

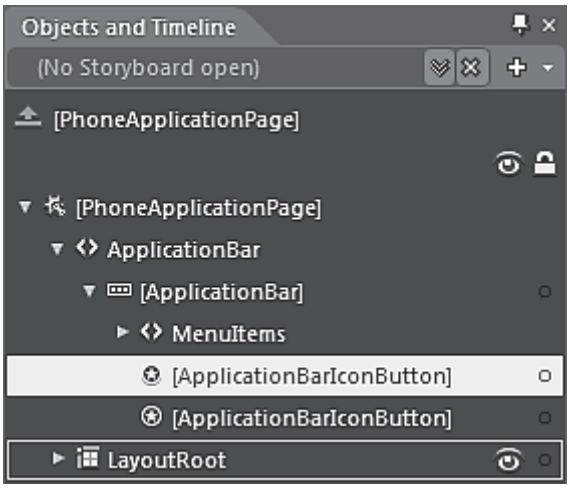


Рис. 2.34. Выбор объекта в дереве объектов

Щелкните на одном из них и изучите вкладку **Properties** (рис. 2.35).

Щелкнув по выпадающему списку **IconUri**, Вы можете выбрать из множества предопределенных значков. Также Вы можете задать текст для кнопок панели приложения.

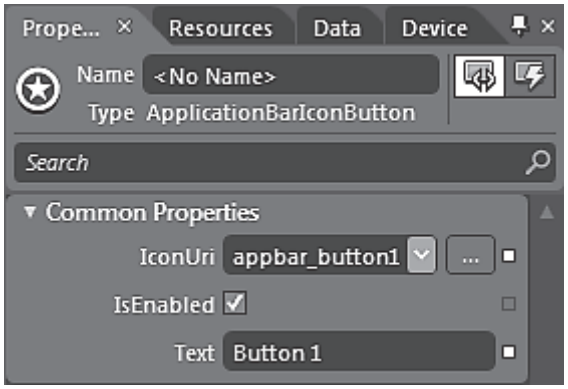


Рис. 2.35. Получение свойств объекта

Пусть Вас не пугает, что стандартные значки имеют белый цвет на черном фоне. Если в телефоне будет использоваться светлая тема (о темах говорилось выше), то значки поменяют свой цвет на противоположный автоматически. Вам нужно взять это на заметку, если собираетесь использовать собственные значки.

2.15.2. События для Application Bar

Теперь неплохо бы заставить кнопки-значки реагировать на наши действия. Добавим событие **Click** и его обработчик в коде:

[XAML]

```
<shell:ApplicationBarIconButton IconUri="/icons/appbar.favs.rest.png"
   Text="Котята" Click="AppBarIconButton1_Click"/>
<shell:ApplicationBarIconButton IconUri="/icons/appbar.refresh.rest.png"
   Text="Кошки" Click="AppBarIconButton2_Click"/>
```

[C#]

```
private void AppBarIconButton1_Click(object sender, EventArgs e)
{
   PageTitle.Text = "котята";
}

private void AppBarIconButton2_Click(object sender, EventArgs e)
{
   PageTitle.Text = "кошки";
}
```

Кроме значков на Application Bar Вы можете видеть троеточие (...). Если коснуться в этом месте, то откроется дополнительная выдвижная панель с текстовым меню. Повторное касание троеточия закроет панель. Сейчас там мы видим команды **menuitem 1** и **menuitem 2**. Поступаем с ними аналогичным образом:

[XAML]

```
<shell:ApplicationBarMenuItem Text="Кто сказал мяу"
   Click="AppBarMenuItem1_Click"/>
<shell:ApplicationBarMenuItem Text="Котенок по имени Гав"
   Click="AppBarMenuItem2_Click"/>
```

[C#]

```
private void AppBarMenuItem1_Click(object sender, EventArgs e)
{
   MessageBox.Show("Наверное, котенок");
}

private void AppBarMenuItem2_Click(object sender, EventArgs e)
{
   MessageBox.Show("Странное имя для котенка");
}
```

В отличие от значков на панели, текстовых команд может быть больше, и при этом поддерживается прокрутка.

2.15.3. Управление прозрачностью Application Bar

Можно использовать свойство **Opacity** для изменения прозрачности панели. Если значение **Opacity** равно 1, то панель непрозрачна, если установить значение «0,5», то панель станет полупрозрачной (рис. 2.36).

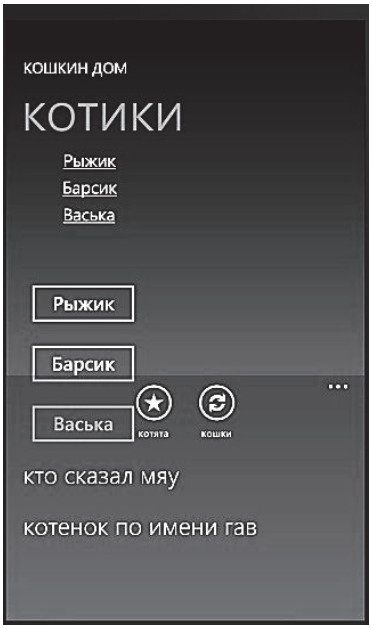


Рис. 2.36. Позрачный объект управления

2.15.4. Системная область

При запущенном приложении мы по-прежнему видим в верхней части экрана часы, уровень сигнала Wi-Fi, заряд батареи и другую системную информацию (на реальном устройстве, на эмуляторе ничего не отображается). Вы можете скрыть эту область, которая видима по умолчанию. Все, что нужно сделать – присвоить свойству **Visible** значение **false**:

```
shell:SystemTray.IsVisible="False"
```

2.15.5. Дополнительные сведения

Есть несколько значков панели приложения, которые устанавливаются вместе с Windows Phone Developer Tools. Вы можете найти эти значки в одном из следующих мест:

- На 32-битных операционных системах: C:\Program Files\Microsoft SDKs\Windows Phone\v7.0\Icons;
- На 64-битных операционных системах: C:\Program Files (x86)\Microsoft SDKs\Windows Phone\v7.0\Icons.

Если Вам необходимо создать собственный значок панели приложения, то он должен соответствовать следующим требованиям:

- размер 48 пикселей на 48 пикселей;
- должен быть белого цвета на прозрачном фоне;
- не включает в себя изображения круга, поскольку он отображается панелью приложения.

2.16. Launcher (задачи выполнения)

В Windows Phone 7 содержится целый ряд классов-задач, называемых Launcher (задачи выполнения) и Choosers (задачи выбора). Сегодня поговорим о задачах выполнения.

Задача выполнения – это особый механизм запуска задачи. Вы запускаете определенную задачу: послать письмо или SMS, позвонить, запустить медиаплеер, перейти на указанный URL и т.п. К сожалению, часть задач доступна только на реальном устройстве, а на эмуляторе показывается только приблизительный аналог. Обратите внимание, что мы только делаем необходимые приготовления (заполняем адрес, текст письма), а запускает задачу только пользователь. Нельзя запустить задачу без ведома владельца телефона.

Список задач выполнения:

- PhoneCallTask;
- SmsComposeTask;
- EmailComposeTask;
- WebBrowserTask;
- MediaPlayerLauncher;
- SearchTask;
- MarketplaceHubTask;
- MarketplaceReviewTask;
- MarketplaceDetailTask;
- MarketplaceSearchTask.

Для начала необходимо установить ссылку на пространство имен Microsoft.Phone.Tasks:

```
using Microsoft.Phone.Tasks;
```

Теперь мы можем выбрать любую доступную задачу. Принцип работы с задачами следующий: сначала мы устанавливаем необходимые свойства для задачи, а затем вызываем метод **Show()**.

1. PhoneCallTask – запуск диалога для звонка.

Команда запускает встроенный диалог операционной системы для осуществления звонка. Можно указать номер и отображаемое имя абонента. Звонок не осуществляется, пока пользователь не нажмёт кнопку «позвонить».

```
PhoneCallTask phoneCallTask = new PhoneCallTask();
phoneCallTask.PhoneNumber = "55555555577";
phoneCallTask.DisplayName = "Gaga";
phoneCallTask.Show();
```

2. SmsComposeTask – послание SMS-сообщения.

Команда запускает приложение для отправки сообщений. Можно определить получателя и тело сообщения, но пользователь должен сам нажать кнопку «Отправить».

```
SmsComposeTask smstask = new SmsComposeTask();
smstask.To = "5555555555";
smstask.Body = "Не забудь купить вискас для кота";
smstask.Show();
```

3. EmailComposeTask – послание e-mail.

На телефоне должна быть настроена учетная запись электронной почты, чтобы не получить сообщение об ошибке.

```
EmailComposeTask emailcomposer = new EmailComposeTask();
emailcomposer.To =
"<a href="mailto:mail@alexanderklimov.ru">mail@alexanderklimov.ru</a>";
emailcomposer.Subject = "Тема";
emailcomposer.Body = "Не забудь погладить кота";
emailcomposer.Show();
```

4. WebBrowserTask – запуск браузера по указанному адресу.

Мы можем также запустить браузер с указанным адресом (рис. 2.37). Например, мы можем перейти на знакомый Вам сайт следующим образом:

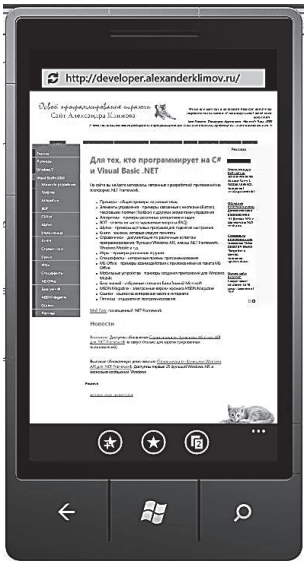


Рис. 2.37. Запуск браузера

```
WebBrowserTask browser = new WebBrowserTask();
browser.URL = "http://developer.alexanderklimov.ru";
browser.Show();
```

5. MediaPlayerLauncher – запуск встроенного плеера и проигрывание заданного файла.

```
MediaPlayerLauncher mediaPlayerLauncher = new
MediaPlayerLauncher();
mediaPlayerLauncher.Media = new Uri("MyVideo.wmv",
UriKind.Relative);
mediaPlayerLauncher.Location = MediaLocationType.Data;
mediaPlayerLauncher.Controls = MediaPlaybackControls.Pause | MediaPlaybackControls.Stop;
mediaPlayerLauncher.Show();
```

6. SearchTask – запуск поиска в Bing из приложения.

```
SearchTask searchTask = new SearchTask();
searchTask.SearchQuery = "Cats";
searchTask.Show();
```

Рис. 2.38. Выполнение задачи Marketplace на эмуляторе

7. MarketplaceHubTask – запуск Marketplace.

Можно определить категорию приложений, которая будет отображаться.

```
var marketplaceHubTask = new Microsoft.Phone.Tasks.MarketplaceHubTask();
marketplaceHubTask.ContentType = MarketplaceContentType.Music;
marketplaceHubTask.Show();
```

Примечание. Если у вас есть подключение к Интернету, то через эмулятор можно подключиться к Marketplace (рис. 2.38). При первой попытке появится сообщение об ошибке. Не обращайте внимания. Вернитесь назад и снова нажмите кнопку, выполняющую код. И Вы сможете бродить по разделам магазина. Жаль, скачать ничего не получится.

8. MarketplaceReviewTask – запуск Marketplace и отображение данных о текущем приложении.

```
MarketplaceReviewTask marketplaceReviewTask = new MarketplaceReviewTask();
marketplaceReviewTask.Show();
```

9. MarketplaceDetailTask – запуск Marketplace и отображение данных о заданном продукте.

```
MarketplaceDetailTask marketplaceDetailTask = new MarketplaceDetailTask();
marketplaceDetailTask.ContentIdentifier = "<ID>";
marketplaceDetailTask.ContentType = MarketplaceContentType.Applications;
marketplaceDetailTask.Show();
```

10. MarketplaceSearchTask – запускаем Marketplace с поисковым запросом.

```
MarketplaceSearchTask marketsearch = new MarketplaceSearchTask();
marketsearch.SearchTerms = "cat";
marketsearch.Show();
```

Для поиска песни:

```
MarketplaceSearchTask marketplaceSearchTask = new MarketplaceSearchTask();
marketplaceSearchTask.ContentType = MarketplaceContentType.Music;
```

```
marketplaceSearchTask.SearchTerms = "song title";
marketplaceSearchTask.Show();
```

2.17. Choosers (задачи выбора)

В предыдущем подразделе мы познакомились с задачами выполнения (Launcher). Настало время узнать о задачах выбора (Choosers). Задачи выбора также относятся к пространству имен **Microsoft.Phone.Tasks**. Основное их отличие от задач выполнения состоит в том, что задачи выбора возвращают какое-то значение.

Чтобы использовать задачи выбора в программе, не забывайте установить ссылку на пространство имен Microsoft.Phone.Tasks:

```
using Microsoft.Phone.Tasks;
```

Рассмотрим некоторые из них:

- **AddressChooserTask** – позволяет выбрать физический адрес из списка контактов для использования в приложении (Windows Phone 7.1 Mango).
- **CameraCaptureTask** – позволяет вызвать диалоговое окно работы с камерой и возвращает приложению сделанную фотографию.
- **EmailAddressChooserTask** – позволяет пользователю выбрать электронный адрес из списка контактов для использования в приложении.
- **PhoneNumberChooserTask** – позволяет пользователю выбрать телефонный номер из списка контактов для использования в приложении.
- **PhotoChooserTask** – позволяет пользователю выбрать фотографию на устройстве для использования в приложении.
- **SaveEmailAddressTask** – позволяет пользователю сохранить электронный адрес.
- **SavePhoneNumberTask** – запускает Контакты, что позволяет пользователю сохранить телефонный номер из Вашего приложения в новую или существующую запись.
- **SaveRingtoneTask** – позволяет сохранить MP3-файл из приложения в качестве рингтона (Windows Phone 7.1 Mango).

Каждая задача выбора имеет свои наборы свойств. После установки необходимых свойств для выбранной задачи нужно вызвать метод **Show()**. Кроме того, Вам необходимо создать обработчик события, который будет что-то делать с возвращаемыми данными.

1. AddressChooserTask – сохранение физического адреса.

Чтобы получить физический адрес пользователя из списка контактов, достаточно написать следующий код:

```
AddressChooserTask address;
void adress_Completed(object sender, AddressResult e)
```

```
{
   if (e.TaskResult == TaskResult.OK)
   {
      MessageBox.Show("Адрес " + e.DisplayName +": " + e.Address);
   }
}

private void buttonGetAdress_Click(object sender, RoutedEventArgs e)
{
   address = new AddressChooserTask();
   address.Completed += new EventHandler<AddressResult>(adress_Completed);
   address.Show();
}
```

2. CameraCaptureTask – съемка на камеру.

Для задачи показа сделанной фотографии в приложении, можно использовать следующий код:

```
CameraCaptureTask cameratask = new CameraCaptureTask();
cameratask.Completed += new EventHandler<PhotoResult>(cameratask_Completed);
cameratask.Show();

void cameratask_Completed(object sender, PhotoResult e)
{
   if (e.TaskResult == TaskResult.OK)
         {
            BitmapImage bmp = new BitmapImage();
            bmp.SetSource(e.ChosenPhoto);
            image1.Source = bmp;
         }
}
```

Когда произойдет метод **Show()**, отобразится стандартный интерфейс фотографирования, пользователь сделает фото и в Вашем приложении произойдёт событие, в обработчике которого данное фото можно получить. При отладке приложения в эмуляторе вместо изображения с камеры Вы увидите картинку, на которой черный блок перемещается по экрану. К сожаленю, эмулятор не поддерживает камеру, даже если на Вашем компьютере установлена веб-камера. В принципе, для отладки вариант с двигающим прямоугольником вполне подойдет, но всё же рекомендуется проверить работу приложения на реальном устройстве перед публикацией приложения в Marketplace.

3. EmailAddressChooserTask – выбор адреса электронной почты.

Если Вы захотите использовать в эмуляторе задачу выбора адреса электронной почты (рис. 2.39), то эмулятор предложит несколько «липовых» адресов из адресной книги:

```
EmailAddressChooserTask emailtask = new EmailAddressChooserTask();
emailtask.Completed += new EventHandler<EmailResult>(emailtask_Completed);
emailtask.Show();

void emailtask_Completed(object sender, EmailResult e)
{
  MessageBox.Show("Вы выбрали адрес: " + e.Email);
}
```

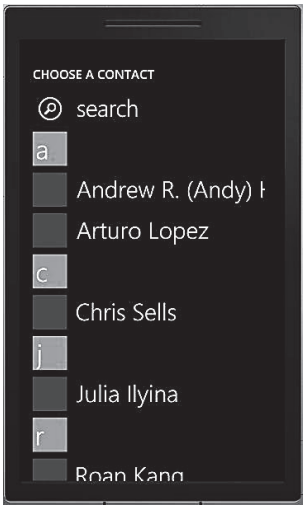


Рис. 2.39. Работа задачи электронной почты

4. PhoneNumberChooserTask – выбор телефонного номера.

Пример практически ничем не отличается от предыдущего.

```
PhoneNumberChooserTask phonetask = new PhoneNumberChooserTask();
phonetask.Completed += new EventHandler<PhoneNumberResult>(phonetask_Completed);
phonetask.Show();
void phonetask_Completed(object sender, PhoneNumberResult e)
{
  MessageBox.Show("Вы выбрали номер: " + e.PhoneNumber);
}
```

5. SavePhoneNumberTask – сохранение телефонного номера в контактах.

```
private void PhoneApplicationPage_Loaded(object sender, RoutedEventArgs e)
{  SavePhoneNumberTask savePhoneNumber = new SavePhoneNumberTask();
savePhoneNumber.Completed += new
                    EventHandler<TaskEventArgs>(savePhoneNumber_Completed);
  savePhoneNumber.PhoneNumber = "1987654320";
  savePhoneNumber.Show();

}

void savePhoneNumber_Completed(object sender, TaskEventArgs e)
{
  MessageBox.Show("Номер сохранён");
}
```

При вызове метода **SavePhoneNumber.Show()** на экране появляется страница «Контакты», позволяющая пользователю добавить заданный телефонный номер (или включить в уже существующий профиль).

Когда номер будет сохранен, пользователь должен нажать кнопку **Back**, чтобы вернуться в приложение. Сохранить телефонный номер без вызова страницы «Контакты» нельзя.

6. SaveEmailAddressTask – сохранение электронного адреса в контактах.

Пример схож с предыдущим методом **SavePhoneNumberTask**:

```
private void PhoneApplicationPage_Loaded(object sender, RoutedEventArgs e)
{  SaveEmailAddressTask saveEmail = new SaveEmailAddressTask();
   saveEmail.Completed += new
EventHandler<TaskEventArgs>(saveEmail_Completed);
   saveEmail.Email = "mail@alexanderklimov.ru";
   saveEmail.Show();

}

void saveEmail_Completed(object sender, TaskEventArgs e)
{
   if (e.TaskResult == TaskResult.OK)
    {
        MessageBox.Show("Электронный адрес сохранён");
    }
}
```

При вызове метода на экране появляется страница «Контакты», позволяющая пользователю добавить электронный адрес.

Когда адрес будет сохранен, пользователь должен нажать кнопку **Back**, чтобы вернуться в приложение. Сохранить адрес без вызова страницы «Контакты» нельзя.

7. SaveRingtoneTask – запуск Менеджера рингтонов.

Команда позволяет приложению запустить Менеджер рингтонов и сохранить музыкальный файл в списке рингтонов. Сохраненный рингтон можно использовать и в качестве основного звонка, и в контактах для определенного пользователя. Музыкальный файл должен соответствовать следующим параметрам:

- рингтон может быть только файлом MP3 или WMA;
- продолжительность музыки не должна превышать 40 с;
- файл не должен иметь цифровую защиту (DRM);
- размер файла не должен превышать 1 Мб.

В примере используется кнопка и текстовый блок

```
using Microsoft.Phone.Tasks;
public partial class MainPage : PhoneApplicationPage
{
   // Declare the SaveRingtoneTask object with page scope
```

```
        // so we can access it from event handlers.
        SaveRingtoneTask saveRingtoneChooser;

        // Constructor
        public MainPage()
        {
            InitializeComponent();
// Инициализация SaveRingtoneTask and assign the Completed handler in the page constructor.
            saveRingtoneChooser = new SaveRingtoneTask();
saveRingtoneChooser.Completed += new

        EventHandler<TaskEventArgs>(saveRingtoneChooser_Completed);
        }

// Пример, как в SaveRingtoneTask обрабатывается событие «нажатие кнопки».
        private void addButton_Click(object sender, RoutedEventArgs e)
        {
            string strMsg = string.Empty;

            try
            {
saveRingtoneChooser.Source = new Uri("appdata:/myTone.mp3");     // имя файла зашито в коде
                saveRingtoneChooser.DisplayName = "My custom ringtone";
                saveRingtoneChooser.Show();
            }
            catch (Exception ex)
            {
                strMsg = ex.Message;
            }

            statusTextBlock.Text = strMsg;
        }

        // The Completed event handler. No data is returned from this Chooser, but
        // the TaskResult field indicates whether the task was completed or cancelled.
        void saveRingtoneChooser_Completed(object sender, TaskEventArgs e)
        {
            if (e.TaskResult == TaskResult.OK)
            {
                statusTextBlock.Text = "Save completed.";
            }
            else if (e.TaskResult == TaskResult.Cancel)
            {
```

Рис. 2.40. Работа задачи PhotoChooserTask

```
                statusTextBlock.Text = "Save cancelled.";
            }
        }
    }
```

8. PhotoChooserTask – выбор картинки.

И, наконец, последний пример – выбор картинки (рис. 2.40).

```
    PhotoChooserTask phototask = new Photo-
ChooserTask();
            phototask.Completed += new
EventHandler<PhotoResult>(phototask_Completed);
            phototask.Show();

    void phototask_Completed(object sender, PhotoRe-
sult e)
    {
        BitmapImage bmp = new BitmapImage();
        bmp.SetSource(e.ChosenPhoto);
        image1.Source = bmp;
    }
```

2.18. Отладка

Отладка является необходимой частью разработки приложения, а возможно, самой важной частью. Всегда необходимо тщательно тестировать свою программу, чтобы не получить в свой адрес обидные слова.

Application.Current.Host.Settings.

Когда Вы запускаете приложение в эмуляторе, то сбоку можно наблюдать странные цифры (рис. 2.41). Это информация о характере работы Вашего приложения.

Рис. 2.41. Дополнительная цифровая информационная строка

Оказывается, если открыть файл App.xaml.cs, то можно там увидеть несколько блоков в операторе **if**:

```
// показывать основную графическую информацию в ходе отладки
if (System.Diagnostics.Debugger.IsAttached)
{
        // Display the current frame rate counters.
        Application.Current.Host.Settings.EnableFrameRateCounter = true;
        // Show the areas of the app that are being redrawn in each frame.
        //Application.Current.Host.Settings.EnableRedrawRegions = true;

        // Enable non-production analysis visualization mode,
        // which shows areas of a page that are being GPU accelerated with a colored
overlay.
        //Application.Current.Host.Settings.EnableCacheVisualization = true;
}
```

Обратите внимание, что по умолчанию работает только первый блок, а остальные два закомментированы. В этой секции кода можно включить некоторые инструменты отладки, а цифры, отображаемые в эмуляторе, показывают полезную информацию о работающем приложении. Например, там отображается количество кадров в секунду. Включение всех инструментов отладки обернуто в проверку свойства Debugger.IsAttached. Финальная версия программы не будет использовать инструменты отладки. Таким образом появляется удобная возможность тестировать различные ситуации при работе с приложением: разблокировка меню, тестирование ознакомительной версии и т.д. Иными словами, в режиме отладки Вы можете отключить ознакомительную версию и использовать программу на всю катушку, а готовый релиз будет работать с ограничениями. Вот краткое описание используемых инструментов.

1. EnableFrameRateCounter.

Напомним, что это единственная опция, включённая по умолчанию, которая позволяет мониторить число кадров в секунду, обновляемых в Вашем приложении. Помните, что для работы вполне хватит 24 кадра в секунду, и Вам не нужно гнаться за более высокими показателями.

Вот что означает каждая из цифр счётчика (рис. 2.42):

- **Render Thread FPS** – число кадров в секунду для потока рендеринга.
- **User Interface Thread FPS** – число кадров в секунду потока пользовательского интерфейса. В данном потоке кроме всего прочего выполняется связывание данных и анимация, не обрабатываемая потоком рендеринга.
- **Texture Memory Usage –** счётчик видеопамяти, используемой для хранения текстур.

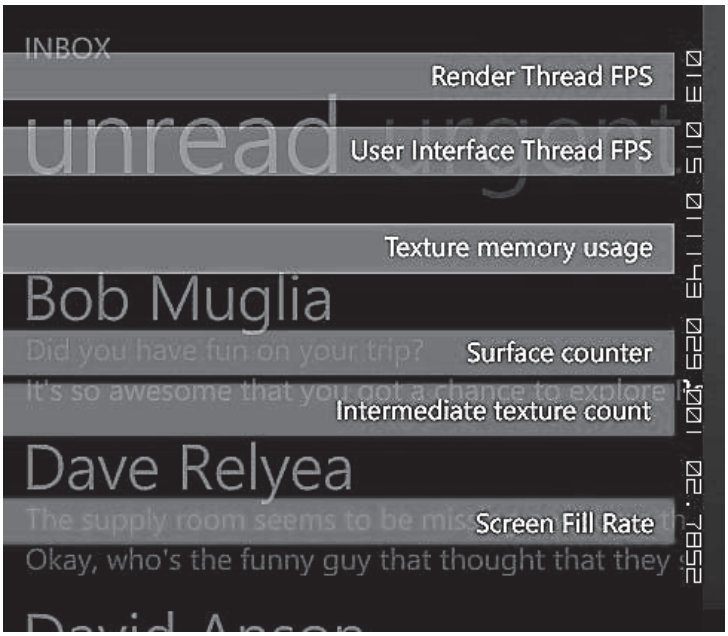


Рис. 2.42. Расшифровка значений дополнительной цифровой строки

- **Surface Counter** – число поверхностей, отправленных на графический ускоритель.
- **Intermediate Texture Count** – число промежуточных текстур.
- **Screen Fill Rate** – число полностью закрашенных экранов на каждый кадр.
- **EnableRedrawRegions** – после включения данной опции Вы сможете увидеть какие области Вашего приложения перерисовываются. Перерисовываемые области подсвечиваются синим цветом (рис. 2.43). Это бывает полезным, если частота обновления кадров очень низка и Вы не можете понять, почему это происходит.

Рис. 2.43. Подсвечивание перерисовываемых областей

- **EnableCacheVisualization** – другая опция, которая подсвечивает области в Вашем приложении. Речь идет об областях, которые используют аппаратное ускорение. Полезно при работе с анимацией и видео.

Численные значения параметров настройки представлены в табл. 2.2.

Таблица 2.2

Значения параметров опции EnableFrameRateCounter

Счетчик	Минимально допустимо	Отлично	Теоретический максимум
Render Thread	30 fps	60 fps	120 fps
UI Thread	15 fps	>15 fps	120 fps
Screen Fill	Rate 1.0	<= 2.0 N/A	

2.19. Ввод информации при помощи клавиатуры

Для ввода информации используется реальная аппаратная или виртуальная клавиатура на сенсорном экране. Проблема заключается в размерах. Все мы знаем, что физические размеры устройств достаточно малы и разместить на экране таких аппаратов несколько десятков кнопок достаточно сложно, если оставлять кнопки большого размера. Поэтому зачастую кнопки остаются небольшими и ввод информации не такой удобный, как на настольных компьютерах или ноутбуках. С аппаратной выдвижной клавиатурой работать более удобно, но упомянутая выше проблема сохраняется – размеры кнопок невелики. Поэтому ввод информации при использовании мобильных устройств – одна из наиболее сложных проблем в удобстве пользования этими устройствами.

Большинство телефонов сейчас не снабжены аппаратной клавиатурой, пользователи используют вместо нее виртуальную клавиатуру. Рассмотрим приемы работы с подобной клавиатурой на примерах. Также поговорим об элементе PasswordBox.

2.19.1. Подключение настольной клавиатуры к эмулятору

В эмуляторе при наборе текста теперь обычно нельзя вводить символы используя свою рабочую клавиатуру на настольном компьютере, как это было раньше в старых эмуляторах под Windows Mobile. Но эта проблема решается очень просто. Достаточно нажать на клавишу Pause/Break на своей клавиатуре, чтобы переключиться на другой режим. Чтобы снова переключиться обратно на виртуальную клавиатуру, нажмите эту клавишу еще раз. Этот прием нужно использовать только в том случае, когда нужно быстро набрать какой-нибудь текст для текстового поля. Нужно понимать, что подобный способ не имеет ничего общего с реальным вводом инфор-

мации на устройстве и Вы не увидите виртуальную клавиатуру. В дальнейшем все примеры используют виртуальную клавиатуру.

2.19.2. InputScope

Каждый раз, когда фокус попадает на TextBox или PasswordText, то появляется возможность использовать **InputScope** (рис. 2.44) – стандартную клавиатуру и клавиатуру для набора телефонных номеров.

Рис. 2.44. Клавиатура InputScope

Если рассмотреть наиболее типичные для пользователя сценарии, то можно понять, что в большинстве случаев они повторяются. Так, например, клавиатура обычно используется для ввода номера телефона, имени контакта, ввода текстовой информации (например, сообщения электронной почты или SMS), ввода адреса электронной почты, адреса веб-ресурса и т.д.

Стоит заметить, что виртуальная клавиатура имеет более гибкие настройки по сравнению с аппаратной клавиатурой. Вы можете предложить пользователю тот вид клавиатуры, который наиболее подходит для решения текущей задачи. Например, если пользователю нужно набрать телефонный номер, то разумно предложить ему клавитуру с цифрами или другими используемыми символами типа * или #. Например, мы можем вывести на экран только цифровую клавиатуру и клавиатуру со смайликами. Вот как будет выглядеть небольшой пример для двух текстовых полей в XAML:

```
<TextBox Height="100" />
<TextBox InputScope="TelephoneNumber" Height="100" />
```

Когда фокус получит первое текстовое поле, то отобразится стандартная клавиатура по умолчанию. Во втором случае отобразится клавиатура для набора телефонных номеров. Вот наиболее используемые виды клавиатур (рис. 2.45).

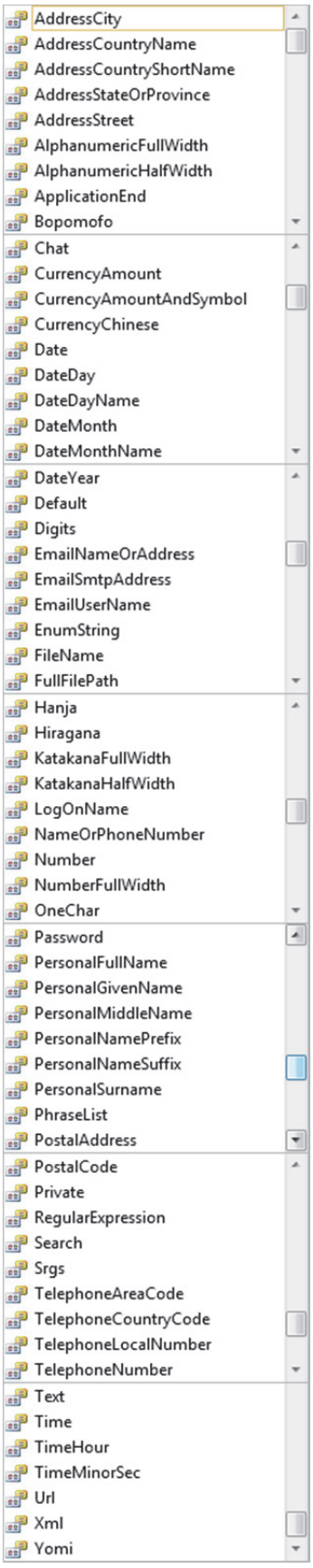


Рис. 2.45. Полный перечень клавиатур

1. Клавиатура URL.

У данной клавиатуры есть кнопка **.com**, позволяющая завершить ввод URL-адреса. Если удерживать кнопку несколько секунд, то появятся также варианты **.net, .org, .edu**. Локализованные версии телефонов могут иметь дополнительные варианты, например, **.ru** (рис. 2.46).

Рис. 2.46. Клавиатура URL

2. Клавиатура Number.

Это обычная цифровая клавиатура. Кроме цифр используются различные математические символы (+, -, %) и другие знаки (рис. 2.47).

Рис. 2.47. Клавиатура Number

3. Клавиатура Text.

Особенность у этой клавиатуры – наличие кнопки со смайликом. Нажав на кнопку, Вы увидите большой список доступных смайликов (рис. 2.48). Подобная клавиатура пригодится в чатах и в программах мгновенных сообщений.

Рис. 2.48. Клавиатура Text

4. Клавиатура TelephoneNumber.

Это клавиатура для набора телефонных номеров (рис. 2.49).

5. Клавиатура EmailNameOrAddress.

При вводе электронных адресов часто используются точки, собачки (@) и окончания доменов, типа .com или .net. Поэтому данные символы представлены в данной клавиатуре (рис. 2.50).

Рис. 2.49. Клавиатура TelephoneNumber

Рис. 2.50. Клавиатура EmailNameOrAddress

Использование метода Intellisense.

Вариантов клавиатур слишком много, чтобы их помнить. Поэтому можно пойти на небольшую хитрость. Когда мы вводим просто InputScope="" в редакторе кода, то всплывающая подсказка (рис. 2.51) не появляется. Поэтому чуть изменим код:

```
<TextBox Height="75">
        <TextBox.InputScope>
                <InputScope>
                <InputScopeName NameValue="Bopomofo" />
                </InputScope>
        </TextBox.InputScope>
</TextBox>
```

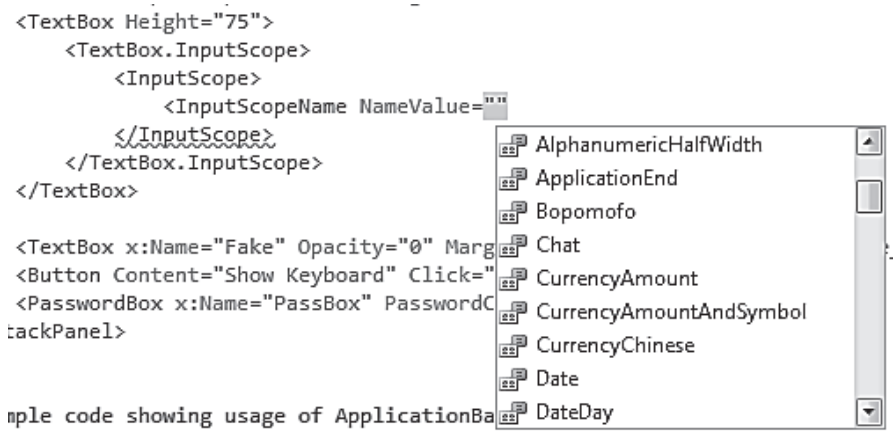


Рис. 2.51. Вид подсказки

Если вам интересно, Bopomofo – официальная фонетическая система ввода для Китайского языка. На рис. 2.45 представлен полный список значений для InputScopeName.

Учтите, что большинство клавиатур не работает. Возможно, поддержка экзотических клавиатур будет осуществляться производителями телефонов.

2.19.3. Программный просмотр всех клавиатур

Для того чтобы увидеть каждый случай, можно, конечно, изменять значение кода клавиатуры и перезапускать приложение; однако мы воспользуемся другим способом. Для начала с помощью Reflection получим всевозможные значения этого перечисления. После этого для каждого значения перечисления создадим новое поле ввода и определим для него соответствующее значение свойства InputScope. Разметка формы при этом изменится следующим образом:

```
<Grid x:Name="ContentPanel" Grid.Row="1"
Margin="12,0,12,0">
<ScrollViewer BorderThickness="0" Height="280"
                                        VerticalAlignment="Top">
      <StackPanel x:Name="Elements">
        <StackPanel.Resources>
          <Style TargetType="TextBox">
            <Setter Property="Margin" Value="10"/>
          </Style>
        </StackPanel.Resources>
      </StackPanel>
    </ScrollViewer>
</Grid>
```

При этом в момент создания и загрузки формы создадим следующий код. Здесь мы получаем все значения перечисления и для каждого значения создадим новое поле ввода и помещаем его на форму. При этом для каждого поля ввода задается текущее значение InputScope.

```
using System.Reflection;
foreach (var inputName in typeof(InputScopeNameValue).
    GetFields(BindingFlags.Public | BindingFlags.Static).Select(t => t.Name))
{
InputScopeNameValue val = (InputScopeNameVal-
ue)(Enum.Parse(typeof(InputScopeNameValue), inputName, true));
  if (val > 0)
  {
    Elements.Children.Add(new TextBox()
    { Text = inputName,
      InputScope = new System.Windows.Input.InputScope
      {Names =
{
  new InputScopeName()
    {
      NameValue = val
    }
}
      }
    });
  }
}
```

Рис. 2.52. Вид экрана эмулятора

При запуске приложения (рис. 2.52) на форме будет созданы элементы управления. При установлении фокуса на каждое поле можно увидеть различные варианты виртуальной клавиатуры.

2.19.4. Программный запуск клавиатуры

Клавиатура показывается только при работе с текстовыми полями. Если по каким-то причинам Вы хотите показать клавиатуру не работая с текстовым полем (хороший пример – игра «Виселица»), то можно сделать следующее:

1. Разместите кнопку в приложении.
2. Разместите TextBox на странице, но сделайте ее невидимой для пользователя (можете сделать ее полностью прозрачной или вывести за пределы страницы). В этом случае пользователь не видит текстовое поле, которое тем не менее присутствует в приложении.

3. Установите обработчик события для кнопки, который установит **Focus()** для невидимого текстового поля.

Способ достаточно специфичный, но, возможно, вам пригодится.

2.19.5. PasswordBox

PasswordBox похож на TextBox, но имеет некоторые особенности. Во-первых, вводимый символ виден на экране, а затем заменяется на кружочек. Во-вторых, Вы можете заменить используемый по умолчанию кружочек на звездочку или другой символ при помощи свойства **PasswordChar**:

```
<PasswordBox x:Name="PassBox" PasswordChar="?" />
```

2.20. Приложение для телефона на XNA

Далее у нас по плану приложение на XNA, отображающее небольшое приветствие в центре экрана. Тогда как в приложениях на Silverlight текст обычно превалирует, в видеоиграх его встретишь не часто. В играх роль текста сведена к описанию правил или отображению счета. Поэтому сама концепция приложения «Здравствуй, мир!» не вполне вписывается в общую идеологию программирования на XNA.

В XNA даже нет встроенных шрифтов, и приложение на XNA, выполняющееся на телефоне, не может использовать те же встроенные шрифты телефона, что и программы на Silverlight, как это можно было бы предположить. Silverlight применяет векторные шрифты TrueType, а XNA ничего не знает о таких экзотических концепциях. Для XNA все, включая шрифты, является растровыми изображениями.

Если в приложении на XNA требуется использовать определенный шрифт, он должен быть встроен в исполняемый файл как коллекция растровых изображений для каждого символа. XNA Game Studio (которая интегрирована в Visual Studio) очень упрощает сам процесс встраивания шрифта, но тут возникают некоторые серьезные правовые проблемы. Вы можете легально распространять приложение на XNA, использующее тот или иной шрифт, только при условии, если имеете право на распространение этого встроенного шрифта, а это невозможно для большинства шрифтов, поставляемых самой ОС Windows или приложениями Windows.

Чтобы помочь в решении этого правового затруднения, Microsoft предоставляет лицензию на использование шрифтов Ascender Corporation именно в целях их применения в приложениях на XNA. Вот эти шрифты:

- Kootenay Lindsey;
- Miramonte Bold Pescadero Bold;
- Pericles Segoe UI Mono;

- Pericles Light Segoe UI Mono Bold.

Обратите внимание, что в шрифтах Pericles в качестве строчных букв используются уменьшенные заглавные, поэтому, вероятно, они подойдут только для заголовков.

В Visual Studio в меню File выберите New и Project. В левой части диалогового окна выберите Visual C# и XNA Game Studio 4.0. В середине выберите Windows Phone Game (4.0). Задайте месторасположение и имя проекта XnaHelloPhone.

Visual Studio создает два проекта, один для логики приложения и другой для его содержимого. Приложения на XNA обычно включают большой объем содержимого, которым преимущественно являются растровые изображения, трехмерные модели и шрифты.

Чтобы добавить шрифт в это приложение, щелкните правой кнопкой мыши проект, созданный для содержимого (он обозначен «XnaHelloPhoneContent (Content)»), и во всплывающем меню выберите Add (Добавить) и New Item (Новый элемент). Выберите Sprite Font, оставьте имя файла как есть **SpriteFont1.spritefont** и щелкните Add.

Слово «sprite» (в переводе на русский означает «эльф») широко распространено в игровых приложениях и обычно обозначает небольшое растровое изображение, которое может очень быстро перемещаться. В XNA даже шрифты являются спрайтами.

SpriteFont1.spritefont появится в списке файлов каталога Content, и Вы можете редактировать изобилующий комментариями XML-файл, описывающий шрифт.

Проект XNA: XnaHelloPhone Файл: SpriteFont1.spritefont (полностью за исключением комментариев)

```
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
<Asset Type="Graphics:FontDescription">
                <FontName>Segoe UI Mono</FontName>
                <Size>14</Size>
                <Spacing>0</Spacing>
                <UseKerning>true</UseKerning>
                <Style>Regular</Style>
                <CharacterRegions>
                <CharacterRegion>
                <Start>&#32;</Start>
                <End>&#126;</End>
                </CharacterRegion>
                </CharacterRegions>
        </Asset>
</XnaContent>
```

Между тегами FontName указан шрифт Segoe UI Mono, но его можно заменить любым другим шрифтом из приведенных ранее. Если желаете использовать Pericles Light, то укажите его полное имя, но для Miramonte Bold, Pescadero Bold или Segoe UI Mono Bold необходимо написать просто Miramonte, Pescadero или Segoe UI Mono и ввести Bold (Полужирный) между тегами Style (Стиль). Bold может использоваться и для других шрифтов, но для них он будет синтезирован, тогда как для Miramonte, Pescadero или Segoe UI Mono будет использоваться специально разработанный полужирный шрифт.

Теги Size (Размер) обозначают размер шрифта в пунктах. В XNA, как и в Silverlight, координаты и размеры задаются в пикселах, но в XNA за основу при преобразовании между пикселами и пунктами взято разрешение 96 DPI. Для XNA-приложения 14 шрифт равен 2/3 пикселам. Это очень близко к шрифту размером 15 пт, что соответствует значению FontSize, равному 20 пикселов в Silverlight для Windows Phone.

В разделе CharacterRegions (Диапазоны символов) файла указываются диапазоны шестнадцатеричных кодировок Unicode. По умолчанию используется диапазон от 0x32 до 0x126, что включает все обычные символы набора символов ASCII.

SpriteFont1.spritefont не является достаточно описательным именем. Предлагаем назвать файл так, чтобы было понятно, о каком шрифте идет речь. Если сохраняются стандартные настройки шрифта, можно переименовать файл в Segoe14.spritefont. Если взглянуть на свойства этого файла (щелкните правой кнопкой мыши имя файла и выберите Properties), то можно увидеть, что значением Asset Name (Имя ресурса) является имя файла без расширения: Segoe14. Значение Asset Name используется для ссылки на шрифт в приложении. Если хотите запутать себя, можете изменить Asset Name и задать ему значение, отличное от имени файла.

Изначально проект XNAHelloPhone включает два C#-файла: Program.cs и Game1.cs. Первый очень простой и, как выясняется, не имеет отношения к играм для Windows Phone 7! Директива препроцессора активирует класс Program (Программа), только если определен символ WINDOWS или XBOX. При компиляции программ для Windows Phone вместо них задается символ WINDOWS_PHONE.

Чаще всего при создании небольших игр основная часть времени уходит на файл Game1.cs. Класс Game1 наследуется от Game (Игра). Первоначально в нем определены два поля: graphics (графические элементы) и spriteBatch (Пакет спрайтов). К этим двум полям добавим еще три:

Проект XNA: XnaHelloPhone Файл: Game1.cs (фрагмент, демонстрирующий поля)

```
namespace XnaHelloPhone
{ public class Game1 : Microsoft.Xna.Framework.Game
```

```
{ GraphicsDeviceManager graphics;
 SpriteBatch spriteBatch;
 string text = "Hello, Windows Phone 7!";
SpriteFont segoe14; Vector2 textPosition;
…
} }
```

В этих трех новых полях просто указан текст для отображения, используемый для этого шрифт и месторасположение текста на экране. Координаты задаются в пикселах относительно верхнего левого угла экрана. Структура Vector2 имеет два поля: X и Y типа float (число с плавающей точкой). В целях обеспечения лучшей производительности в XNA все значения с плавающей точкой берутся с одинарной точностью (В Silverlight – с двойной точностью). Структура Vector2 часто используется для задания точек, размеров и даже векторов в двухмерном пространстве.

При запуске игры на телефоне создается экземпляр класса Game1 и выполняется конструктор Game1. Рассмотрим стандартный код:

Проект XNA: XnaHelloPhone Файл: Game1.cs (фрагмент)

```
public Game1()
{ graphics = new GraphicsDeviceManager(this);
Content.RootDirectory = "Content"; //По умолчанию частота кадров для Windows Phone составляет 30 кадр/с.
TargetElapsedTime = TimeSpan.FromTicks(333333); }
```

Первое выражение обеспечивает инициализацию поля graphics. Во втором выражении для Game определяется свойство Content (Содержимое) типа ContentManager (Диспетчер содержимого), и RootDirectory (Корневой каталог) является свойством этого класса. Значение «Content» этого свойства соответствует папке Content, в которой хранится шрифт Segoe размером 14 пт. В третьем выражении задается время игрового цикла программы, что управляет частотой обновления изображения. Экраны устройств Windows Phone 7 обновляются с частотой 30 кадров в секунду.

Когда экземпляр Game1 создан, вызывается его метод Run (Выполнить), и базовый класс Game инициирует процесс запуска игры. Один из первых шагов – вызов метода Initialize (Инициализировать), который может быть перегружен в производных от Game классах. XNA Game Studio автоматически формирует скелетный метод, в который предлагается ничего не добавлять:

Проект XNA: XnaHelloPhone Файл: Game1.cs (фрагмент)

```
protected override void Initialize()
{
        base.Initialize();
}
```

В методе Initialize шрифт или любое другое содержимое не должно загружаться. Это происходит несколько позже, когда базовый класс вызывает метод LoadContent (Загрузить содержимое).

Проект XNA: XnaHelloPhone Файл: Game1.cs (фрагмент)

```
protected override void LoadContent()
{
spriteBatch = new SpriteBatch(GraphicsDevice);
segoe14 = this.Content.Load<SpriteFont>("Segoe14");
Vector2 textSize = segoe14.MeasureString(text);
Viewport viewport = this.GraphicsDevice.Viewport;
textPosition = new Vector2((viewport.Width - textSize.X) / 2,
 (viewport.Height - textSize.Y) / 2);
}
```

Первое выражение данного метода формируется автоматически. Вскоре мы увидим, как этот объект spriteBatch используется для вывода спрайтов на экран.

Все остальные выражения добавляет программист. Как можно заметить, перед всеми именами свойств, таких как Content и GraphicsDevice (Графическое устройство), ставится ключевое слово this, чтобы напомнить, что это свойства, а не статические классы. Как уже говорилось, свойство Content типа ContentManager. Универсальный метод Load (Загрузить) обеспечивает загрузку содержимого в приложение, в данном случае это содержимое типа SpriteFont. Имя, указанное в кавычках, соответствует Asset Name, указанному в свойствах содержимого. Это выражение обеспечивает сохранение результата в поле segoe14 типа SpriteFont.

В XNA спрайты (включая текстовые строки) обычно позиционируются через задание координат в пикселах верхнего левого угла спрайта относительно верхнего левого угла экрана. Для расчета этих координат необходимо знать и размер экрана, и размер текста при отображении его конкретным шрифтом.

У класса SpriteFont есть чрезвычайно удобный метод MeasureString (Измерить строку), возвращающий объект Vector2 с размером конкретной текстовой строки в пикселах (для шрифта Segoe UI Mono размером 14 пт, высота которого эквивалентна 18-2/3 пикселам, метод MeasureString возвратит высоту 28 пикселов).

Как правило, для получения размера экрана в приложении на XNA используется свойство Viewport (Окно просмотра) класса GraphicsDevice. Оно доступно через свойство GraphicsDevice класса Game и предоставляет свойства Width (Ширина) и Height (Высота).

После этого довольно просто вычислить textPosition (Положение текста) – координаты точки относительно верхнего левого угла окна просмотра, в которой будет располагаться верхний левый угол текстовой строки.

На этом этап инициализации программы завершается и начинается фактическое действие. Приложение входит в игровой цикл. Синхронно с обновлением экрана, которое происходит с частотой 30 кадров в секунду, в приложении вызываются два метода: Update (Обновить) и за ним Draw (Рисовать). Снова и снова: Update, Draw, Update, Draw, Update, Draw… (На самом деле, все несколько сложнее; методу Update необходимо 1/30 секунды для выполнения).

Метод Draw обеспечивает отрисовку образов на экране. И это все, что он может делать. Все подготовительные вычисления для отрисовки должны осуществляться в методе Update. Метод Update подготавливает программу к выполнению метода Draw. Очень часто приложения на XNA реализуют перемещение спрайтов по экрану на основании пользовательского ввода. Для телефона пользовательский ввод осуществляется преимущественно посредством сенсорного ввода. Вся обработка пользовательского ввода также должна происходить во время выполнения метода Update.

Методы Update и Draw должны быть написаны так, чтобы они выполнялись максимально быстро. Однако здесь также имеются некоторые очень важные моменты.

Следует избегать включения в методы Update и Draw кода, выполняющего рутинные операции по распределению памяти из локальной кучи приложения. В определенный момент времени сборщик мусора .NET захочет вернуть часть этой памяти, и пока он будет выполнять свою работу, игра может немного притормаживать.

Скорее всего, в методах Draw не будет возникать никаких проблем. Обычно все неприятности кроются в методе Update. Избегайте применения выражений new для классов. Это всегда приводит к распределению памяти. Однако нет ничего страшного в создании экземпляров структур, потому что эти экземпляры хранятся в стеке, а не в куче (XNA использует структуры, а не классы для многих типов объектов, необходимых в Update). Но распределение памяти из кучи также может происходить и без явных выражений new. Например, конкатенация двух строк приводит к созданию новой строки в куче. Для выполнения каких-либо операций со строками в Update следует использовать StringBuilder (Построитель строк). Очень удобно, что XNA предоставляет для отображения текста методы, использующие объекты StringBuilder.

Но в нашем приложении XnaHelloPhone метод Update абсолютно тривиальный. Отображаемый текст зафиксирован в одной единственной точке. Все необходимые вычисления уже выполнены в методе LoadContent. Поэтому оставляем метод Update без изменений, просто в том виде, в каком он был изначально создан XNA Game Studio:

Проект XNA: XnaHelloPhone Файл: Game1.cs (фрагмент)

```
protected override void Update(GameTime gameTime)
```

```
{
if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                                        this.Exit(); base.Update(gameTime);
}
```

В формируемом по умолчанию коде для проверки события нажатия кнопки Back используется статический класс GamePad (Игровой планшет). Это событие является сигналом к выходу из игры.

И, наконец, метод Draw. Его автоматически созданная версия просто закрашивает фон голубым цветом:

Проект XNA: XnaHelloPhone Файл: Game1.cs (фрагмент)

```
protected override void Draw(GameTime gameTime)
{  GraphicsDevice.Clear(Color.CornflowerBlue);
   base.Draw(gameTime);
}
```

Васильковый цвет (CornflowerBlue) приобрел культовый статус в сообществе разработчиков на XNA. При работе над программой на XNA увидеть голубой экран очень утешительно, поскольку это означает, что программа, по крайней мере, дошла до метода Draw. Но в целях энергосбережения при использовании ОСИД-экранов желательно применять более темные фоны. Предлагается компромиссный вариант, в котором фон темно-синий. Как и Silverlight, XNA поддерживает 140 цветов, которые уже считаются стандартными. Выводимый текст будет белого цвета:

Проект XNA: XnaHelloPhone Файл: Game1.cs (фрагмент)

```
protected override void Draw(GameTime gameTime)
{
GraphicsDevice.Clear(Color.Navy);
spriteBatch.Begin();
spriteBatch.DrawString(segoe14, text, textPosition, Color.White);
spriteBatch.End();
base.Draw(gameTime);
}
```

Спрайты выводятся на экран пакетами в составе объекта SpriteBatch, который был создан во время вызова метода LoadContent. Между вызовами Begin (Начало) и End (Конец) может осуществляться множество вызовов метода DrawString (Отрисовать строку) для отрисовки текста и Draw для отрисовки растровых изображений. Вызываться могут только эти методы. В данном конкретном вызове DrawString указан шрифт, выводимый текст, местоположение верхнего левого угла текста относительно верхнего левого угла экрана и цвет. И вот, что мы получаем (рис. 2.53).

Рис. 2.53. Работа программы в эмуляторе

Рис. 2.54. Второй вариант работы программы в эмуляторе

Вот это любопытно! По умолчанию программы на Silverlight отображаются в портретном режиме, а программы на XNA – в альбомном. Давайте повернем телефон или эмулятор (рис. 2.54).

Намного лучше!

Но тут возникает вопрос: всегда ли приложения на Silverlight выполняются в портретном режиме, а приложения на XNA – в альбомном?

2.20.1. Ориентация в приложении на XNA

По умолчанию в приложениях на XNA для Windows Phone используется альбомная ориентация, возможно, для обеспечения совместимости с другими экранами, используемыми для игр. Поддерживается альбомная ориентация во всех направлениях, так что при переворачивании устройства как на левый, так и на правый бок, ориентация изображения на экране будет меняться соответственно. Если Вы предпочитаете игры с портретным расположением, изменить эту настройку не составляет труда. Добавим в конструктор класса Game1 приложения XnaHelloPhone следующие выражения:

```
graphics.PreferredBackBufferWidth = 320;
graphics.PreferredBackBufferHeight = 480;
```

Задний буфер – это область, в которой XNA создает графические элементы, выводимые на экран методом Draw. И размером, и пропорциями

этого буфера можно управлять. Поскольку заданная здесь ширина буфера меньше его высоты, XNA предполагает, что изображение требуется выводить в портретном режиме.

Посмотрите на это (рис. 2.55). Соотношение размеров заднего буфера отличается от пропорций экрана устройства Windows Phone 7, поэтому изображение выводится с черными полосами вверху и внизу экрана! Текст имеет тот же размер в пикселах, но выглядит больше, потому что разрешение экрана уменьшилось.

Рис. 2.55. Изменение пропорций экрана

Если Вы не являетесь большим поклонником такой зернистости изображения, вызывающей ностальгические воспоминания, то подумайте о применении меньшего заднего буфера, если игре не требуется такое высокое разрешение, какое предоставляет экран телефона. Это обеспечит повышение производительности и снизит энергопотребление. Размер заднего буфера может быть любым в диапазоне от 240×240 до 480×800 (для портретного режима) или 800×480 (для альбомного). XNA использует соотношение размеров для определения используемого режима отображения.

Задание необходимого заднего буфера является замечательным способом ориентировать приложение на экран определенного размера в коде, но также обеспечивает возможности применения устройств с другими размерами экранов, которые могут появиться в будущем.

По умолчанию размер заднего буфера равен 800×480, но его фактический размер на экране несколько меньше, поскольку требуется обеспечить место для панели задач. Чтобы избавиться от панели задач (и досадить пользователям, которые всегда хотят знать, который сейчас час), в конструкторе Game1 можно задать

```
graphics.IsFullScreen = true;
```

Кроме того, можно сделать так, чтобы игры на XNA реагировали на изменение ориентации экрана, но для этого, конечно же, придется немного изменить их структуру. Самый простой тип реструктуризации для обеспечения учета изменения ориентации продемонстрирован в проекте XnaOrientableHelloPhone. Его поля теперь включают переменную textSize (Размер текста):

Проект XNA: XnaOrientableHelloPhone Файл: Game1.cs (фрагмент, демонстрирующий поля)

```
public class Game1 : Microsoft.Xna.Framework.Game
{
```

```
GraphicsDeviceManager graphics;
SpriteBatch spriteBatch;
string text = "Hello, Windows Phone 7!";
SpriteFont segoe14;
Vector2 textSize;
Vector2 textPosition;
…
}
```

Конструктор Game1 включает выражение, определяющее свойство SupportedOrientations поля graphics:

Проект XNA: XnaOrientableHelloPhone Файл: Game1.cs (фрагмент)

```
public Game1()
{
graphics = new GraphicsDeviceManager(this);
Content.RootDirectory = "Content";
 // Делаем возможным отображение и в портретном режиме
graphics.SupportedOrientations =     DisplayOrientation.Portrait |
                                     DisplayOrientation.LandscapeLeft |
                                     DisplayOrientation.LandscapeRight;
// По умолчанию для Windows Phone частота кадров составляет 30 кадр/с.
 TargetElapsedTime = TimeSpan.FromTicks(333333);
}
```

Применяя SupportedOrientation, мы можем ограничить поддерживаемые телефоном режимы отображения. Это выражение, обеспечивающее поддержку и портретного, и альбомного режимов отображения, выглядит простым, но здесь имеются некоторые побочные эффекты. При изменении ориентации происходит сброс графического устройства (что приводит к формированию некоторых событий), и размеры заднего буфера изменяются. Можно подписаться на событие OrientationChanged класса GameWindow (Игровое окно), которое доступно через свойство Window (Окно), либо проверять свойство CurrentOrientation (Текущая ориентация) объекта GameWindow.

Предлагается несколько иной подход. Рассмотрим новый метод LoadContent, который, как можно будет заметить, принимает размер текста и сохраняет его как поле, но не получает размеров окна просмотра.

Проект XNA: XnaOrientableHelloPhone Файл: Game1.cs (фрагмент)

```
protected override void LoadContent()
{
spriteBatch = new SpriteBatch(GraphicsDevice);
segoe14 = this.Content.Load<SpriteFont>("Segoe14");
```

```
textSize = segoe14.MeasureString(text);
}
```

Параметры окна просмотра можно получить в ходе выполнения метода Update, поскольку размеры окна просмотра отражают ориентацию экрана.

Проект XNA: XnaOrientableHelloPhone Файл: Game1.cs (фрагмент)

```
protected override void Update(GameTime gameTime)
{
//Обеспечивает возможность выхода из игры
if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
 this.Exit();
Viewport viewport = this.GraphicsDevice.Viewport;
 textPosition = new Vector2((viewport.Width - textSize.X) / 2,
                                                    (viewport.Height -
textSize.Y) / 2);
base.Update(gameTime);
}
```

Какой бы ни была текущая ориентация, метод Update вычисляет положение текста. Метод Draw аналогичен тем, что были представлены ранее.

Проект XNA: XnaOrientableHelloPhone Файл: Game1.cs (фрагмент)

```
protected override void Draw(GameTime gameTime)
{
GraphicsDevice.Clear(Color.Navy);
spriteBatch.Begin();
spriteBatch.DrawString(segoe14, text, textPosition, Color.White);
 spriteBatch.End();
base.Draw(gameTime);
}
```

Теперь телефон или эмулятор можно переворачивать в разных направлениях, и изображение на экране будет менять ориентацию соответственно.

Если требуется получить размер экрана телефона независимо от заднего буфера или ориентации (но с учетом панели задач), сделать это можно с помощью свойства ClientBounds (Границы клиентской области) класса GameWindow, обратиться к которому можно из свойства Window класса Game:

```
Rectangle clientBounds = this.Window.ClientBounds;
```

2.20.2. Простые часы (цифровые)

В предыдущем подразделе были рассмотрены два события Silverlight (SizeChanged и OrientationChanged), но использовались они по-разному. Событие SizeChanged ассоциировалось с обработчиком события в XAML, а для события OrientationChanged переопределился эквивалентный метод OnOrientationChanged.

Конечно, обработчики этих событий могут быть определены полностью в коде. Одним очень удобным для приложений на Silverlight классом является DispatcherTimer (Таймер-диспетчер), который периодически обращается к приложению посредством события Tick (Тик) и побуждает его выполнить какую-то работу. Например, таймер используется для приложения, моделирующего часы.

Сетка для содержимого проекта SilverlightSimpleClock включает только расположенный по центру TextBlock:

Проект Silverlight: SilverlightSimpleClock Файл: MainPage.xaml (фрагмент)

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
 <TextBlock Name="txtblk"
                                HorizontalAlignment="Center"
                                VerticalAlignment="Center" />
</Grid>
```

Рассмотрим файл выделенного кода полностью. Обратите внимание на директиву using для пространства имен System.Windows.Threading, которое не используется по умолчанию. Это пространство имен, в котором находится DispatcherTimer.

Проект Silverlight: SilverlightSimpleClock Файл: MainPage.xaml.cs

```
using System;
using System.Windows.Threading;
using Microsoft.Phone.Controls;
namespace SilverlightSimpleClock
{
public partial class MainPage : PhoneApplicationPage
{
  public MainPage()
{ InitializeComponent();
                    DispatcherTimer tmr = new DispatcherTimer();
                    tmr.Interval = TimeSpan.FromSeconds(1);
                    tmr.Tick += OnTimerTick;
                   tmr.Start();
}
```

```
void OnTimerTick(object sender, EventArgs args)
{ txtblk.Text = DateTime.Now.ToString();
}
}
}
```

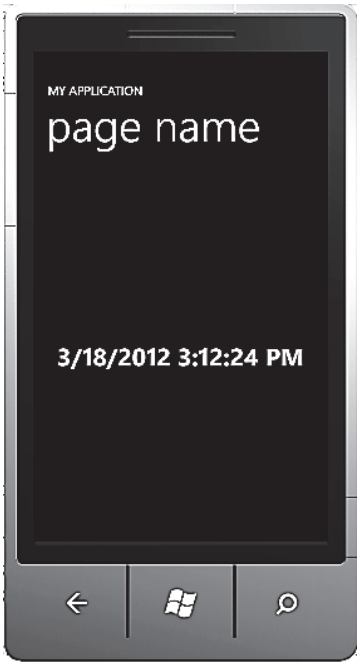


Рис. 2.56. Работа программы в окне эмулятора

Конструктор инициализирует DispatcherTimer, указывая ему вызывать OnTimerTick (По тику таймера) раз в секунду. Обработчик этого события просто преобразует текущее время в строку, чтобы присвоить ее как значение TextBlock (рис. 2.56).

Несмотря на то что DispatcherTimer определен в пространстве имен System.Windows.Threading, метод OnTimerTick вызывается в одном потоке со всем приложением. В противном случае приложение не имело бы прямого доступа к TextBlock. Элементы Silverlight и связанные с ними объекты не являются потокобезопасными и будут препятствовать доступу к ним из других потоков.

Часы – это еще одно приложение на Silverlight в данном подразделе, в котором свойство Text элемента TextBlock меняется динамически во время выполнения. Новое значение выводится как по мановению волшебной палочки, без всякой дополнительной работы. Все это очень отличается от более ранних сред работы с графикой, в которых использовались Windows API или MFC. В них приложение выполняет отрисовку «по требованию», т.е. когда область окна становится недействительной и требует перерисовки или когда приложение намеренно объявляет область недействительной, чтобы вызвать принудительную перерисовку.

Зачастую кажется, что приложение на Silverlight вообще ничего не отрисовывает. По сути своей Silverlight – это слой визуальной компоновки, который работает в заданном графическом режиме и организует все визуальные элементы в единую композицию. Элементы, такие как TextBlock, существуют как действительные сущности этого слоя композиции. В некоторый момент TextBlock формирует собственное визуальное представление и выполняет повторную отрисовку при изменении одного из свойств, таких как Text, но все, что он отрисовывает, сохраняется вместе с визуализированным выводом всех остальных элементов дерева визуальных элементов.

Для сравнения, приложение на XNA выполняет отрисовку для каждого нового кадра экрана. Это концептуально отличается от более ранних

сред разработки для Windows, так же как и от Silverlight. Это очень мощная возможность, но всем прекрасно известно, чем чревата такая мощь.

Иногда экран приложения XNA статичен, программе нет необходимости обновлять его с каждым кадром. Для сохранения энергии метод Update может вызывать метод SuppressDraw (Отменить отрисовку) класса Game, чтобы воспрепятствовать вызову соответствующего метода Draw. Метод Update по-прежнему будет вызываться 30 раз в секунду, потому что он должен выполнять проверку пользовательского ввода, но если код в Update вызывает SuppressDraw, метод Draw не будет выполняться в этом игровом цикле. Если метод Update не вызывает SuppressDraw, то метод Draw выполняется.

Программе на XNA, моделирующей часы, не нужен таймер, потому что таймер уже встроен в обычный игровой цикл. Для создаваемых здесь часов мы не предполагаем отображения миллисекунд, т.е. экран должен обновляться лишь каждую секунду. Применим метод SuppressDraw, который будет предотвращать лишние вызовы метода Draw.

Рассмотрим поля XnaSimpleClock:

Проект XNA: XnaSimpleClock Файл: Game1.cs (фрагмент, демонстрирующий поля)

```
public class Game1 : Microsoft.Xna.Framework.Game
{  GraphicsDeviceManager graphics;
   SpriteBatch spriteBatch;
   SpriteFont segoe14;
   Viewport viewport;
   Vector2 textPosition;
   StringBuilder text = new StringBuilder();
   DateTime lastDateTime;
...
}
```

Обратите внимание, что вместо того чтобы определять поле text типа string, задано StringBuilder. При создании в методе Update новых строк для отображения во время метода Draw (как будет делать данное приложение) следует использовать StringBuilder. Это позволяет избежать распределений кучи, которые возникают в случае применения обычного типа string. Наше приложение будет лишь создавать новую строку каждую секунду, поэтому нам на самом деле нет особой необходимости применять здесь именно StringBuilder, но сделаем это в качестве тренировки. Чтобы работать со StringBuilder, необходимо добавить директиву для пространства имен System.Text.

Также обратите внимание на поле LastDateTime (Текущие дата и время). Оно используется в методе Update для определения необходимости обновления отображаемого времени.

Метод LoadContent принимает шрифт и окно просмотра экрана:

Проект XNA: XnaSimpleClock Файл: Game1.cs (фрагмент)

```
protected override void LoadContent()
{
spriteBatch = new SpriteBatch(GraphicsDevice);
segoe14 = this.Content.Load<SpriteFont>("Segoe14");
viewport = this.GraphicsDevice.Viewport;
}
```

Логика сравнения значений DateTime (Дата и время) для определения, изменилось ли время с момента последнего вызова метода Update, несколько запутанная, потому что объекты DateTime, полученные в ходе двух последовательных вызовов метода Update будут разными всегда. Отличаться в них будут значения поля Millisecond (Миллисекунды). Поэтому новое значение DateTime вычисляется на основании текущего времени, полученного посредством DateTime.Now, но за вычетом миллисекунд:

Проект XNA: XnaSimpleClock Файл: Game1.cs (фрагмент)

```
protected override void Update(GameTime gameTime)
{
// Обеспечиваем возможность выхода из игры
if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                                                                    this.Exit();
// Получаем DateTime без миллисекунд
DateTime dateTime = DateTime.Now;
dateTime = dateTime - new TimeSpan(0, 0, 0, 0, dateTime.Millisecond);
if (dateTime != lastDateTime)
        { text.Remove(0, text.Length);
          text.Append(dateTime);
          Vector2 textSize = segoe14.MeasureString(text);
          textPosition = new Vector2((viewport.Width - textSize.X) / 2,
                                                    (viewport.Height - textSize.Y) / 2);
  lastDateTime = dateTime;
}
else
{
  SuppressDraw();
}
base.Update(gameTime);
}
```

Здесь все просто. Если время изменилось, вычисляются новые значения text, textSize и textPosition. Поскольку text – это StringBuilder, а не string, старое содержимое удаляется и сохраняется новое. В методе

MeasureString класса SpriteFont есть перегрузка для StringBuilder, поэтому вызов выглядит точно так же.

Если время не изменилось, то вызывается SuppressDraw. В результате метод Draw вызывается лишь раз в секунду.

DrawString также имеет перегрузку для StringBuilder:

Проект XNA: XnaSimpleClock Файл: Game1.cs (фрагмент)

```
protected override void Draw(GameTime gameTime)
{
GraphicsDevice.Clear(Color.Navy);
spriteBatch.Begin();
spriteBatch.DrawString(segoe14, text, textPosition, Color.White);
spriteBatch.End();
base.Draw(gameTime);
}
```

И вот результат (рис. 2.57).

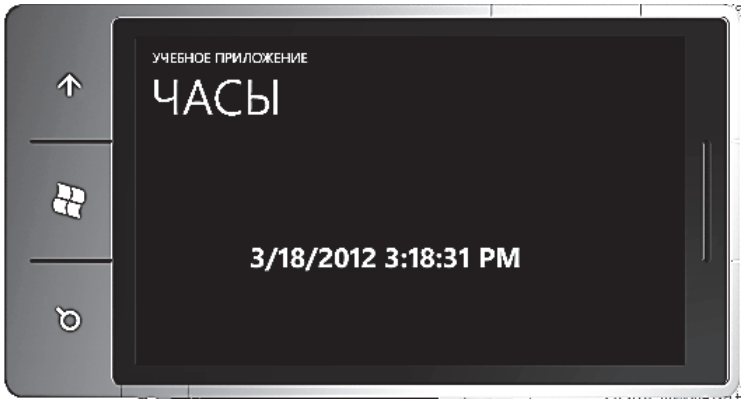


Рис. 2.57. Вид программы в окне эмулятора

Могут возникнуть некоторые сложности с использованием SuppressDraw во время первого запуска программы. Но применение этого метода является одной из основных методик в XNA для сокращения энергопотребления приложений.

2.20.3. Основы работы с сенсорным вводом

Даже для опытных разработчиков на Silverlight и XNA Windows Phone 7 предлагает возможность, которая, скорее всего, окажется новой и необычной. Экран телефона чувствителен к прикосновению и существенно отличается от старых сенсорных экранов, в основном повторяющих ввод с мыши, или экранов планшетных устройств, которые могут распознавать рукописный ввод.

Мультисенсорный экран устройства Windows Phone 7 может распознавать одновременное касание как минимум в четырех точках. Именно обработка взаимодействия этих одновременных касаний делает задачу реализации мультисенсорного ввода такой сложной для разработчиков. Но для данного пособия отведена несколько менее амбициозная цель. Требуется познакомить читателей с сенсорными интерфейсами в контексте примеров приложений, которые могут реагировать на простые касания.

Тестирование критически важного кода реализации мультисенсорного ввода необходимо выполнять на реальном устройстве, работающем под управлением Windows Phone 7. Между тем эмулятор телефона будет реагировать на действия мыши и преобразовывать их в сенсорный ввод. Чтобы использовать сенсорный ввод непосредственно на эмуляторе, его необходимо запустить под управлением Windows 7 на устройстве с поддерживающим мультисенсорный ввод экраном.

Приведенные в данном подразделе программы во многом схожи с простыми приложениями «Hello, Windows Phone 7!» из подраздела 2.11. Единственное отличие в том, что при касании текста пальцем он будет случайным образом менять свой цвет, а при касании вне области текста он будет опять возвращаться к исходному белому цвету (или любому другому цвету, какой будет применен к тексту при запуске программы).

В программе на Silverlight сенсорный ввод реализован через события. В приложении на XNA сенсорный ввод передается через статический класс, опрашиваемый в ходе выполнения метода Update. Одно из основных назначений XNA-метода Update – проверка состояния сенсорного ввода и внесение изменений, которые отображаются на экране во время выполнения метода Draw.

2.20.4. Обработка простого касания в XNA

В XNA устройства мультисенсорного ввода называют сенсорной панелью. Для обработки такого ввода используются методы статического класса TouchPanel (Сенсорная панель). Имеется также возможность обработки жестов, но пока начнем с более простых данных касания.

Можно (но это не является обязательным) получать сведения о самом устройстве мультисенсорного ввода через вызов статического метода TouchPanel.GetCapabilities. Объект TouchPanelCapabilities (Возможности сенсорной панели), возвращаемый этим методом, имеет два свойства:

- IsConnected (Подключен) имеет значение true, если сенсорная панель доступна. Для телефона его значение всегда true.
- MaximumTouchCount (Максимальное число касаний) возвращает количество точек касания (как минимум 4 для телефона).

Для большинства задач достаточно использовать один из двух статических методов TouchPanel. Для получения ввода от простого касания при каждом вызове метода Update после запуска программы, скорее всего, будет вызываться этот метод:

```
TouchCollection touchLocations = TouchPanel.GetState();
```

TouchCollection (Коллекция касаний) – это коллекция, включающая нуль или более объектов TouchLocation (Место касания). TouchLocation имеет три свойства:

• State (Состояние). Его значениями являются элементы перечисления TouchLocationState (Состояние места касания): Pressed (Нажат), Moved (Перемещен), Released (Высвобожден).

• Position (Местоположение) – это Vector2, обозначающий положение пальца относительно верхнего левого угла окна просмотра.

• Id – целое число, идентифицирующее отдельное касание от состояния Pressed до Released, т.е. в течение всего времени касания.

Если ни один палец не касается экрана, коллекция TouchCollection пуста. Когда палец впервые касается экрана, в TouchCollection появляется объект, свойство State которого имеет значение Pressed. Последующие вызовы TouchPanel.GetState покажут, что значение State объекта TouchLocation равно Moved, даже если фактически палец никуда не перемещался. Когда палец будет убран с экрана, свойство State объекта TouchLocation примет значение Released. Последующие вызовы TouchPanel.GetState продемонстрируют, что коллекция TouchCollection опять пуста.

Примечание. Если палец быстро «постукивает» по экрану, т.е. поднимается и опускается на экран с частотой примерно 1/30 секунды, то свойство State объекта TouchLocation от значения Pressed сразу перейдет к значению Released, минуя состояния Moved.

Описано касание всего одним пальцем. Как правило, экрана будут касаться множество пальцев, и опускаться, перемещаться и покидать экран они будут независимо друг от друга. Для отслеживания отдельного касания используется свойство Id (Идентификатор). Для одного отдельно взятого касания Id будет неизменным от состояния Pressed, на протяжении всех перемещений (значения Moved) и до состояния Released.

Часто при работе с простым касанием используется объект Dictionary (Словарь) с ключами, созданными на основании свойства Id, для извлечения данных конкретного касания.

TouchLocation также имеет очень удобный метод TryGetPreviousLocation (Попытаться получить предыдущее местоположение), который вызывается следующим образом:

```
TouchLocation previousTouchLocation; bool success =
                    touchLocation.TryGetPreviousLocation(out previousTouchLocation);
```

Вызов этого метода практически всегда происходит, когда touchLocation.State имеет значение Moved, для получения предыдущего местоположения и вычисления разницы. Если значение touchLocation.State равно Pressed, TryGetPreviousLocation возвратит false, и значением previousTouchLocation.State будет элемент перечисления TouchLocation-State.Invalid. Такие же результаты будут получены в случае вызова этого метода для TouchLocation, который был возвращен TryGetPreviousLocation.

Предлагаемое здесь приложение меняет цвет текста при касании пользователем экрана, т.е. обработка TouchPanel.GetStates будет относительно простой. Логика приложения будет проверять только объекты TouchLocation, значение свойства State которых равно Pressed.

Назовем этот проект XnaTouchHello. Как и во всех рассматриваемых до этого проектах на XNA, нам понадобится шрифт, который можно немного увеличить, чтобы обеспечить более удобную для касания мишень. Потребуется еще несколько дополнительных полей:

Проект XNA: XnaTouchHello Файл: Game1.cs (фрагмент, демонстрирующий поля)

```
public class Game1 : Microsoft.Xna.Framework.Game
{
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        Random rand = new Random();
        string text = "Hello, Windows Phone 7!";
        SpriteFont segoe36;
        Vector2 textSize;
        Vector2 textPosition;
        Color textColor = Color.White;
 …
}
```

Метод LoadContent аналогичен используемому ранее, за исключением того, что textSize сохраняется как поле. Это обеспечит возможности доступа к нему при последующих вычислениях:

Проект XNA: XnaTouchHello Файл: Game1.cs (фрагмент)

```
protected override void LoadContent()
{
        spriteBatch = new SpriteBatch(GraphicsDevice);
        segoe36 = this.Content.Load<SpriteFont>("Segoe36");
        textSize = segoe36.MeasureString(text);
        Viewport viewport = this.GraphicsDevice.Viewport;
textPosition = new Vector2((viewport.Width - textSize.X) / 2,
                                                (viewport.Height - textSize.Y) / 2);
}
```

Как это свойственно приложениям на XNA, «действие» происходит преимущественно в методе Update. Этот метод вызывает TouchPanel.GetStates и затем поэлементно обходит коллекцию объектов TouchLocation, выбирая те из них, значение State которых равно Pressed.

Проект XNA: XnaTouchHello Файл: Game1.cs (фрагмент)

```
protected override void Update(GameTime gameTime)
{
if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();
TouchCollection touchLocations = TouchPanel.GetState();
foreach (TouchLocation touchLocation in touchLocations)
        {
        if (touchLocation.State == TouchLocationState.Pressed)
                { Vector2 touchPosition = touchLocation.Position;
                  if (touchPosition.X >= textPosition.X &&
                  touchPosition.X < textPosition.X + textSize.X &&
                  touchPosition.Y >= textPosition.Y &&
                  touchPosition.Y < textPosition.Y + textSize.Y)
                        {       textColor       =       new       Color((byte)rand.Next(256),
                (byte)rand.Next(256),
                        (byte)rand.Next(256));
                }
        else
        { textColor = Color.White; }
                        }
        }
base.Update(gameTime);
}
```

Если Position оказывается где-нибудь внутри области, занимаемой текстовой строкой, то полю textColor (Цвет текста) присваивается случайное значение RGB для цвета с помощью одного из конструкторов структуры Color (Цвет). В противном случае textColor присваивается значение Color.White.

Метод Draw практически аналогичен используемому в предыдущих примерах, только цвет текста теперь стал переменным:

Проект XNA: XnaTouchHello Файл: Game1.cs (фрагмент)

```
protected override void Draw(GameTime gameTime)
{
        this.GraphicsDevice.Clear(Color.Navy);
        spriteBatch.Begin();
        spriteBatch.DrawString(segoe36, text, textPosition, textColor);
         spriteBatch.End();
        base.Draw(gameTime);
}
```

Единственная проблема в том, что касание не так строго детерминировано, как этого хотелось бы. Даже при касании одним пальцем контактов с экраном может быть несколько. В некоторых случаях один и тот же цикл foreach в методе Update может задавать textColor несколько раз! После запуска программа работает следующим образом: «тычем» пальцем в надпись, и она меняет цвет (рис. 2.58).

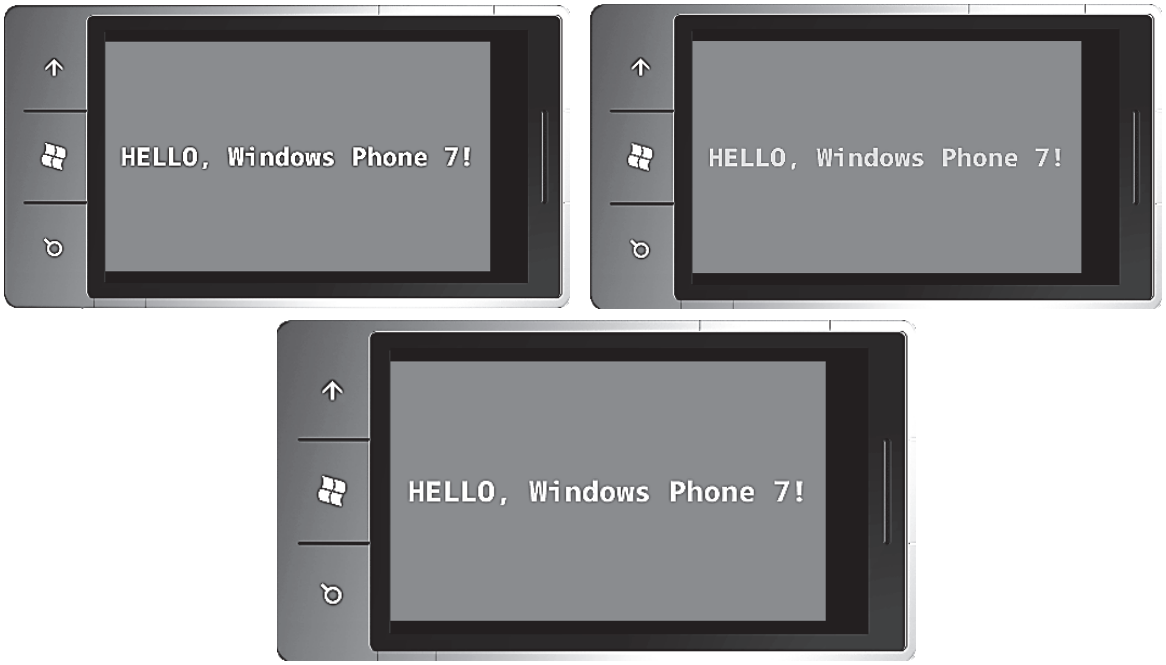


Рис. 2.58. Изменение цвета шрифта на сенсорном экране

Еще раз отмечаем, что на обычном экране монитора эффекта «обработка касания пальцем» мы не увидим. Экран ДОЛЖЕН БЫТЬ сенсорным.

2.20.5. Обработка жестов в XNA

Класс TouchPanel также включает возможность распознавания жестов, что демонстрирует проект XnaTapHello. В данном проекте используются те же поля, что и в XnaTouchHello, но несколько отличается метод LoadContent:

Проект XNA: XnaTapHello Файл: Game1.cs (фрагмент)

```
protected override void LoadContent()
{
        spriteBatch = new SpriteBatch(GraphicsDevice);
        segoe36 = this.Content.Load<SpriteFont>("Segoe36");
        textSize = segoe36.MeasureString(text);
        Viewport viewport = this.GraphicsDevice.Viewport;
        textPosition = new Vector2((viewport.Width - textSize.X) / 2,
```

```
                                        (viewport.Height - textSize.Y) / 2);
        TouchPanel.EnabledGestures = GestureType.Tap;
}
```

Обратите внимание на последнее выражение. GestureType (Тип жеста) – это перечисление, элементами которого являются Tap (Касание), DoubleTap (Двойное касание), Flick (Скольжение), Hold (Удержание), Pinch1 (Сведение), PinchComplete (Сведение завершенно), FreeDrag (Произвольное перетягивание), HorizontalDrag (Перетягивание по горизонтали), VerticalDrag (Перетягивание по вертикали) и DragComplete (Перетягивание завершено). Эти элементы определены как битовые флаги, таким образом, они могут комбинироваться с помощью побитового C#-оператора OR (они обычно используются для операции Zoom (Масштабирование) путем сведения или разведение пальцев).

Метод Update совсем другой:

Проект XNA: XnaTapHello Файл: Game1.cs (фрагмент)

```
protected override void Update(GameTime gameTime)
{
// Обеспечиваем возможность выхода из игры
if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
 this.Exit();
while (TouchPanel.IsGestureAvailable)
        {
        GestureSample gestureSample = TouchPanel.ReadGesture();
        if (gestureSample.GestureType == GestureType.Tap)
                { Vector2 touchPosition = gestureSample.Position;
                if (    touchPosition.X >= textPosition.X &&
                        touchPosition.X < textPosition.X + textSize.X &&
                        touchPosition.Y >= textPosition.Y &&
                        touchPosition.Y < textPosition.Y + textSize.Y)
                { textColor = new Color((byte)rand.Next(256),
                        (byte)rand.Next(256),
                        (byte)rand.Next(256));
                }
else
                        { textColor = Color.White; }
                }
        }
base.Update(gameTime);
}
```

Несмотря на то что данное приложение рассматривает только один тип жеста, код довольно универсален. Если жест доступен, метод TouchPanel.ReadGesture (Прочитать жест) возвращает его как объект типа GestureSample (Пример жеста). Кроме применяемых здесь GestureType и

Position, у этого объекта имеется еще свойство Delta (Приращение), обеспечивающее данные о перемещении в виде объекта Vector2. Для некоторых жестов (таких как Pinch) GestureSample также предоставляет состояние второй точки касания через свойства Position2 и Delta2.

Метод Draw аналогичен используемому в предыдущем случае, но поведение данного приложения будет несколько иным. В предыдущей программе текст меняет цвет, когда палец касается экрана, а здесь изменение цвета происходит, когда палец убирается с экрана. Средству распознавания жестов необходимо дождаться завершения жеста, чтобы определить его тип.

2.20.6. События простого касания в Silverlight

Как и XNA, Silverlight поддерживает два разных программных интерфейса для работы с мультисенсорным вводом, которые можно категоризировать как интерфейс обработки простого и интерфейс обработки сложного касания. Интерфейс обработки простого касания построен на событии Touch.FrameReported, которое очень похоже на XNA-класс TouchPanel. Отличается оно лишь тем, что это событие, и оно не включает обработку жестов.

Интерфейс обработки сложного касания включает три события, определяемые классом UIElement: ManipulationStarted (Обработка началась), ManipulationDelta (Приращение в ходе обработки) и ManipulationCompleted (Обработка завершилась). События Manipulation, как их обобщенно называют, консолидируют взаимодействие множества касаний в движение и коэффициенты масштабирования.

Ядром интерфейса обработки простого касания в Silverlight является класс TouchPoint (Точка касания), экземпляр которого представляет отдельное касание экрана. TouchPoint имеет четыре свойства только для чтения:

1. Action (Действие) типа TouchAction (Действие касания) – перечисление с элементами Down (Вниз), Move (Перемещение) и Up (Вверх).

2. Position типа Point (Точка), значение которого определяется относительно верхнего левого угла конкретного элемента. Будем называть этот элемент опорным.

3. Size типа Size. Это свойство должно представлять область касания (и, следовательно, давление, создаваемое пальцем), но эмулятор Windows Phone 7 не возвращает полезных значений.

4. TouchDevice типа TouchDevice. Объект TouchDevice имеет два свойства только для чтения:

• Id типа int используется для идентификации касаний. Каждое отдельное касание ассоциировано с уникальным Id на протяжении всех событий начиная с Down и до Up.

• DirectlyOver (Непосредственно над) типа UIElement – самый верхний элемент, расположенный прямо под пальцем.

Как видите, Silverlight-объекты TouchPoint и TouchDevice предоставляют преимущественно те же сведения, что и XNA-объект TouchLocation. Свойство DirectlyOver объекта TouchDevice часто очень полезно для определения, какого элемента пользователь касается.

Для использования интерфейса обработки простого касания необходимо установить обработчик статического события Touch.FrameReported:

```
Touch.FrameReported += OnTouchFrameReported;
```

Метод OnTouchFrameReported выглядит следующим образом:

```
void OnTouchFrameReported(object sender, TouchFrameEventArgs args) { … }
```

Этот обработчик события принимает все события касания в ходе выполнения приложения. Объект TouchFrameEventArgs (Аргументы события касания рамки) имеет свойство TimeStamp (Отметка времени) типа int и три метода:

1. GetTouchPoints(refElement) (Получить точки касания) возвращает TouchPointCollection (Коллекция точек касания).

2. GetPrimaryTouchPoint(refElement) (Получить основную точку касания) возвращает один TouchPoint.

3. SuspendMousePromotionUntilTouchUp() (Приостановить перемещения мыши до завершения касания).

В общем случае вызывается метод GetTouchPoints и в него передается опорный элемент. Значения свойств Position объектов TouchPoint в возвращенной коллекции определяются относительно этого элемента. Если передать null в GetTouchPoints, то значения свойств Position будут установлены относительно верхнего левого угла окна просмотра приложения.

Между опорным элементом и элементом DirectlyOver нет никакой связи. Событие всегда обрабатывает все касания приложения в целом. Вызов GetTouchPoints или GetPrimaryTouchPoints для конкретного элемента не означает, что будут обрабатываться касания только этого элемента, это означает лишь то, что значение свойства Position будет определяться относительно этого элемента (поэтому координаты Position вполне могут быть отрицательными, если место касания находится слева или над опорным элементом). Элемент DirectlyOver определяет элемент, находящийся непосредственно под пальцем.

Для разговора о втором и третьем методах необходимо сделать небольшое вступление. Событие Touch.FrameReported появилось в Silverlight для настольных приложений, в которых для логики обработки событий мыши существующих элементов управления удобно автоматически использовать сенсорный ввод. По этой причине события касания приравнены к событиям мыши.

Но это распространяется только на «первую» точку касания, т.е. на действия пальца, коснувшегося экрана первым, когда ни один другой палец его не касается. Если Вы не хотите, чтобы действия этого касания трактовались как события мыши, обработчик события должен начинаться так:

```
void OnTouchFrameReported(object sender, TouchFrameEventArgs args)
{
        TouchPoint primaryTouchPoint = args.GetPrimaryTouchPoint(null);
        if (primaryTouchPoint != null &&
                primaryTouchPoint.Action == TouchAction.Down)
                { args.SuspendMousePromotionUntilTouchUp();
                }
…
}
```

Метод SuspendMousePromotionUntilTouchUp может вызываться только в момент первого касания первым пальцем, когда все остальные пальцы еще не коснулись экрана.

Для Windows Phone 7 такая логика представляет некоторые проблемы. Как сказано выше, по сути происходит отключение обработки событий мыши для всего приложения. Если приложение для телефона включает элементы управления Silverlight, изначально написанные для ввода с помощью мыши и не обновленные для приема сенсорного ввода, то эти элементы управления фактически будут деактивированы.

Конечно, можно проверять свойство DirectlyOver и делать селективную приостановку обработки событий мыши. Но в телефоне не должно быть элементов, обрабатывающих ввод с помощью мыши, кроме тех, которые не обрабатывают сенсорный ввод! Поэтому, вероятно, больше смысла будет в том, чтобы никогда не приостанавливать обработку событий мыши.

Решение остается за вами и за устаревшими элементами управления, обрабатывающими ввод посредством мыши. Для приложения, которое Вы пишете, важна только первоначальная точка касания в момент, когда ее TouchAction имеет значение Down, поэтому предлагается использовать ту же самую логику.

В проекте SilverlightTouchHello TextBlock описывается в XAML-файле:

Проект Silverlight: SilverlightTouchHello Файл: MainPage.xaml (фрагмент)

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <TextBlock Name="txtblk"
                Text="Hello, Windows Phone 7!"
                Padding="0 34"
                HorizontalAlignment="Center"
```

```
                VerticalAlignment="Center" />
</Grid>
```

Обратите внимание на значение Padding. Известно, что свойство FontSize отображаемого здесь текста равно 20 пикселам, это обеспечивает TextBlock высотой около 27 пикселов. Также рекомендуется, чтобы мишень касания не была меньше 9 мм. Если разрешение экрана телефона равно 264 DPI, то 9 мм – это 94 пиксела (9 мм разделить на 25,4 мм/дюйм и умножить на 264 пиксела/дюйм). TextBlock не хватает 67 пикселов. Поэтому задаем значение Padding, которое добавляет по 34 пиксела сверху и снизу (но не по бокам).

Применим здесь Padding, а не Margin, потому что Padding – это область внутри TextBlock. Таким образом, TextBlock фактически становится больше, чем предполагает размер текста. Margin – это область вне TextBlock, она не является частью TextBlock, и касания ее не учитываются как касания TextBlock.

Рассмотрим файл выделенного кода полностью. Конструктор MainPage определяет обработчик событий Touch.FrameReported.

Проект Silverlight: SilverlightTouchHello Файл: MainPage.xaml.cs

```
using System;
using System.Windows.Input;
using System.Windows.Media;
using Microsoft.Phone.Controls;
namespace SilverlightTouchHello
{
public partial class MainPage : PhoneApplicationPage
{
Random rand = new Random();
Brush originalBrush;
public MainPage()
{
InitializeComponent();
 originalBrush = txtblk.Foreground
Touch.FrameReported += OnTouchFrameReported;
}

 void OnTouchFrameReported(object sender, TouchFrameEventArgs args)
{
        TouchPoint primaryTouchPoint = args.GetPrimaryTouchPoint(null);
if (primaryTouchPoint != null &&
        primaryTouchPoint.Action == TouchAction.Down)
        {
        if (primaryTouchPoint.TouchDevice.DirectlyOver == txtblk)
        {
```

```
                    txtblk.Foreground = new SolidColorBrush(
                            Color.FromArgb(255, (byte)rand.Next(256),
                            (byte)rand.Next(256), (byte)rand.Next(256)));
            }
        else
        {
        txtblk.Foreground = originalBrush;
         }
        }}}}
```

Этот обработчик событий обрабатывает только первые точки касания, для которых Action имеет значение Down. Если свойство DirectlyOver возвращает элемент txtblk, то создается случайный цвет. В отличие от XNA структура Color в Silverlight не имеет конструктора для задания цвета как комбинации значений красного, зеленого и синего, но в ней есть статический метод FromArgb, который создает объект Color на основании значений альфа, красного, зеленого и синего каналов, где альфа – это прозрачность. Для получения непрозрачного цвета задайте альфа-каналу значение 255. Это очевидно не во всех файлах XAML, но свойство Foreground типа Brush (Кисть) – это абстрактный класс, от которого наследуется SolidColorBrush (Одноцветная кисть).

Если DirectlyOver не txtblk, то цвет текста не меняется на белый. В случае если бы пользователь выбрал цветовую тему с черным текстом на белом фоне, это привело бы к тому, что текст исчез бы с экрана. Вместо этого свойству Foreground присваивается изначально заданный для TextBlock цвет. Он определяется в конструкторе.

2.20.7. События Manipulation

Интерфейс обработки сложного касания в Silverlight включает три события: ManipulationStarted, ManipulationDelta и ManipulationCompleted. Эти события не занимаются отдельными касаниями, они консолидируют действия множества касаний в операции преобразования и масштабирования. Также они аккумулируют сведения о скорости, поэтому могут использоваться для реализации инерции, несмотря на то что не поддерживают ее напрямую.

События Manipulation будут рассмотрены более подробно далее. В этом подразделе применяется ManipulationStarted просто для выявления контакта пальца с экраном, но не производится обработка последующих действий.

Тогда как Touch.FrameReported обеспечивал данные касания для всего приложения, события Manipulation делают это для отдельных элемен-

тов. Таким образом, в SilverlightTapHello1 обработчик события ManipulationStarted может быть закреплен за TextBlock:

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
 <TextBlock Text="Hello, Windows Phone 7!"
        Padding="0 34"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"                                    ManipulationStart-
ed="OnTextBlockManipulationStarted" />
 </Grid>
```

Файл MainPage.xaml.cs включает такой обработчик события:

Проект Silverlight: SilverlightTapHello1 Файл: MainPage.xaml.cs (фрагмент)

```
public partial class MainPage : PhoneApplicationPage
{ Random rand = new Random();
public MainPage()
        { InitializeComponent(); }

        void OnTextBlockManipulationStarted(object sender,
                                            ManipulationStartedEventArgs args)
        { TextBlock txtblk = sender as TextBlock;
        Color clr = Color.FromArgb(255, (byte)rand.Next(256),
                (byte)rand.Next(256), (byte)rand.Next(256));
        txtblk.Foreground = new SolidColorBrush(clr); args.Complete();
        }
}
```

Обработчик события получает элемент, формирующий это событие, из аргумента sender. Им всегда будет TextBlock. Сведения о TextBlock также доступны из свойства args.OriginalSource и свойства args.ManipulationContainer.

Обратите внимание на вызов метода Complete (Завершить) аргументов события в конце. Он не является обязательным, но эффективен для уведомления системы о том, что в дальнейшем события Manipulation для обработки этого касания не нужны.

Данная программа некорректна. Если выполнить ее, то можно заметить, что она работает только частично. Прикосновение к TextBlock меняет цвет текста случайным образом, но если коснуться вне TextBlock, цвет не возвращается к исходному белому. Поскольку это событие было задано для TextBlock, обработчик события вызывается, только когда пользователь касается TextBlock. Программа не обрабатывает никаких других событий Manipulation.

Приложение, удовлетворяющее исходным техническим условиям, должно обрабатывать все события касания страницы. Обработчик события

ManipulationStarted должен быть определен для MainPage, а не только для TextBlock.

Хотя, безусловно, этот вариант возможен, но существует более простой способ. Класс UIElement определяет все события Manipulation. Но класс Control (от которого наследуется MainPage) дополняет эти события защищенными виртуальными методами. Можно не задавать обработчик события ManipulationStarted в MainPage, а просто перегрузить виртуальный метод OnManipulationStarted (Когда обработка началась).

Такой подход реализован в проекте SilverlightTapHello2. Файл XAML не обрабатывает никакие события, но задает имя для TextBlock, которое может использоваться в коде:

Проект Silverlight: SilverlightTapHello2 Файл: MainPage.xaml (фрагмент)

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
 <TextBlock Name="txtblk"
        Text="Hello, Windows Phone 7!"
        Padding="0 34"
        HorizontalAlignment="Center"
        VerticalAlignment="Center" />
</Grid>
```

Класс MainPage перегружает метод OnManipulationStarted:

Проект Silverlight: SilverlightTapHello2 Файл: MainPage.xaml.cs (фрагмент)

```
public partial class MainPage : PhoneApplicationPage
{
Random rand = new Random();
Brush originalBrush;

public MainPage()
        { InitializeComponent();
        originalBrush = txtblk.Foreground;
        } protected override

void OnManipulationStarted(ManipulationStartedEventArgs args)
         { if (args.OriginalSource == txtblk)
        { txtblk.Foreground = new SolidColorBrush(
                                        Color.FromArgb(255, (byte)rand.Next(256),
                                        (byte)rand.Next(256), (byte)rand.Next(256)));
        }
         else
        { txtblk.Foreground = originalBrush;
        }
args.Complete();
```

```
base.OnManipulationStarted(args);
        }
}
```

В ManipulationStartedEventArgs (Аргументы события обработка началась) свойство OriginalSource (Первоначальный источник) указывает на источник события, т.е. на расположенный поверх всех остальных элемент, которого касается пользователь. Если значение этого свойства равно txtblk, то метод создает случайный цвет для свойства Foreground, если нет, то свойство Foreground возвращается к исходному цвету.

В этом методе OnManiulationStarted обрабатываются события MainPage, но свойство OriginalSource показывает, что фактически событие произошло в элементе, расположенном ниже в визуальном дереве. Это так называемая возможность обработки маршрутизированных событий в Silverlight.

Маршрутизированные события.

В разработке для Microsoft Windows ввод посредством клавиатуры и мыши всегда поступает в конкретные элементы управления. Ввод с клавиатуры всегда направляется в элемент управления, имеющий фокус ввода. Ввод с мыши всегда направляется в активный элемент управления, расположенный непосредственно под указателем мыши, поверх всех остальных элементов. Так же обрабатывается ввод со стилуса и касание. Но иногда такая логика неудобна. В некоторых случаях нижележащий элемент управления нуждается в пользовательском вводе больше, чем элемент, расположенный непосредственно под указателем устройства ввода.

Для большей гибкости Silverlight реализует систему обработки маршрутизированных событий. Большинство событий пользовательского ввода, включая три события Manipulation, формируются соответственно парадигме, применяемой в Windows: они формируются расположенным поверх остальных активным элементом, к которому прикасается пользователь. Но если этот элемент не обрабатывает данное событие, оно передается в его родительский элемент и далее вверх по визуальному дереву, заканчивая элементом PhoneApplicationFrame. Любой элемент в этой цепочке может захватывать этот ввод и каким-то образом его обрабатывать, а также останавливать его дальнейшее распространение вверх по дереву.

Вот почему можно перегрузить метод OnManipulationStarted в MainPage и принимать события Manipulation для TextBlock. По умолчанию TextBlock не обрабатывает эти события.

Аргумент события ManipulationStarted – ManipulationStartedEventArgs. Он наследуется от RoutedEventArgs (Аргументы маршрутизированного события). Именно RoutedEventArgs определяет свойство OriginalSource, обозначающее элемент, который сформировал событие. Использование этих свойств предлагает другой подход, сочетающий в себе две методики,

представленные в SilverlightTapHello1 и SilverlightTapHello2. Рассмотрим XAML-файл проекта SilverlightTapHello3:

Проект Silverlight: SilverlightTapHello3 Файл: MainPage.xaml (фрагмент)

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <TextBlock Name="txtblk"
                Text="Hello, Windows Phone 7!"
                Padding="0 34"
                HorizontalAlignment="Center"
                VerticalAlignment="Center"
                ManipulationStarted="OnTextBlockManipulationStarted" />
 </Grid>
```

Имя (Name) TextBlock аналогично используемому в первой программе. В ней обработчик события ManipulationStarted задан для TextBlock. И обработчик события, и перегруженный OnManipulationStarted находятся в файле выделенного кода:

Проект Silverlight: SilverlightTapHello3 Файл: MainPage.xaml.cs (фрагмент)

```
public partial class MainPage : PhoneApplicationPage
        { Random rand = new Random();
                Brush originalBrush;
        public MainPage()
                { InitializeComponent();
                originalBrush = txtblk.Foreground;
                }

void OnTextBlockManipulationStarted(object sender, ManipulationStartedEventArgs
args)
        { txtblk.Foreground = new SolidColorBrush(
                        Color.FromArgb(255, (byte)rand.Next(256),
                        (byte)rand.Next(256), (byte)rand.Next(256)));
        args.Complete();
        args.Handled = true;
        }

protected override void OnManipulationStarted(ManipulationStartedEventArgs args)
        { txtblk.Foreground = originalBrush;
        args.Complete();
        base.OnManipulationStarted(args);
        }
}
```

Логика разнесена на два метода, что делает всю обработку намного более элегантной. Метод OnTextBlockManipulationStarted принимает собы-

тия, только когда происходит касание TextBlock. Событие OnManipulationStarted отвечает за все события MainPage.

На первый взгляд может показаться, что здесь ошибка. После вызова OnTextBlockManipulationStarted событие продолжает свое «путешествие» вверх по визуальному дереву, и OnManipulationStarted возвращает цвет к исходному белому. Но на самом деле происходит не это. Правильность выполнения всей логики обеспечивает выражение в конце обработчика OnTextBlockManipulationStarted для TextBlock:

```
args.Handled = true;
```

Это выражение говорит о том, что событие уже обработано и не должно передаваться далее вверх по визуальному дереву. Удалите это выражение – и TextBlock никогда не изменит своего исходного цвета.

3. РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЯ НА РЕАЛЬНОМ УСТРОЙСТВЕ

В каталоге \bin\Debug проекта, созданного Visual Studio для Silverlight Phone, можно увидеть файл *.xap (звездочка означает название проекта). Это так называемый XAP-файл (произносится «зап»). Именно этот файл развертывается на телефоне или эмуляторе телефона.

XAP-файл – это пакет файлов, упакованных в очень популярном формате сжатия ZIP. Чтобы заглянуть внутрь файла, переименуйте *.xap в *.zip. Там вы обнаружите несколько файлов растровых изображений, являющихся частью проекта, XML-файл, XAML-файл и файл *.dll, который является скомпилированным двоичным файлом (кодом) приложения.

Все ресурсы, необходимые приложению, можно сделать частью проекта Visual Studio и добавить в этот XAP-файл. Приложение будет осуществлять доступ к этим файлам во время выполнения.

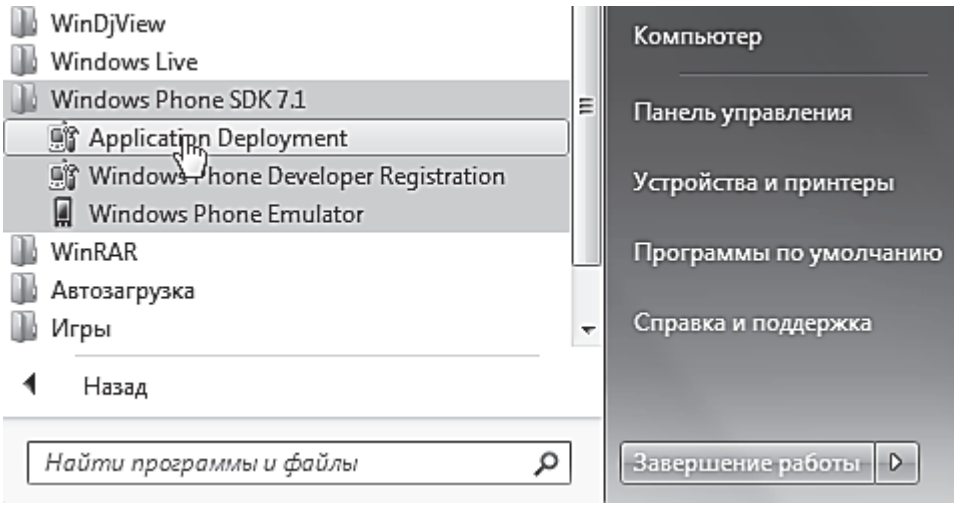


Рис. 3.1. Утилита установки программы

После установки пакета Windows Phone 7 Developer Tools Вы могли заметить на своем компьютере ярлык к программе Application Deployment (рис. 3.1), которая позволяет открывать XAP-файл и устанавливать его на эмулятор или реальное устройство.

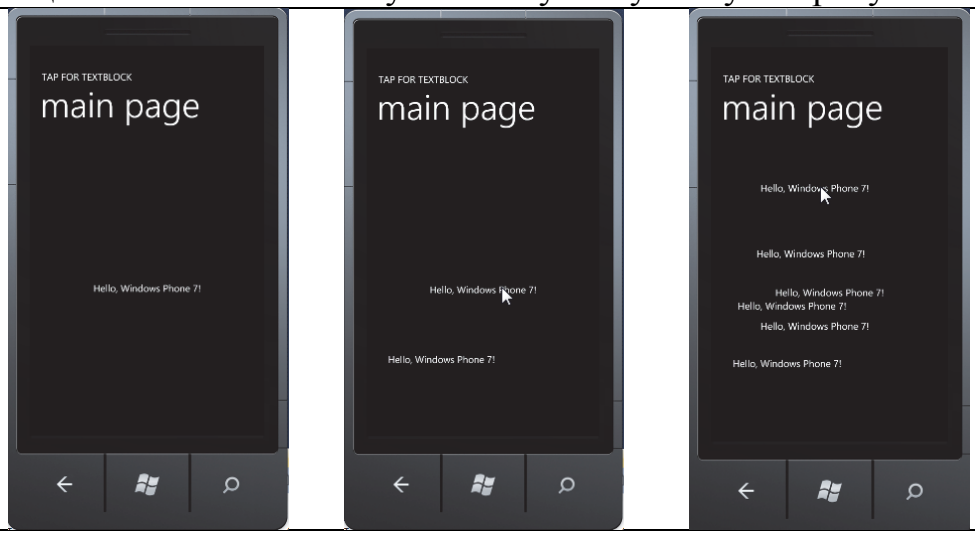
Если Вы написали полезную программу, то, вероятно, Вам хотелось бы получить отзывы о ней от своих знакомых разработчиков. Вы можете поделиться своей программой с друзьями (считайте, что дали им потестировать бета-версию). При помощи Application Deployment Вы можете установить у себя программу, которую пришлет Вам знакомый разработчик, а он в свою очередь установит Ваше приложение. Делается это очень просто. Запускаете Application Deployment, выбираете XAP-файл на Вашем компьютере и щелкаете по кнопке Deploy. В качестве конечного устройства можете выбрать эмулятор или свой телефон.

4. КОНТРОЛЬНЫЕ ЗАДАНИЯ ПО КУРСУ

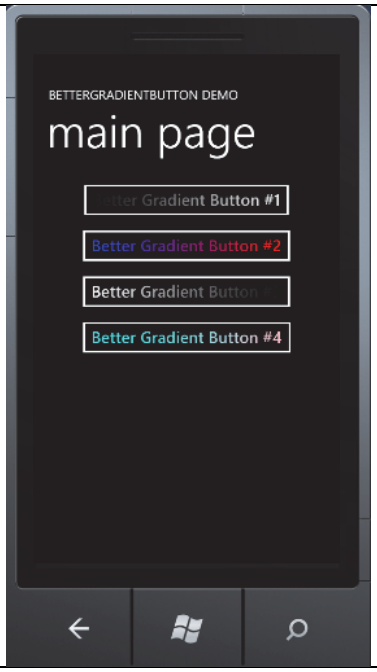
Понятно, что материал, изложенный выше, не покрывает и десятой части рассматриваемой технологии, но общее представление о приемах создания программного обеспечения для мобильных устройств Вы получили. Теперь следует закрепить полученные навыки на практических задачах.

В контрольную работу входят два задания. В первом необходимо создать простую программу для смартфона на платформе Silverlight, во втором – на платформе XNA.

Задание 1 (платформа Silverlight). По вариантам указано, что должна воспроизводить на экране программа, и приведен примерный вид экрана в момент работы программы (стрелки – показывают направление движения объекта на экране, их программировать не надо).

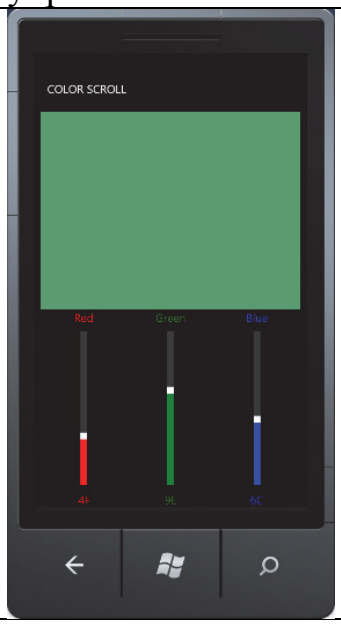
1. На строке «HELLO, Window Phone 7!» щёлкаем мышью – и в случайном месте появляется новая аналогичная строка, по которой тоже можно щелкать мышью и получать новую случайную строку


2. Стрелочные часы из текстовых строк	3. Вращающийся текст
	

4. Элементы управления типа выключателей I-Pad. Когда движок в левой стороне, он черный и слева от него написано OFF. Когда движок в правой стороне, он зеленый и слева от него написано ON	5. Создать кнопки, текст которых закрашен случайным градиентным цветом
	

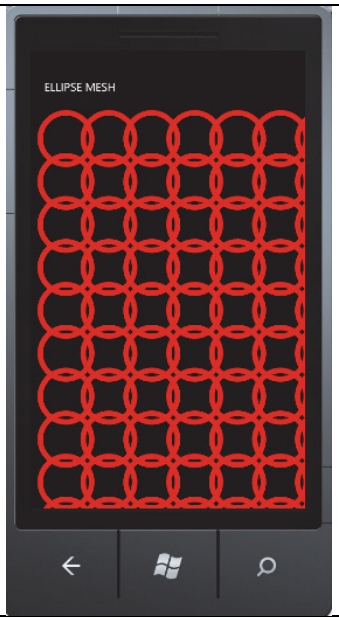
6. Создать многостраничный интерфейс с кнопками:
«+» – добавление страницы;
«←» – переходы на страницы;
корзина – удаление страницы;
«…» – смена цвета темы страницы.
Щелчок мыши на странице ставит на ней графическую точку.

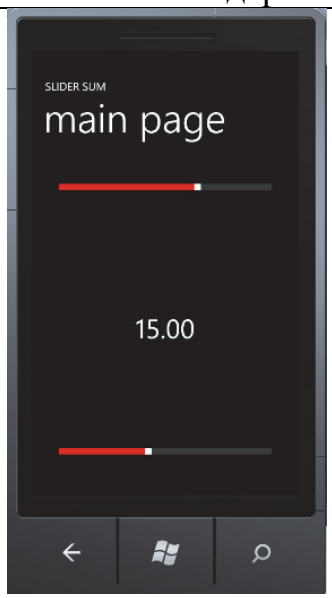
7. Три слайдера (RED, GREEN, BLUE) задают суммарный цвет элемента управления GREED



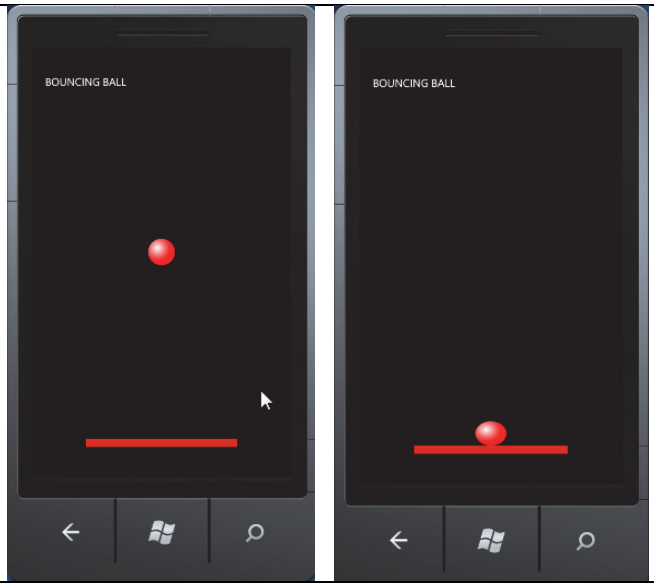
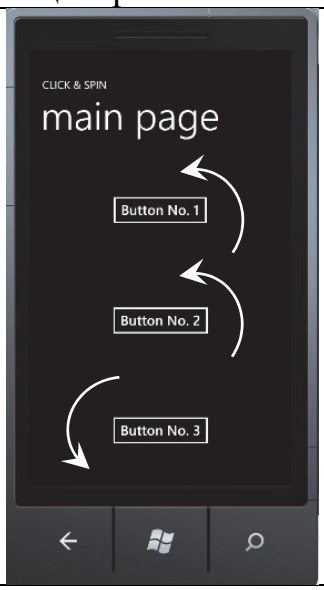
8. Фон с картинкой, на котором овалом выделен какой-то элемент

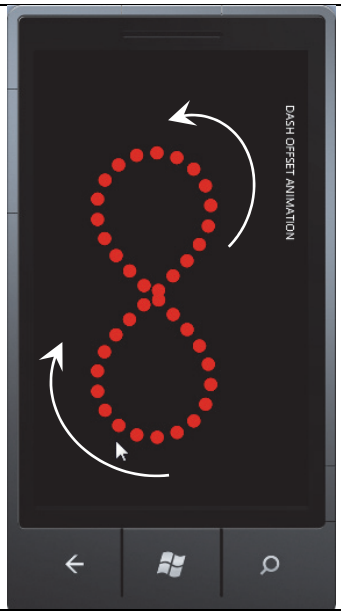
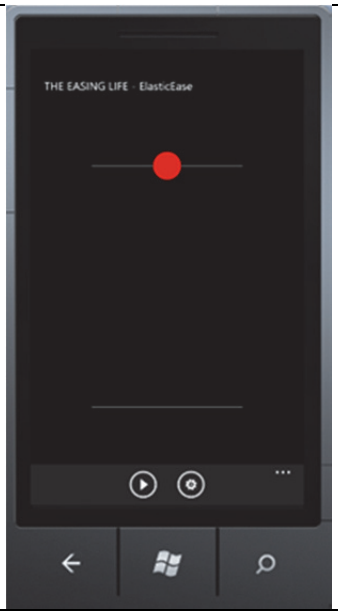
9. Сетка из множества цветных кругов



10. Цепь из нескольких цветных кругов	11. Два слайдера формируют в середине экрана число – результат разности значений слайдеров	12. Часы векторной графики
		

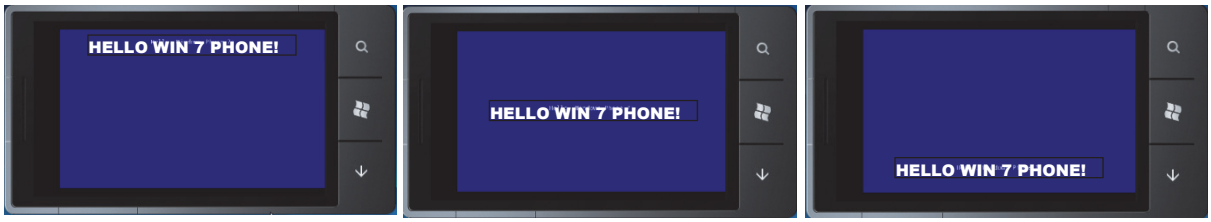
13. Управляемый слайдер. Изменяет свое положение при нажатии на кнопки	14. Выполнение команд векторной графики. При нажатии на кнопку команда отображается в верхнем поле и реализуется в нижнем	15. Вращающаяся спираль
		

16. Прыгающий мячик	17. Вращающиеся кнопки. При нажатии на кнопку, она делает три оборота вокруг своего центра
	

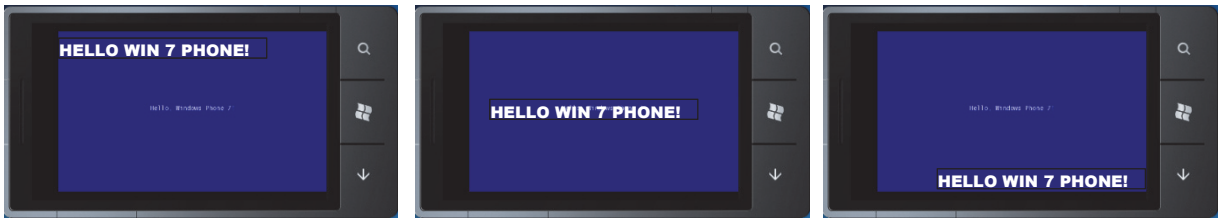
18. Движущаяся восьмерка. Движутся точки, составляющие фигуру	19. Движущиеся концентричные окружности. Возникают из центра, расширяются, потом тускнеют и исчезают	20. Динамичный мячик. При нажатии на кнопку резко перемещается вниз до нижней сетки. При нажатии на другую кнопку – возвращается назад
		

Задание 2 (платформа XNA). По вариантам указано, что должна воспроизводить на экране программа, и приведен примерный вид экрана в момент работы программы (стрелки показывают направление движение объекта, их программировать НЕ НАДО).

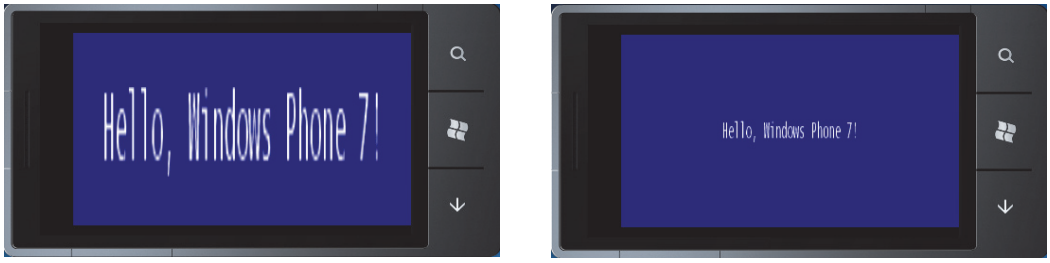
1. Бегающая строка 1. Строка «HELLO WIN 7 PHONE!» плавает вверх-вниз по экрану, отражаясь от границ экрана.



2. Бегающая строка 2. Строка «HELLO WIN 7 PHONE!» плавает по диагонали экрана, отражаясь от углов экрана.

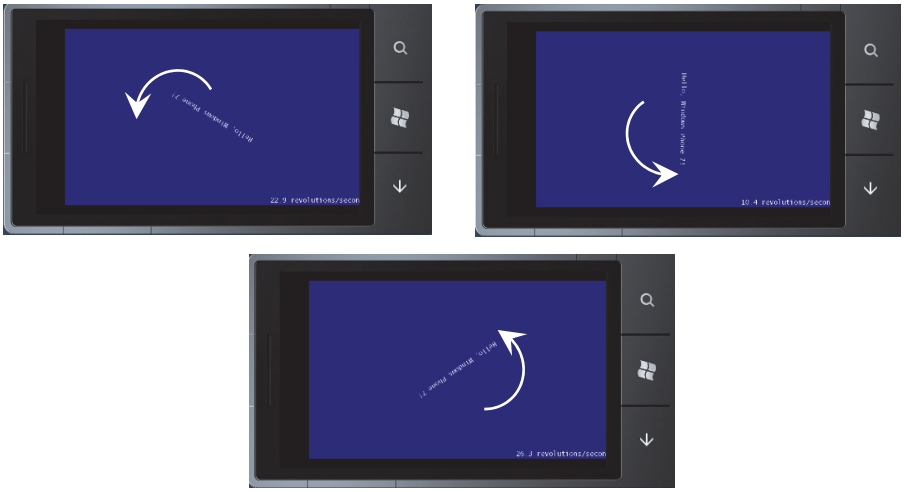


3. Масштабирование. Строка «HELLO WIN 7 PHONE!» приближается и удаляется на экране, создавая эффект масштабирования.

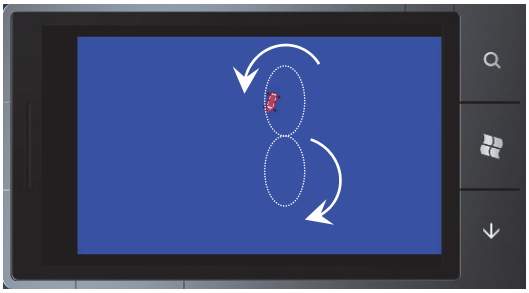


4. Текст «HELLO WIN 7 PHONE!» ползает по кругу вдоль границы экрана.

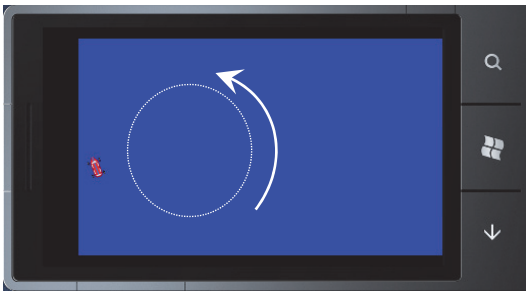
5. Текстовая строка «HELLO WIN 7 PHONE!» вращается в центре экрана с ускорением, потом замедлением, сначала в одну сторону, потом в другую.



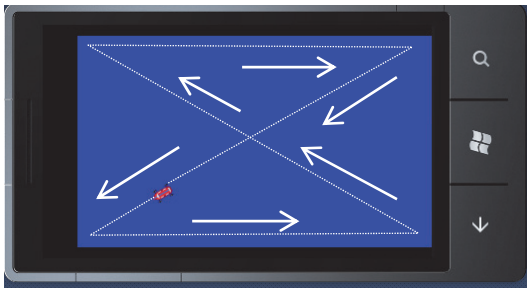
6. Объект выписывает восьмерку на экране.



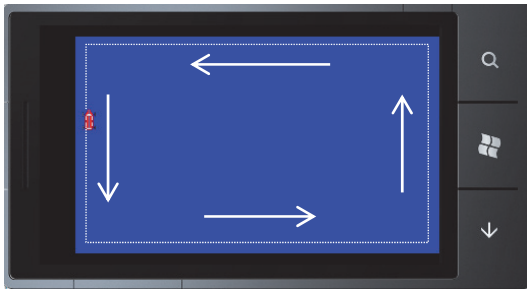
7. Объект выписывает окружность на экране.



8. Объект носится по произвольной траектории на экране.

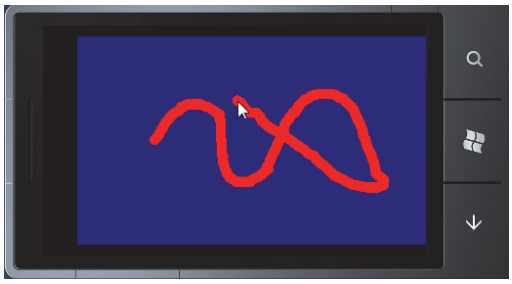


9. Объект движется по прямоугольной границе на экране.

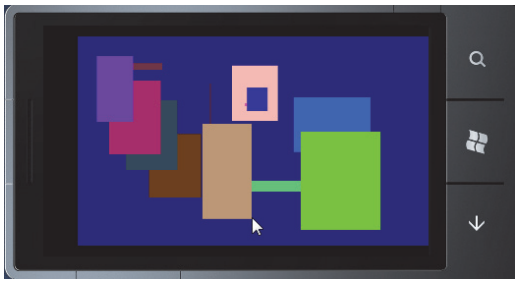


10. Слово «HELLO» при щелчке на нем разлетается на составляющие линии, а потом собирается назад.

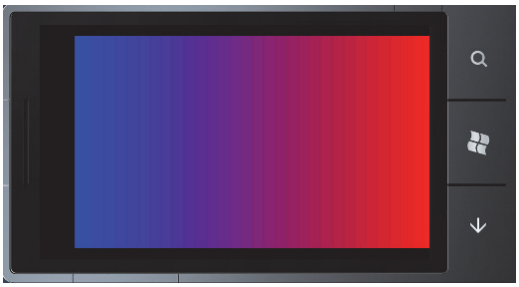
11. Пальцем (мышкой) рисуем на экране произвольную линию.



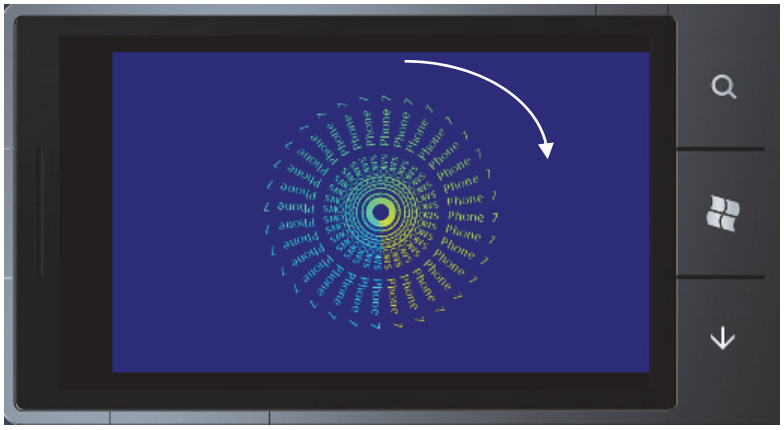
12. Пальцем (мышкой) рисуем на экране произвольные прямоугольники случайного цвета.



13. Программа создает экран с цветной градиентной заливкой.



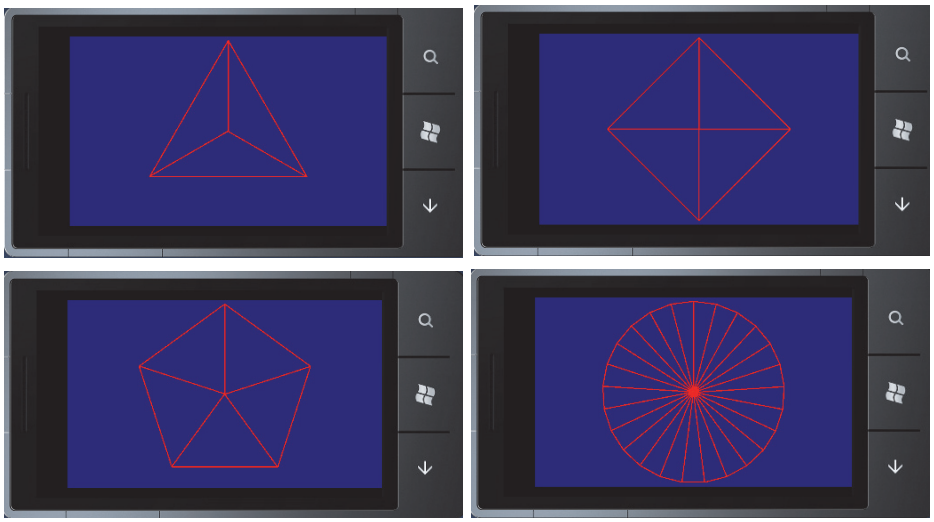
14. Из слова «Phone 7» сделаны крутящиеся круги клонированные (с уменьшением масштаба) друг в друга.



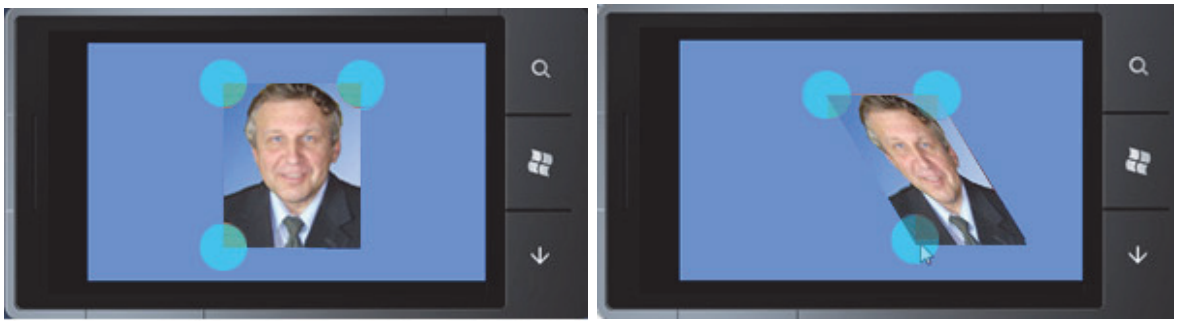
15. Прорисовка случайных прямоугольников со случайным цветом и прозрачностью.

16. Ваша фотография развевается и идет волнами, как флаг.

17. Начинается с треугольников. При каждом щечке мыши (пальца) число сторон многоугольника увеличивается на 1 (TapForPolygon).



18. Разработать программу деформации изображения за углы.



ЗАКЛЮЧЕНИЕ

В представленном пособии собран материал для начального ознакомления программиста с технологиями подготовки программного обеспечения для мобильных устройств с операционной системой Windows Phone. Нельзя сказать, что данная ОС занимает подавляющее пространство на рынке мобильных устройств, однако программный инструментарий и документацию для ознакомления программистов по её использован ию Microsoft предоставляет бесплатно, что весьма важно для студентов и вузов.

Принципы разработки мобильных приложений в Windows Phone весьма схожи с принципами программирования в ведущих системах, таких как Android или I-OS, так что, познакомившись с материалом пособия, достаточно легко перейти к программированию мобильных устройств на любых других платформах.

Удачи Вам в нелегком и высокоинтеллектуальном труде квалифицированного программиста!

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Климов, А. Структура проекта Windows Phone / А. Климов // URL: developer.alexanderklimov.ru/windowsphone.solution-structure.php.

2. Writing your First Windows Phone 7 Application // URL: channel9.msdn.com /Series/Windows-Phone-7-Development-for-Absolute-Beginners/Writing-your-First-Windows-Phone-7-Application.

3. Theme Resources for Windows Phone // URL: http://msdn.microsoft.com/en-us/library/ff769552(VS.92).aspx.

4. XAML Overview // URL: http://msdn.microsoft.com/en-us/library/cc189036(VS.95).aspx.

5. Пейтзольд, Ч. Программируем Windows Phone 7 / Ч. Пейтзольд. – М. : Microsoft Press International, 2011. – 695 с.

6. Климов, А. Программирование для мобильных устройств под управлением Windows Mobile / А. Климов. – СПб. : Питер, 2009. – 332 с.

7. Дерси, Л. Андроид за 24 часа / Л. Дерси, Ш. Кондер. – М. : Рид Групп, 2009. – 463 с.

8. Монахов, В. Язык программирования Java и среда NetBeans / В. Монахов. – СПб. : БХВ-Перербург, 2011. – 703 с.

Учебное издание

Тихомиров Владимир Александрович

РАЗРАБОТКА ПРОСТЕЙШИХ ПРИЛОЖЕНИЙ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ

Учебное пособие

Научный редактор – канд. техн. наук, профессор В. П. Котляров

Редактор Т. Н. Карпова

Подписано в печать 24.09.2013.
Формат 60 × 84 1/16. Бумага 65 г/м^2. Ризограф EZ570E.
Усл. печ. л. 8,12. Уч.-изд. л. 7,76. Тираж 75 экз. Заказ 25782.

Редакционно-издательский отдел
Федерального государственного бюджетного образовательного учреждения
высшего профессионального образования
«Комсомольский-на-Амуре государственный технический университет»
681013, Комсомольск-на-Амуре, пр. Ленина, 27.

Полиграфическая лаборатория
Федерального государственного бюджетного образовательного учреждения
высшего профессионального образования
«Комсомольский-на-Амуре государственный технический университет»
681013, Комсомольск-на-Амуре, пр. Ленина, 27.