



Лабораторный практикум

Предисловие

Дорогие друзья!

Вы приступаете к выполнению лабораторных работ по созданию универсальных приложений на платформе Windows 10. Эти работы позволят вам познакомиться на практике с тем, как разрабатываются приложения, способные работать на широком спектре устройств: от смартфона, планшета, ноутбука или настольного компьютера, до таких экзотических устройств, как Microsoft HoloLens или микрокомпьютер, обслуживающий интернет вещей.

В первых работах рассматриваются самые основы построения приложений с использованием языков XAML/C#, а также показывается, как на основе XAML можно строить адаптивный пользовательский интерфейс, изменяющийся и настраиваясь на то семейство устройств, на котором выполняется приложение. Более сложные лабораторные работы включают в себя взаимодействие с облаком Microsoft Azure, использование рукописного ввода с распознаванием текста или голосового помощника Cortana. В завершение мы поговорим про публикацию приложений в магазине Windows и особенностях монетизации, в т.ч. с помощью рекламы.

Все лабораторные работы с инструкциями и примерами кода расположены в репозитории <http://github.com/evangelism/WinDevWorkshop>. Однако в нескольких лабораторных работах сложно успеть рассмотреть все особенности платформы, поэтому некоторые моменты остались за рамками этого практикума: работа с датчиками, акселерометром, геолокацией, картами, камерой и т.д. Если вам потребуются соответствующие примеры кода для работы с платформой, то рекомендую посмотреть на стандартные примеры в репозитории <https://github.com/Microsoft/Windows-universal-samples>, а для приложений в области интернета вещей - <https://github.com/ms-iot/samples>. Эти примеры очень хорошо использовать в режиме справочника, когда нужно задействовать в своём приложении те или иные возможности платформы. Также много интересных примеров простых универсальных приложений для изучения есть в нашем репозитории примеров <https://github.com/evangelism/HackSamples> – здесь вы найдете некоторые стандартные решения, от простого предсказателя погоды, взаимодействующего с интернетом по REST-протоколу, до небольших игр на XAML/C#.

Я надеюсь, что данный лабораторный практикум вызовет у вас интерес и непреодолимое желание к разработке универсальных приложений, а остальные материалы позволят в увлекательной форме изучить все возможности платформы в деталях. Успехов!

Дмитрий Сошников,
технологический евангелист, Майкрософт Россия
<http://twitter.com/shwars>





Лабораторный практикум

Программа "Hello World" для
универсальной платформы Windows

Октябрь 2015 года



Общие сведения

В Windows 10 впервые появилась универсальная платформа Windows (UWP), которая представляет собой дальнейшее развитие модели Windows Runtime и включает её в ядро ОС Windows 10. Как часть ядра, UWP сейчас предоставляет собой стандартную платформу для приложений, доступную на каждом устройстве, которое запускается под Windows 10. В результате приложения, работающие на UWP на устройстве определенного класса, могут вызывать специфические для данного семейства устройств API, в дополнение к API WinRT, которые являются общими для всех устройств. Вы можете с помощью UWP создать один пакет приложения, который может быть установлен на широкий ассортимент устройств.

В этой работе вы будете использовать инструменты разработки для универсальной платформы Windows (Universal Windows App Development Tools) для создания приложения "Hello World", которое будет запускаться на всех устройствах под Windows 10. Ваше приложение будет отображать информацию об устройстве, на котором оно выполняется, включая семейство устройства и размер текущего окна приложения. Вы также создадите приложение "Hello World" в среде Blend и используете возможности Blend для генерации выборки данных.

Цели

Эта работа научит вас:

- Создавать UWP приложение из пустого шаблона.
- Отображать приветствие в своём приложении
- Находить и отображать данные о семействе устройства
- Динамически отображать размер окна приложения
- Запускать приложение на локальном компьютере
- Запускать эмулятор мобильной платформы
- Запускать приложение на устройствах "интернета вещей"
- Генерировать выборку данных в Blend

Системные требования

Чтобы выполнить этот курс, необходимо обладать следующим набором программных инструментов:

- Microsoft Windows 10
 - Microsoft Visual Studio 2015
-

Дополнительные задачи

Если вы захотите проходить дополнительные задачи в этом курсе, вам понадобятся:

- Устройство Windows 10 Mobile или эмулятор
 - Устройство "интернета вещей", запускающееся под Windows 10 (Raspberry Pi 2)
 - Дисплей, который соединяется с устройством "интернета вещей"
-

Настройка

Вы должны осуществить следующие шаги для подготовки своего компьютера для этого курса:

1. Установите Microsoft Windows 10.
 2. Установите Microsoft Visual Studio 2015. Выберите пользовательскую установку и убедитесь, что в списке дополнительных функций выбраны инструменты разработки для приложений Windows.
 3. Дополнительно: Установите эмулятор Windows 10 Mobile.
 4. Дополнительно: Установите Windows 10 на устройство "интернета вещей".
-

Упражнения

Эта практическая работа включает следующие упражнения:

1. Начало работы с универсальной платформой Windows
 2. Программа "Hello World" на устройствах под Windows
 3. Программа "Hello World" в среде Blend
-

Расчётное время для завершения работы: **от 30 до 45 минут.**

Упражнение 1: Начало работы с универсальной платформой Windows

Если у вас установлены универсальные инструменты разработки приложений Windows, то в Visual Studio будет доступен шаблон для создания приложений UWP. В ходе этого упражнения вы создадите проект из пустого шаблона приложения.

Задача 1 – Создаем пустое приложение под Windows

Мы начнём с создания проекта из шаблона приложения.

1. В новой версии Visual Studio 2015 выберите File (Файл) -> New (Новый) -> Project (Проект), чтобы открыть диалоговое окно New Project (Новый проект). Далее выберите Installed (Установленные) > Templates (Шаблоны) > Visual C# > Windows > Universal, а затем выберите шаблон Blank App приложения Universal Windows.

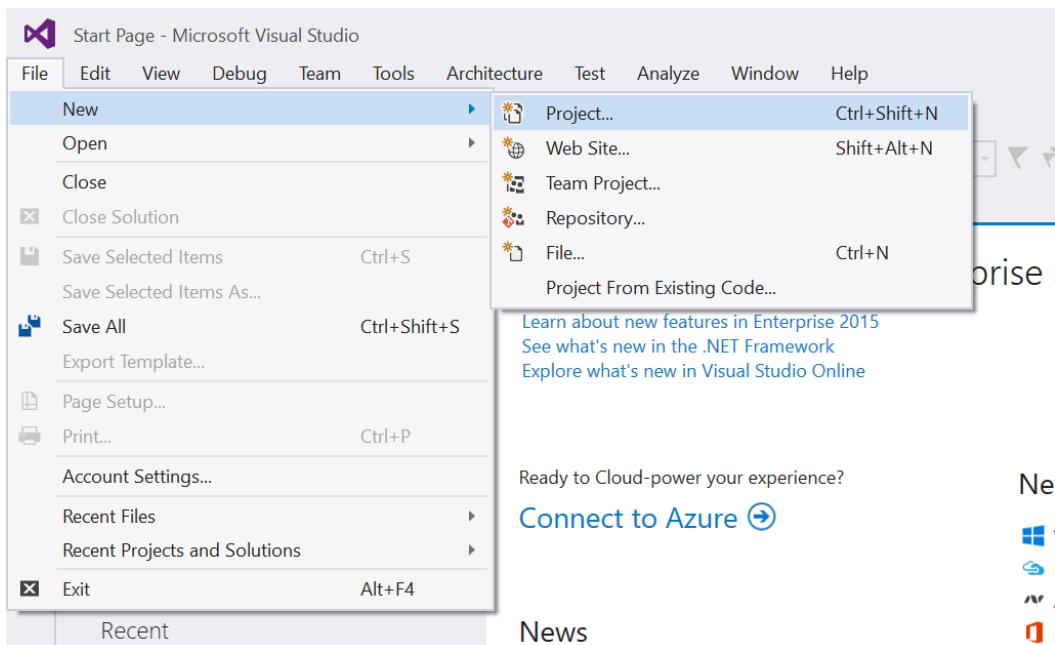


Рисунок 1

Открытие диалогового окна New Project (Новый проект) в Visual Studio 2015.

2. Назовите свой проект **HelloUWP** и выберите местоположение в файловой системе, где вы храните работы этого практикума. Мы создали папку на диске **C** и переименовали ее в **HOL**, именно папку с этим названием вы будете видеть на скриншотах в ходе всего практического курса.
3. Выберите опции **Create new solution (Создать новое решение)** и **Create directory for solution (Создать папку для решения)**. Вы можете снять галочку с опции **Add to source**

control (Добавить контроль исходного кода), если вы не хотите хранить свой код в каком-либо репозитории. Нажмите **OK** для создания проекта.

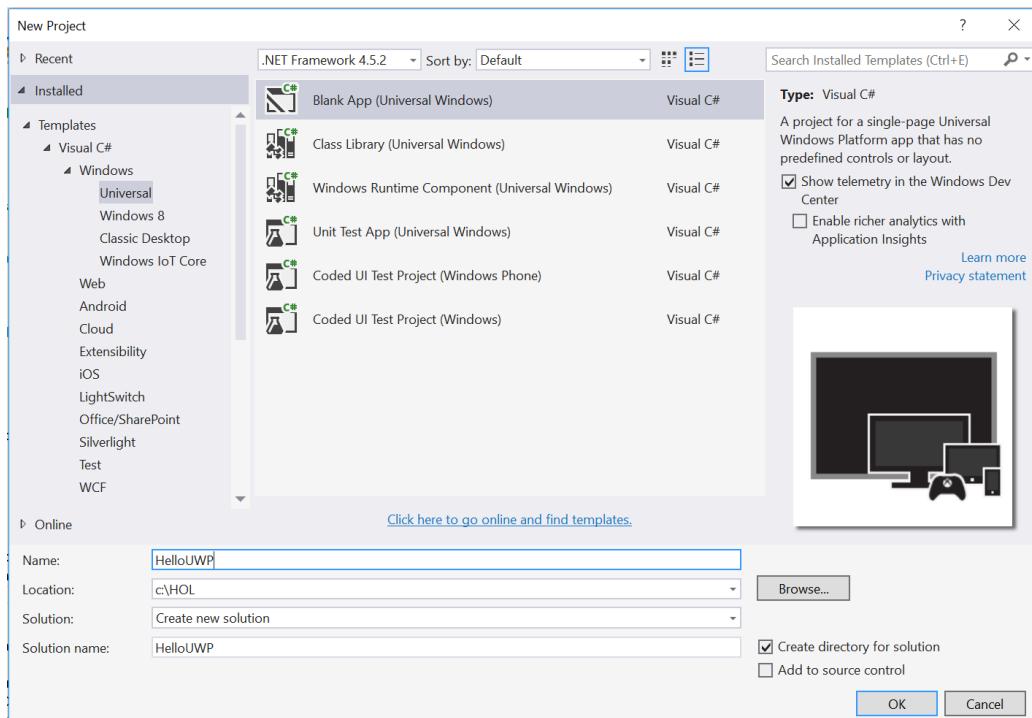


Рисунок 2

Создание нового проекта приложения в Visual Studio 2015.

4. Установите значение **Debug (Отладка)** для параметра **Solution Configuration** (Конфигурации решения), и значение **x86** для параметра **Solution Platform** (Платформа решения). Выберите локальный компьютер в выпадающем списке **Debug Target** (Цели отладки) возле кнопки **Start Debugging** (Запустить отладку).

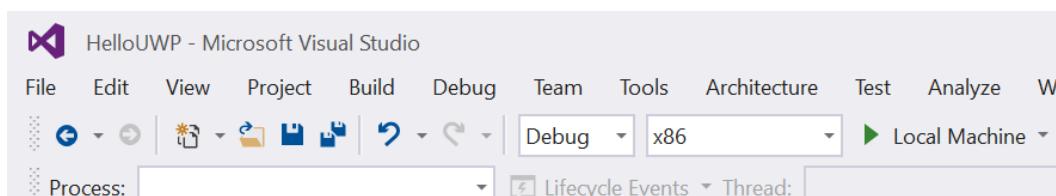


Рисунок 3

Конфигурация приложения для запуска на локальном компьютере.

Примечание: ► Кнопка Start Debugging (Запуск отладки).

5. Нажмите кнопку **Start Debugging** (Запуск отладки), чтобы создать и запустить своё приложение. Вы увидите пустое окно приложения со счётчиком скорости перерисовки экрана, который показывается по умолчанию в режиме отладки.

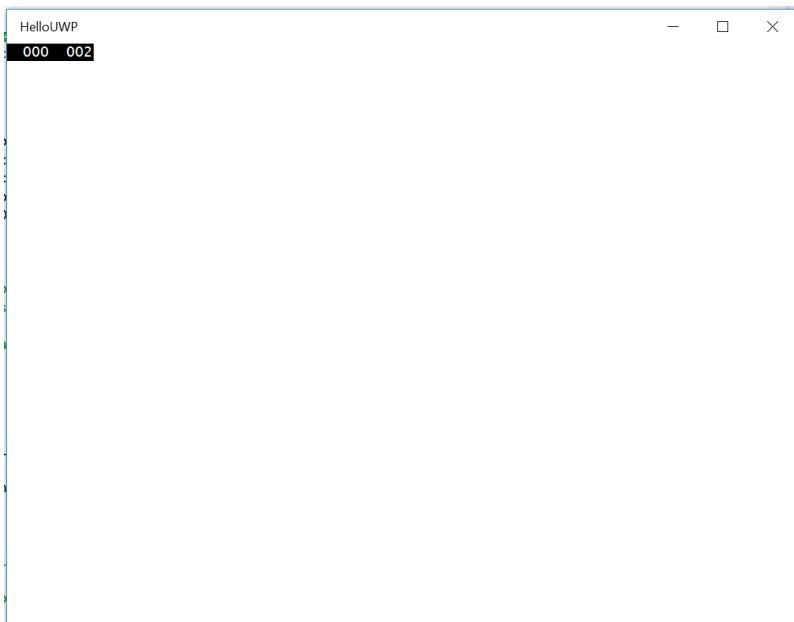


Рисунок 4

Пустое универсальное приложение, запущенное в режиме настольного компьютера.

Примечание: Счётчик скорости обновления кадра является инструментом отладки, который помогает следить за производительностью вашего приложения. Он полезен для приложений, которые требуют интенсивной обработки графики, но не используется для простых приложений, которые вы будете создавать в рамках практикума.

В шаблоне приложения директива препроцессора для активации или отключения счётчика скорости кадра находится в файле **App.xaml.cs**. Счётчик скорости обновления кадра может перекрывать или скрывать содержание приложения, если вы оставите его включенным. Поэтому мы отключим его.

6. Вернитесь в Visual Studio и отключите отладку.
7. Откройте App.xaml.cs. Для отключения счётчика скорости кадра в директиве препроцессора **#if DEBUG**, установите значение **this.DebugSettings.EnableFrameRateCounter = false;**

C#

```
#if DEBUG
    if (System.Diagnostics.Debugger.IsAttached)
    {
        this.DebugSettings.EnableFrameRateCounter = false;
    }
#endif
```

8. Создайте и запустите своё приложение повторно. В этот раз вы увидите пустое окно приложения без счётчика скорости обновления кадров.

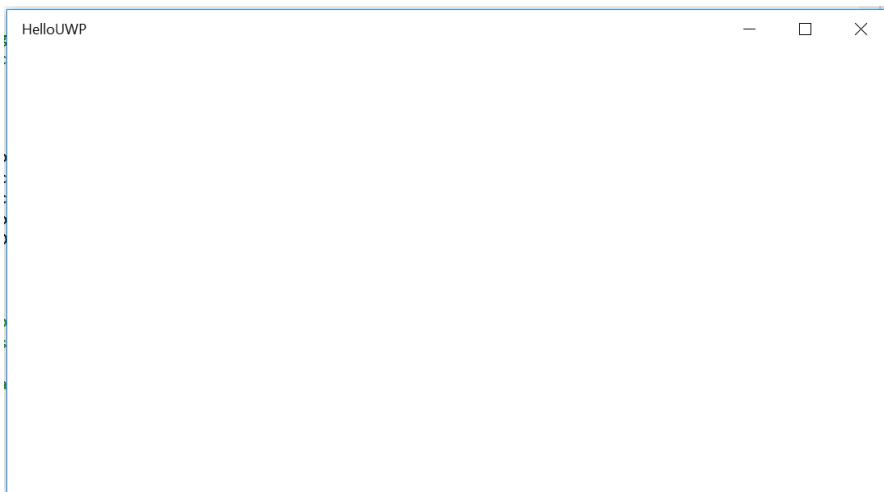


Рисунок 5

Пустое приложение после отключения счётчика скорости обновления кадров.

9. Отключите отладку и вернитесь в Visual Studio.

Задача 2 – Изучаем шаблон

Шаблон пустого приложения обеспечивает базовую структуру, которая вам будет нужна при создании приложения UWP. Давайте рассмотрим, как он устроен.

1. **App.xaml.cs** – точка входа для вашего приложения, этот файл содержит методы обработки активации и приостановления работы приложения. В этом шаблоне конструктор запускает инструмент Application Insights, вызывает функцию `InitializeComponent()` и изучает список отложенных событий. В соответствующем файле **App.xaml**, вы можете объявить ресурсы, которые будут совместно использованы в приложении.

Функция **OnLaunched** в App.xaml.cs настраивает содержимое страницы (Frame) как навигационный контекст и подключает его к текущему окну (Window). Вы можете добавить код для загрузки состояния предыдущего исполнения приложения (это важно, чтобы приложение могло восстановить своё состояние после приостановки). Если состояние не было сохранено, произойдет переход на страницу **MainPage**.

Вы также можете заметить, что функция **OnLaunched()** включает код для активации счётчика скорости кадра при отладке, условно включенного в директиву препроцессора `#if DEBUG`. Дополнительную информацию о некоторых параметрах отладки вы можете получить по ссылке:

<https://msdn.microsoft.com/en-us/library/windows.ui.xaml.debugsettings.aspx>.

Функция **Window.Current.Activate()** является важной частью процесса активации и требуется для всех сценариев активации приложения, включая второстепенные окна.

2. **MainPage.xaml** и **MainPage.xaml.cs** совместно определяют вид страницы **MainPage**, унаследованной от базового класса **Page**. Вы можете добавить элементы

пользовательского интерфейса в **MainPage.xaml** и обработчики логики и событий в код **MainPage.xaml.cs**. Вы сможете больше узнать о MainPage в следующем упражнении.

3. Папка **Assets (Ресурсы)** содержит ресурсы по умолчанию для приложения. В шаблоне пустого приложения представлен минимальный набор ресурсов. Если вы захотите задать собственные или дополнительные ресурсы, вы можете добавить их в эту папку, кликнув правой кнопкой мыши на имени папки в Solution Explorer и выбрав **Add (Добавить) > Existing Item (Существующий компонент)**. Как только ресурсы были добавлены, вы можете посетить манифест приложения, чтобы зарегистрировать свои новые ресурсы.
4. Файл **Package.appxmanifest** описывает свойства вашего приложения. В этот файл вы можете добавить разрешение на использование определенных функций и расширений – например, функцию доступа устройства к микрофону. В нем также содержится ссылка на экран заставки, "плитку" по умолчанию и иконка приложения. **Package.appxmanifest** открывается по умолчанию в **Manifest editor (Редактор манифеста)**, который позволяет редактировать все полезные свойства приложения в режиме диалога. На вкладке **Visual Assets (Визуальные ресурсы)**, например, редактор проектов указывает, какие ресурсы логотипов рекомендованы к использованию. Вы также можете просматривать манифест приложений в виде XML файла: чтобы сделать это, кликните правой кнопкой мыши на файле в Solution Explorer и выберите **View Code (Просмотр кода)**.

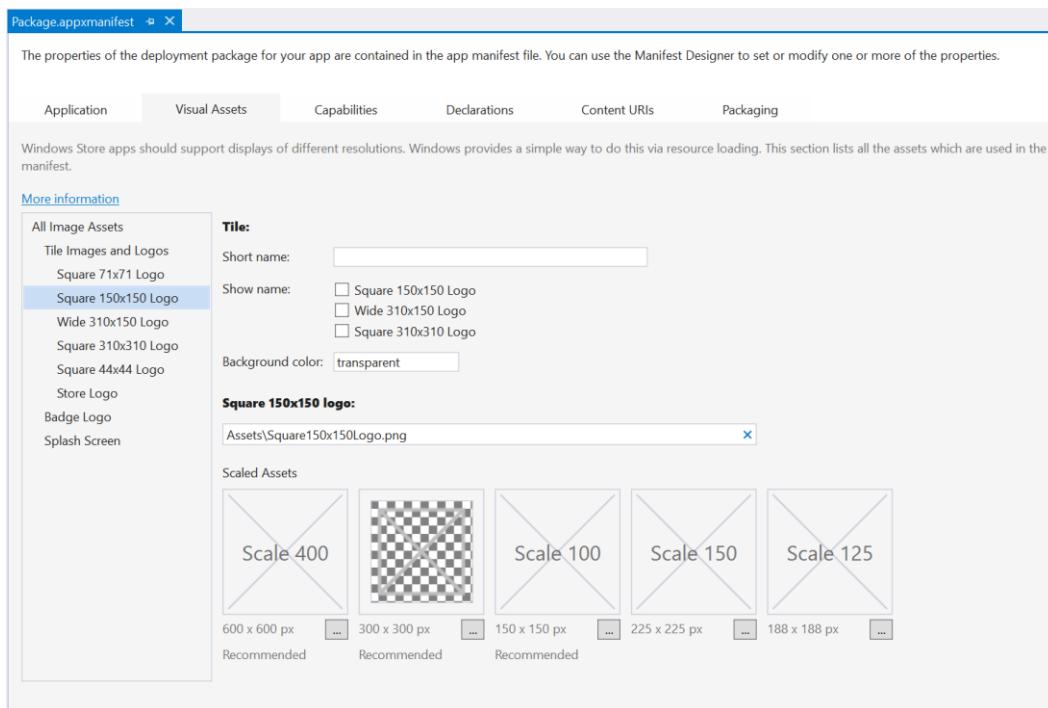


Рисунок 6

В манифесте приложений указывается, какие иконки рекомендованы к использованию.

- Папка **References (Ссылки)** содержит список SDK, библиотек и компонентов, которые используются в вашем приложении. Если вы хотите добавить ссылку, кликните правой кнопкой мыши на папке References (Ссылки) и выберите **Add Reference (Добавить Ссылку)**, чтобы открыть **Reference Manager (Менеджер ссылок)**.
- Файл **project.json** в корневом каталоге вашего проекта является новым для платформы UWP и упрощает управление пакетами NuGet (пакетами внешних подключаемых библиотек). Как правило, лучше не редактировать этот файл напрямую, а использовать средства менеджера пакетов NuGet для просмотра и управления пакетами для использования в своём проекте. Дополнительную информацию см. по ссылке <http://docs.nuget.org/consume/ProjectJson-Intro>.
- Файл **Properties** позволяет вам управлять событиями сборок и конфигурацией отладки. По мере дальнейшего развития платформы Windows 10, вам может потребоваться использовать вкладку **Application (Приложение)**, чтобы обновить оптимальную версию и минимальную версию платформы UWP и Windows 10, которые ваше приложение будет поддерживать.

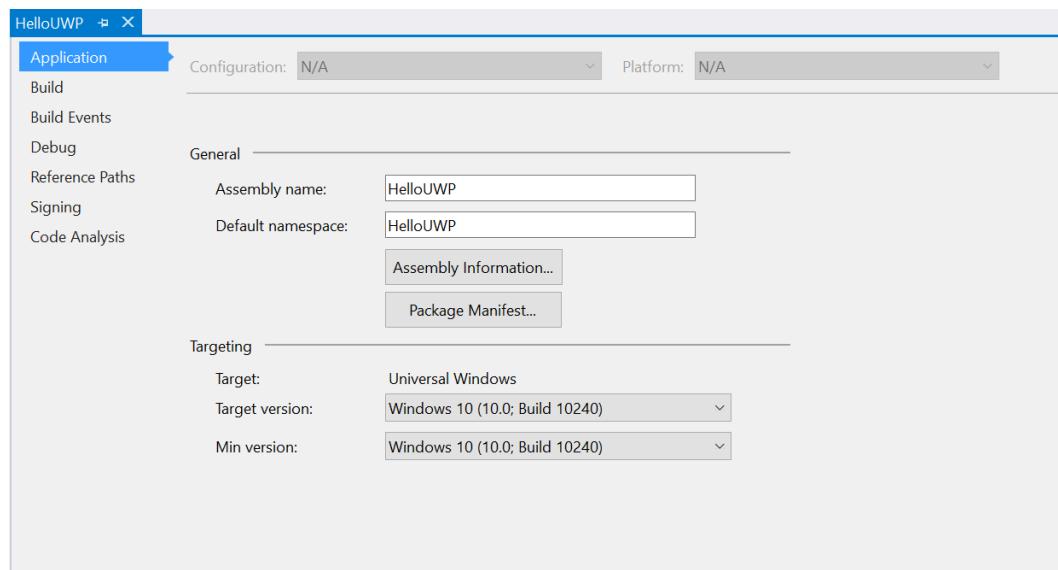


Рисунок 7

Изменение поддерживаемой целевой и минимальной версии Windows 10 в свойствах вашего приложения.

Упражнение 2: Программа "Hello World" на устройствах под Windows

Приложения Windows сейчас являются универсальными для всех устройств, что означает, что они могут работать в полноэкранном режиме на мобильных телефонах и планшетах, а также в оконном режиме на персональных компьютерах. В этом упражнении вы создадите приложение "Hello World", которое отображает приветствие и информацию об устройствах под Windows 10, включая настольные компьютеры, планшеты, смартфоны и устройства "интернета вещей".

Задача 1 – Отображение приветствия

Первая задача состоит в том, чтобы открыть решение HelloUWP, которое вы создали в предыдущем упражнении.

1. Откройте директорию, в которой вы сохранили своё приложение **HelloUWP** в Упражнении 1. Откройте файл **HelloUWP.sln** в Visual Studio 2015. Также вы можете открыть решение, используя диалоговое окно **Open Project (Открыть Проект)** Visual Studio.
2. Откройте **MainPage.xaml** в Solution Explorer. Добавьте элемент **StackPanel**, содержащий текстовый блок **TextBlock** к существующему файлу разметки.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel>
        <TextBlock />
    </StackPanel>
</Grid>
```

Примечание: Элемент **StackPanel** является контейнерным элементом управления, который расставляет дочерние элементы вертикально или горизонтально. Есть некоторое количество других контейнеров, которые могут помочь в компоновке вашего пользовательского интерфейса, включая **Grid (Сетка)**, **Canvas (Холст)** и **RelativePanel (Дополнительная панель)**. Мы опишем элементы Grid (Сетка) и RelativePanel (Дополнительная панель) позже в рамках курса **Создание адаптивного пользовательского интерфейса**.

3. Добавление текста и дизайна в ваш блок **TextBlock**.

XAML

```
<TextBlock Text="Hello UWP World!" FontSize="24" FontWeight="Light"
Margin="12" />
```

Примечание: Вы можете заметить, что во всех практикумах значения полей, размера и положения всегда кратны 4. Приложения UWP используют эффективные пиксели на 4x4-пиксельной сетке, чтобы гарантировать высокое качество визуального опыта в устройствах, использующих экраны с различными разрешениями и масштабами. Чтобы гарантировать правильный вид для вашего приложения, размещайте элементы пользовательского интерфейса в 4x4-пиксельной сетке. Эта сетка не применяется к тексту, так что вы можете продолжить использовать любой размер шрифта, который пожелаете.

Более подробную информацию о эффективных пикселях и соответствующем дизайне приложений см. по ссылке <https://msdn.microsoft.com/en-us/library/windows/apps/Dn958435.aspx>

4. Соберите и запустите своё приложение на **локальном компьютере**.

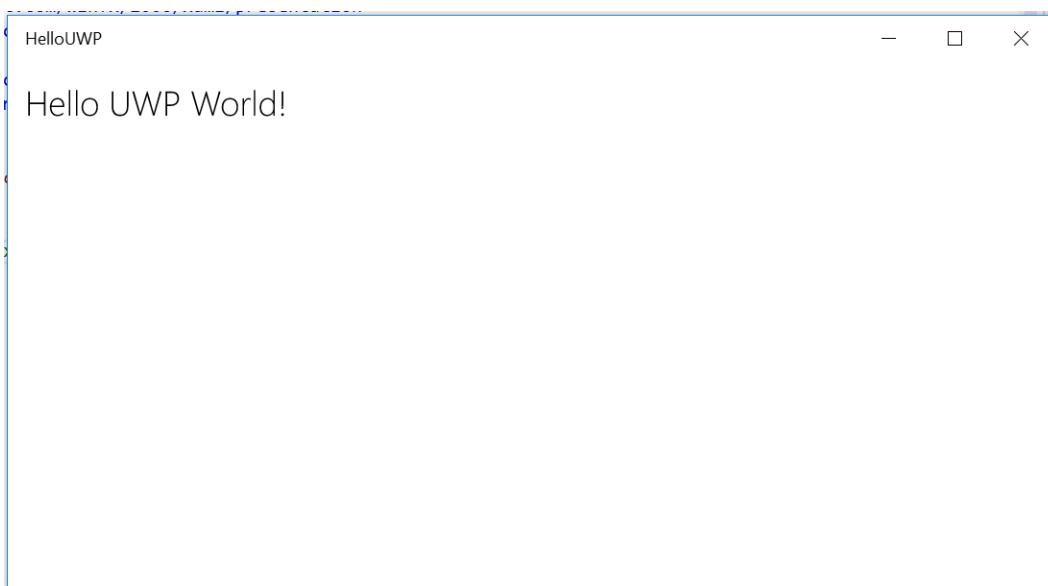


Рисунок 8

Приложение HelloUWP отображает приветствие.

5. Отключите отладку и вернитесь в Visual Studio. Позднее в этом упражнении вы научитесь запускать приложение в мобильном эмуляторе и на устройствах «интернета вещей». Необходимо добавить дополнительные элементы управления для отображения информации, которая будет указывать тип устройства и размеры экрана.

Задача 2 – Определение семейства устройства

Сейчас, когда одно приложение UWP может выполняться на всех устройствах под Windows 10, было бы полезно иметь возможность определять отдельные семейства устройств, которые совместно используют специфические функции API. Например, вы можете задать некоторые функции, которые будут доступны только на мобильных устройствах, и скрыть их на настольной версии приложения. Для этого вводят понятие **семейства устройств**, т.е. набора устройств, для которых имеются те или иные разновидности API. В этой задаче вы обнаружите и покажете семейство устройства, на котором выполняется ваше приложение.

1. Откройте **MainPage.xaml.cs**. Добавьте поле, чтобы запомнить информацию о семействе устройств.

C#

```
public string DeviceFamily = "Device Family " +  
    AnalyticsInfo.VersionInfo.DeviceFamily;  
  
public MainPage()  
{  
    this.InitializeComponent();  
}
```

Примечание: С приходом Windows 10 меняется взгляд на таргетирование приложений.

Согласно новой концептуальной модели, приложение ориентируется на группу родственных устройств, называемых семейством устройств. Семейство устройств – набор API, собранных вместе и наделённых названием и номером версии. Ваше приложение может запускаться на множестве семейств устройств через адаптивный код и иметь специфические уникальные функции для каждого семейства устройств.

Более подробную информацию о семействах устройств в приложениях UWP см. по ссылке <https://msdn.microsoft.com/en-us/library/windows/apps/dn894631.aspx>

2. Добавьте пространство имен **Windows.System.Profile** в файле MainPage.

C#

```
using Windows.System.Profile;
```

3. В MainPage.xaml свяжите **TextBlock** с полем **DeviceFamily**.

Примечание: Связывание данных обеспечивает простой способ взаимодействия вашего UI с данными. Отображение данных отделяется от бизнес-логики приложения, а связывание позволяет осуществлять привязку свойств элементов UI с полями данных. Связывание может использоваться для одноразового отображения данных (One Time), обновления UI при изменении данных (One Way) или захвата данных, вводимых пользователями (Two Way).

{x:Bind} является новым расширением разметки в Windows 10 UWP и предлагает компилируемую альтернативу классическому связыванию {Binding}. Более подробную информации о x:Bind см. по ссылке <https://msdn.microsoft.com/en-us/library/windows/apps/Mt204783.aspx>

XAML

```
<StackPanel>  
    <TextBlock Text="Hello UWP World!" FontSize="24" FontWeight="Light"  
    Margin="12" />  
    <TextBlock Text="{x:Bind DeviceFamily}" Margin="12,0,0,0" />  
</StackPanel>
```

Примечание: StackPanel по умолчанию располагает элементы вертикально. Вы можете указать горизонтальную ориентацию с помощью атрибута Orientation="Horizontal".

- Соберите и запустите своё приложение на **локальном компьютере**. Вы увидите отображение данных о семействе устройства под приветствием.

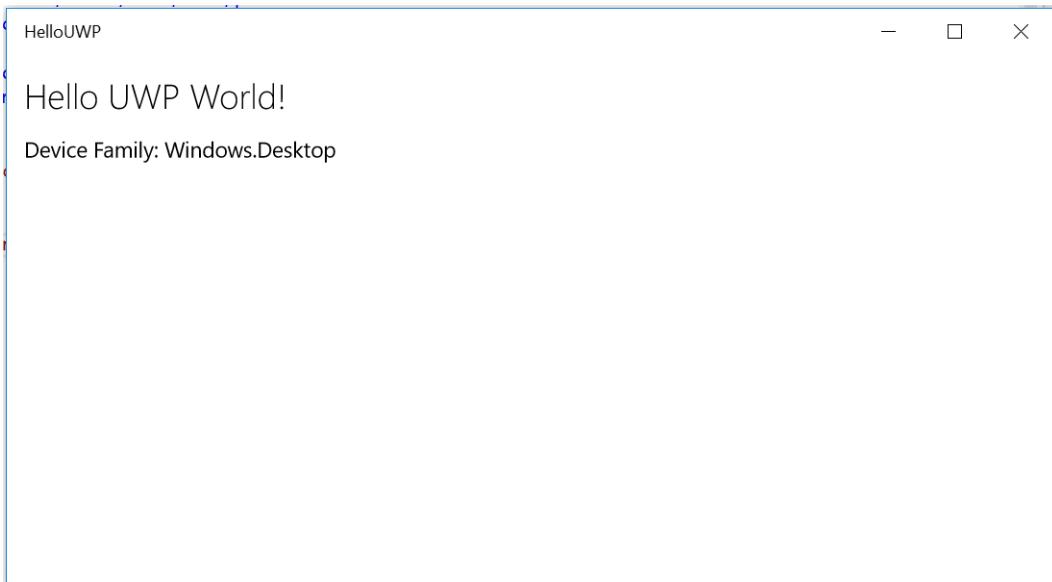


Рисунок 9

Приложение HelloUWP отображает семейство устройств для устройства, на котором оно выполняется.

Примечание: Если вы занимаетесь разработкой в настольной среде и используете локальный компьютер, ваше семейство устройств отобразится как Windows.Desktop. Список других семейств устройств, доступных в Universal Device Family включает Mobile, Xbox, IoT, IoT headless и другие.

- Отключите отладку и вернитесь в Visual Studio.

Задача 3 – Динамическое отображение размеров окна приложения

Очередное свойство, которое любопытно сравнивать на разных устройствах – размер окна приложения, который выражен в эффективных пикселях. Универсальные приложения Windows могут выполняться в полноэкранном режиме на мобильных устройствах и планшетах, а также в оконном режиме на рабочих станциях. В этой задаче вы создадите код для считывания данных и отображения размерности окна приложения. Данные будут обновляться, когда вы измените размер окна приложения.

- Добавьте блок **TextBlock** в элемент управления **StackPanel** в файле MainPage.xaml и привяжите его к свойству **Dimensions (Размеры)**. Вы создадите свойство **Dimensions (Размеры)** в следующих шагах.

XAML

```
<StackPanel>
    <TextBlock Text="Hello UWP World!" FontSize="24" FontWeight="Light"
Margin="12" />
    <TextBlock Text="{x:Bind DeviceFamily}" Margin="12,12,0,0" />
```

```
<TextBlock Text="{x:Bind Dimensions, Mode=TwoWay}" Margin="12,0,0,0" />
</StackPanel>
```

1. Откройте **MainPage.xaml.cs** и создайте свойства для размеров, текущей ширины и текущей высоты.

C#

```
public string DeviceFamily = "Device Family " +
    AnalyticsInfo.VersionInfo.DeviceFamily;

public string Dimensions { get; set; } = "Initial Value";
private double _currentWidth;
private double _currentHeight;
```

2. Добавьте обработчик события **MainPage_SizeChanged** в код ниже. Когда событие **SizeChanged** запустится, этот обработчик определит новую ширину и высоту окна и округлит их до целочисленных значений для отображения.

C#

```
public MainPage()
{
    this.InitializeComponent();
    this.SizeChanged += MainPage_SizeChanged;
}

private void MainPage_SizeChanged(object sender, SizeChangedEventArgs e)
{
    _currentWidth = Window.Current.Bounds.Width;
    _currentHeight = Window.Current.Bounds.Height;
    Dimensions = string.Format("Current Window Size: {0} x {1}",
        (int)_currentWidth, (int)_currentHeight);
}
```

3. Чтобы ваше скомпилированная привязка работала со свойством **Dimensions** (**Размеры**), вы должны вызвать событие **PropertyChanged**. Используйте интерфейс **INotifyPropertyChanged**.

C#

```
public sealed partial class MainPage : Page, INotifyPropertyChanged
{
    public string DeviceFamily = "Device Family " +
        AnalyticsInfo.VersionInfo.DeviceFamily;

    public event PropertyChangedEventHandler PropertyChanged;

    public string Dimensions { get; set; } = "Initial Value";
    private double _currentWidth;
    private double _currentHeight;

    public MainPage()
```

```

{
    this.InitializeComponent();
    this.SizeChanged += MainPage_SizeChanged;
}

private void MainPage_SizeChanged(object sender, SizeChangedEventArgs e)
{
    _currentWidth = Window.Current.Bounds.Width;
    _currentHeight = Window.Current.Bounds.Height;
    Dimensions = string.Format("Current Window Size: {0} x {1}",
(int)_currentWidth, (int)_currentHeight);

    if (PropertyChanged != null)
    {
        PropertyChanged(this,
            new
PropertyChangedEventArgs(nameof(Dimensions)));
    }
}

```

- Добавьте пространство имен **System.ComponentModel** в код MainPage для завершения реализации использования интерфейса **INotifyPropertyChanged**.

C#

```
using System.ComponentModel;
```

- Скомпилируйте и запустите своё приложение. Текстовый блок **Dimensions (Размеры)** отобразит исходный размер окна и обновится при изменении размера окна.

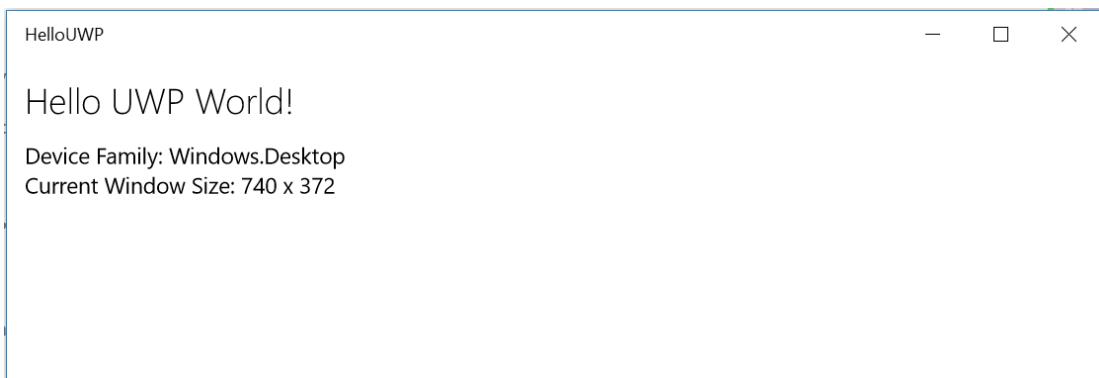


Рисунок 10

Размер обновляется, поскольку размер окна изменён.

- Остановите отладку и вернитесь в Visual Studio.

Задача 4 – Запуск эмулятора мобильной платформы (опционально)

Приложение HelloUWP, которое вы создали в предыдущей задаче, уже настроено для запуска на любом устройстве Windows 10. До появления UWP приложений разработчикам приходилось бы создавать отдельный проект для приложения, работающего на мобильных

устройствах Windows Phone. Преимущество UWP состоит в том, что один проект будет работать на всех устройствах под Windows 10. В этой задаче мы будем использовать эмулятор мобильной платформы Windows 10 Mobile, чтобы протестировать работу приложения HelloUWP на мобильном устройстве.

Примечание: Эмулятор Microsoft мобильной платформы на Windows 10 является частью SDK для Windows 10, и может быть установлен во время процесса установки Visual Studio 2015. Для работы эмулятора требуется Windows 8.1 (x64) Professional edition и выше или Windows 10 Pro или Enterprise (x64). Кроме того, вам понадобится процессор, который поддерживает технологию Client Hyper-V и Second Level Address Translation (SLAT).

Посетите [страницу системных требований для Visual Studio 2015](#) для получения дополнительной информации.

1. Измените значение настройки **Debug Target (Цель Отладки)**, чтобы использовать один из эмуляторов, предоставляемых Microsoft Emulator for Windows 10 Mobile. Мы решили использовать опцию **Mobile Emulator 10.0.10240.0 720p 5 inch 1GB**.

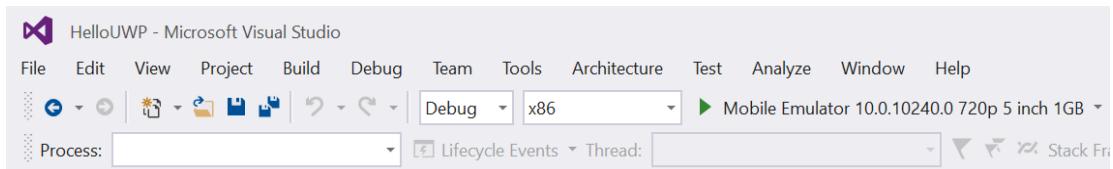


Рисунок 11

В качестве цели отладки установлен эмулятор мобильной платформы.

2. Скомпилируйте и запустите своё приложение. Приложение HelloUWP отобразится на вашем мобильном устройстве со значением семейства устройств Windows.Mobile. Размер окна будет отражать размеры экрана мобильного устройства (за исключением строки состояния), потому что приложения мобильного всегда работают в полноэкранном режиме.



Рисунок 12

HelloUWP запущена в эмуляторе мобильной платформы.

3. Остановите отладку и вернитесь в Visual Studio.

Задача 5 – Запуск приложения на устройствах "интернета вещей" (дополнительно)

Семейство универсальных устройств Windows 10 включает устройства "интернета вещей" (IoT) в дополнение к настольным компьютерам, мобильным устройствам и Xbox. Устройства IoT обычно включают датчики и средства подключения для записи и обмена данными с другими устройствами. Вы можете выполнить эту задачу для запуска своего приложения HelloUWP на устройстве IoT, если вы имеете его в своем распоряжении. Для этой задачи мы будем использовать Raspberry Pi 2, работающее под Windows 10. Если вы не имеете доступа к устройству IoT, вы можете пропустить эту задачу.

Примечание: Пройдите по ссылке <http://www.windowsondevices.com/> для загрузки инструкций и инструментов для установки Windows 10 на устройство IoT. Список поддерживаемых устройств включает Raspberry Pi, Minnowboard Max, Galileo и Arduino.

Семейство универсальных устройств также включает семейство IoT устройств без собственных средств управления. Подобные устройства IoT работают без графического интерфейса пользователя. Наше приложение HelloUWP отображает информацию визуально, так что мы будем работать с семейством обычных IoT устройств вместо семейства IoT устройств без графического интерфейса (headless).

1. Убедитесь, что ваше устройство IoT имеет питание, и загрузите его под Windows 10. Подключите устройство к монитору.
2. Подключите устройство IoT к своей локальной сети. Вы можете подключиться напрямую посредством Ethernet или Wi-Fi или использовать совместный доступ в Интернет (ICS), чтобы подключиться через свой компьютер разработчика.

Примечание: Дополнительную информацию по подключению своего устройства IoT к своей локальной сети см. по ссылке <https://ms-iot.github.io/content/en-US/win10/ConnectToDevice.htm>.

3. Задайте локальный IP-адрес для своего устройства IoT. Raspberry Pi 2, запускающееся под Windows 10, по умолчанию для устройств IoT отображает имя устройства и IP-адрес на начальном экране.
4. Используйте PowerShell, чтобы подключить и сконфигурировать своё устройство IoT под Windows 10, как описано по ссылке: <http://ms-iot.github.io/content/en-US/win10/samples/PowerShell.htm> Во многих случаях дополнительные шаги конфигурации не нужны.
5. Откройте проект HelloUWP в Visual Studio на своем рабочем компьютере. Настройте Solution Platform (Платформа решения) на **ARM**.
6. Дважды кликните по **Properties (Свойства)** проекта в Solution Explorer. Вы также можете правой кнопкой мыши кликнуть на названии проекта и выбрать пункт **Properties (Свойства)** из меню.

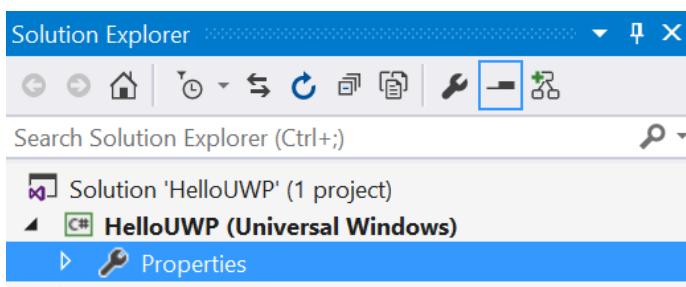


Рисунок 13

Откройте свойства проекта в Solution Explorer.

- Перейдите на вкладку свойств **Debug (Отладка)**. Используйте выпадающий список **Target device (Целевое устройство)** для выбора **Remote Machine (Удалённый компьютер)** в качестве цели отладки.
- Отмените выбор опции **Use authentication (Аутентификация при использовании)**.
- Выведите имя устройства IoT или локальный IP-адрес в поле **Remote machine (Удалённый компьютер)**.
- Сохраните файл свойств HelloUWP.

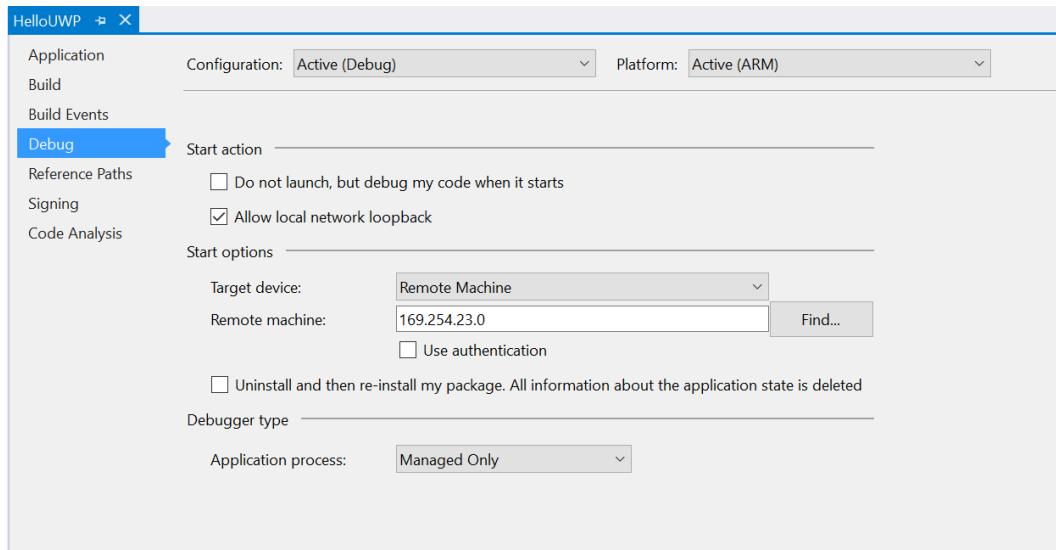


Рисунок 14

Сконфигурируйте свойства проекта для запуска на удалённом компьютере в локальной сети.

- Скомпилируйте и запустите своё приложение. Вы увидите, что оно запустилось на дисплее, подключённом к вашему устройству IoT. Семейство Устройств будет отображаться как **Windows.IoT**.
- Отключите отладку и вернитесь в Visual Studio.

Упражнение 3: Программа "Hello World" в среде Blend

Blend для Visual Studio 2015 был перепроектирован, чтобы облегчить проектирование и создание приложений. В дополнение к пользовательскому интерфейсу и улучшениям производственного процесса, теперь поддерживаются XAML IntelliSense и базовые инструменты для отладки приложений. В этом упражнении мы будем использовать Blend, чтобы создать новый проект и заполнить её выборочными данными.

Задача 1 – Создаем новый проект в Blend

Blend даёт возможность начать создание нового приложения без открытия Visual Studio.

1. Запустите Blend для Visual Studio 2015. Используйте панель **Start Page (Стартовая страница)** или меню **File (Файл) > New (Новый) > Project (Проект)**, чтобы открыть диалоговое окно **New Project (Новый Проект)**.
2. В меню **Templates (Шаблоны) -> Visual C# (Визуальный C#)**, выберите шаблон **Blank App (Пустое приложение)**.
3. Присвойте своему проекту имя **HelloBlend** и сохраните его в папку, в которой вы сохраняете свои проекты практикума

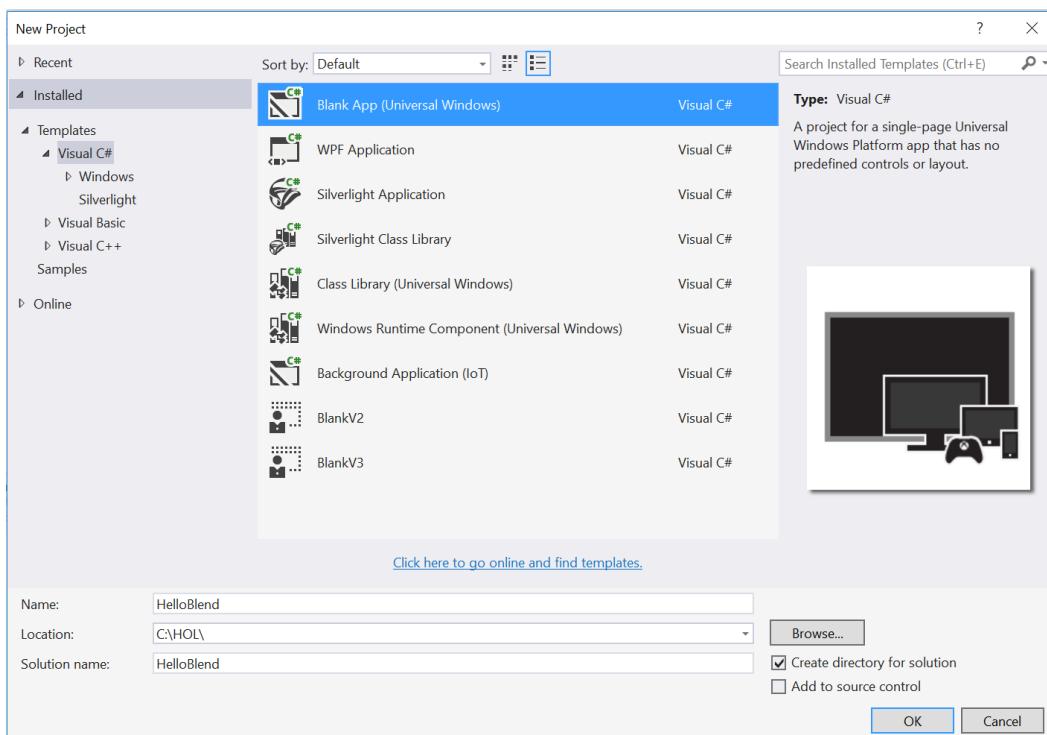


Рисунок 15

Создание проекта HelloBlend в Blend.

Примечание: Хотя вы создали этот проект в Blend, он является полностью действующим проектом под Visual Studio.

- Blend отображает дизайн для MainPage.xaml в вашем новом проекте. Шаблон пустого приложения обеспечивает вам пустой холст для начала работы.

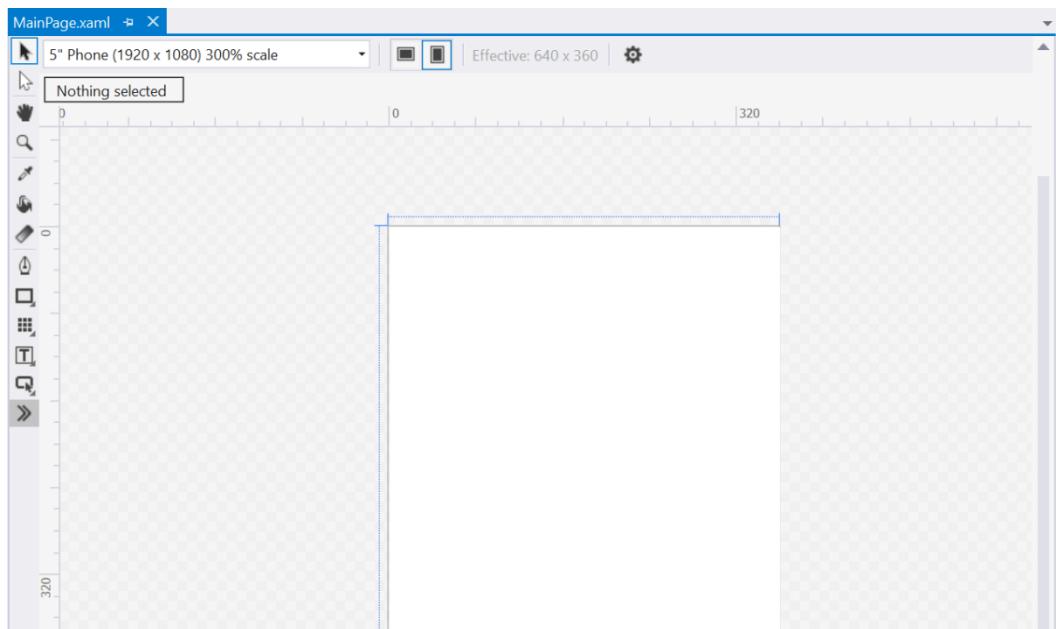


Рисунок 16
MainPage проекта пустого шаблона HelloBlend.

Задача 2 – Генерируем данные выборки

Данные выборки могут помочь начать разработку вашего приложения, ориентированного на работу с данными. Вы можете быстро сгенерировать выборочные данные в Blend через панель **Data (Данные)**. В этой задаче вы сгенерируете выборочные данные и будете просматривать их в запущенном приложении.

- В открытом в Blend проекте **HelloBlend** перейдите в панель **Data (Данные)**. В макете окна по умолчанию панель Data (Данные) совместно использует часть интерфейса с Solution Explorer.

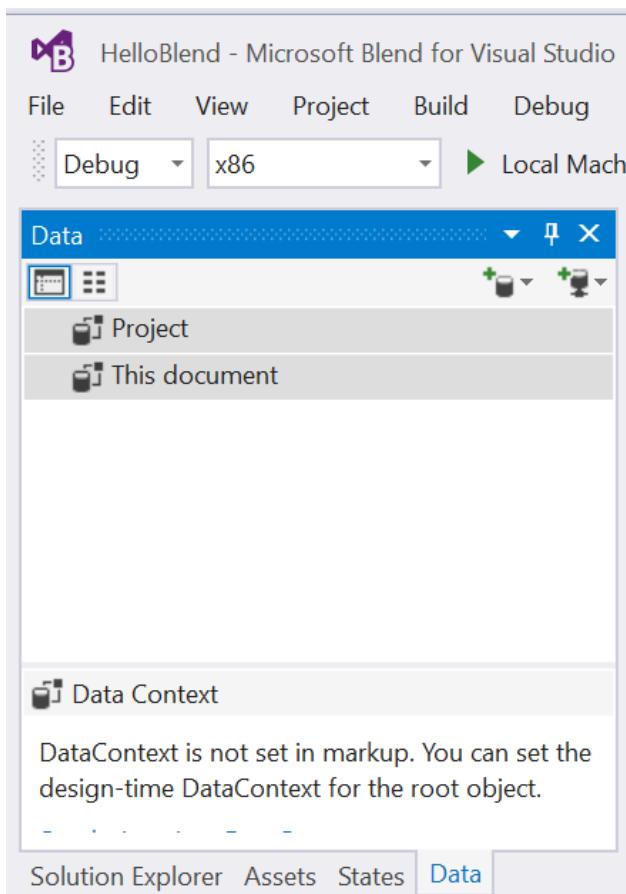


Рисунок 17
Панель данных в Blend.

Примечание: Если вы потеряете или закроете панель, вы можете набрать её название в поиске Quick Launch (Быстрый запуск).

2. В Панели данных откройте меню **Create sample data (Создать данные выборки)** и выберите **New Sample Data (Новые выборочные данные)**.

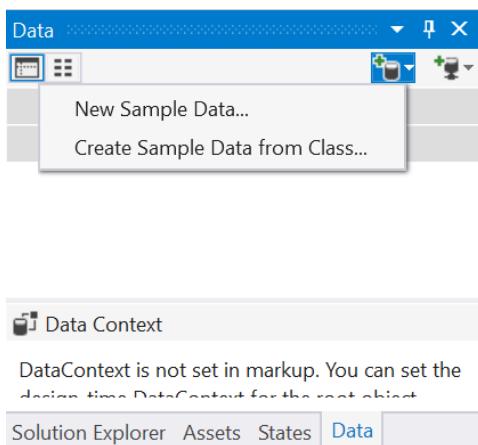


Рисунок 18
Создание выборочных данные из панели данных.

Примечание: Этот инструмент может сгенерировать данные определенного вида, а также позволяет вам использовать функцию **Create Sample Data from Class** (**Создать выборочные данные из класса**) для проектов, в которых вы уже определили схему данных.

3. Когда диалоговое окно **New Sample Data** (**Новые выборочные данные**) откроется, присвойте своему источнику данных имя и выберите, определить ли их для проекта или для текущего документа. Если вы выберете **Project** (**Проект**), выборочные данные будут доступными по всем документам в проекте. Для этой демо-версии приложения мы сделаем выборочные данные доступными для текущего документа XAML, который используется в качестве MainPage (Главная страница).
4. Используйте чекбокс, чтобы включить выборку данных, когда приложение будет выполняться. Нажмите OK для генерации выборочных данных.

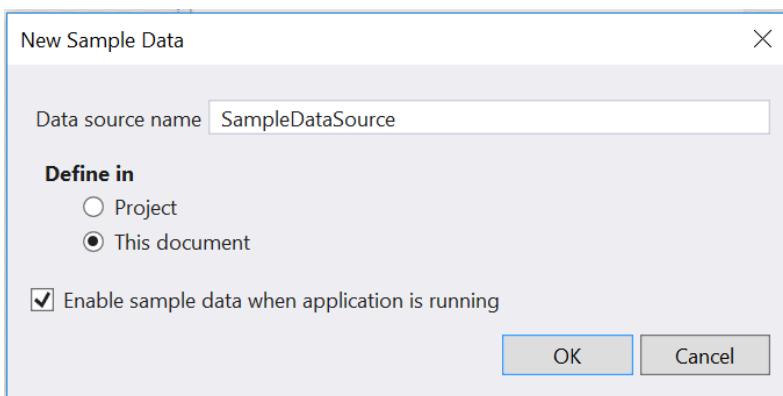


Рисунок 19

Определение выборочных данных для текущего документа XAML.

5. Ваша схема выборочных данных появится в панели **Data** (**Данные**) под пунктом **This document** (**Этот документ**). Возможна небольшая задержка, связанная с загрузкой схемы. Когда она появится, кликните и перетащите на узел **Items** (**Элементы**), чтобы навести курсор мыши на MainPage. Всплывающая подсказка появится, чтобы предварительно представить элементы управления, которые отобразятся в вашем приложении, если вы перетащите узел на окно приложения.

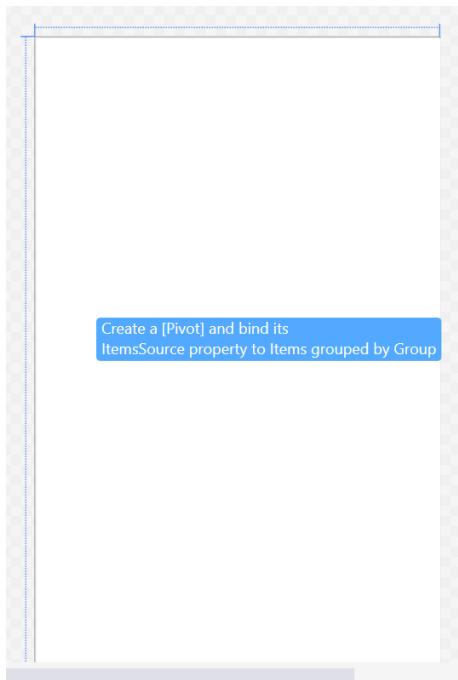


Рисунок 20

Всплывающие подсказки помогают вам предварительно просматривать выборочные данные до того, как вы добавляете их в ваше представление XAML.

6. Перетащите узел **Items (Элементы)** на страницу **MainPage**. Вы увидите образец пользовательского интерфейса для выборки данных.

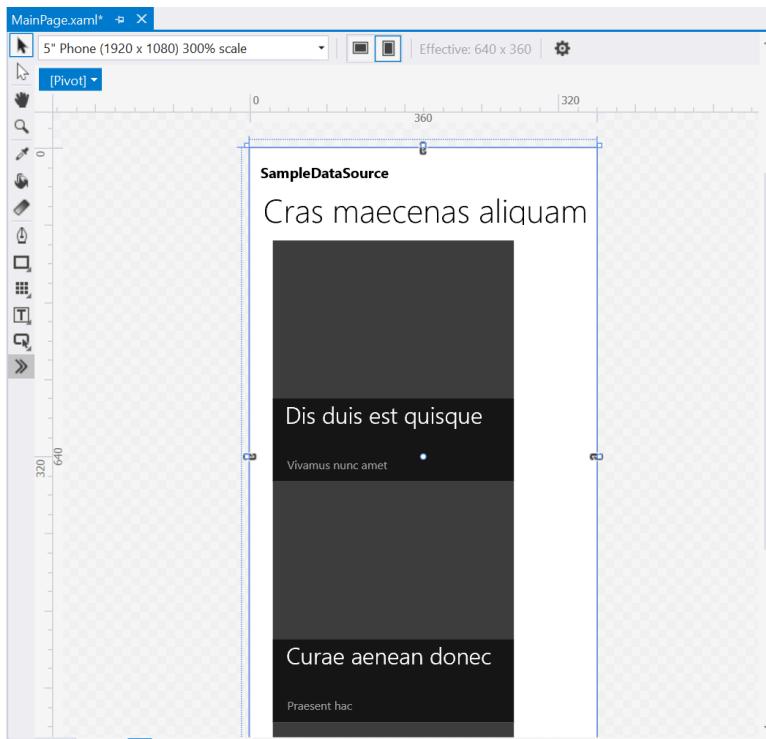


Рисунок 21

Пользовательский интерфейс для выборочных данных в средстве проектирования.

- Соберите и запустите своё приложение на **локальном компьютере**. Используйте те же параметры отладки, которые бы вы использовали в Visual Studio. Ваши выборочные данные будут отображаться динамически в приложении.

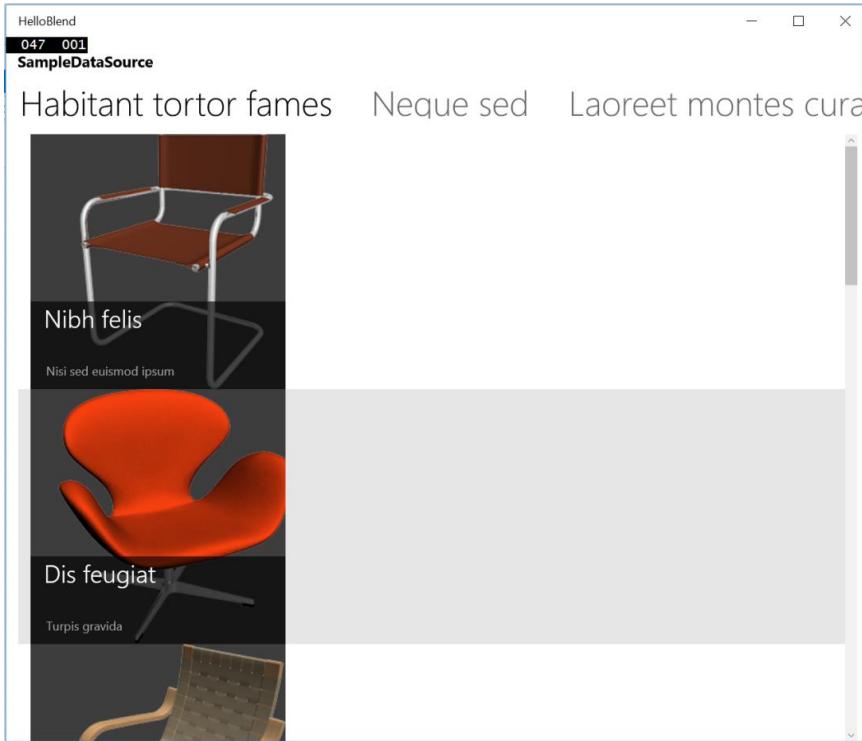


Рисунок 22

Выборочные данные в выполняющемся приложении.

- Отключите отладку и вернитесь в Visual Studio.
- Дополнительно: Создайте и запустите своё приложение в эмуляторе мобильной платформы или на устройстве IoT, чтобы посмотреть, как приложение будет отображаться в семействах других устройств.

Заключение

Универсальная платформа Windows является мощной коллекцией API, которая позволяет вам ориентироваться на широкий ряд устройств с помощью единственного приложения. В этом курсе вы разработали пустое приложение, созданное из шаблона в приложении Hello World, которое отображает специфическую для устройства информацию на всех устройствах под Windows 10. Вы также научились использовать выборочные данные в Blend, чтобы быстро начать создавать и визуализировать интерфейс своего приложения. В следующей работе вы научитесь, как использовать навигацию внутри приложения UWP, работать с обратной навигацией при помощи кнопки "Назад" и внедрять пользовательские элементы управления "Вперед" и "Назад".



Лабораторный практикум

Навигация и кнопка "Назад"

Октябрь 2015 года



Обзор

В платформе UWP заложены инструменты навигации, которые призваны помочь вам обеспечить удобство при работе с любыми устройствами. В то время как приложение UWP будет выполняться любым устройством под Windows 10, необходимо планировать сценарии, в которых аппаратные средства заметно отличаются.

Класс `SystemNavigationManager` позволяет вам использовать программную кнопку возврата, когда аппаратная кнопка "Назад" недоступна или вы не хотите реализовывать в интерфейсе своего приложения навигацию для возврата на шаг назад, используя предоставляемые ОС решения. `SystemNavigationManager` также предлагает универсальное событие `BackRequested`, которое вы можете использовать, чтобы обработать события навигации независимо от семейства устройств или структуры пользовательского интерфейса. Эти новые возможности позволяют вам использовать возможности обратной навигации без дополнительных SDK или платформенно-зависимого кода. В приложениях под Windows 8 кнопка возврата, реализованная в интерфейсе приложения, была единственным способом возврата к предыдущему элементу интерфейса, доступным для планшетов и настольных компьютеров. Сейчас вы можете выбрать любой метод, который лучше всего работает в вашей ситуации, в зависимости от дизайна приложения и пользовательского опыта.

В этой работе мы изучим основы навигации и посмотрим, как использовать ее в простом приложении UWP. Мы поймем, как мы можем передавать данные между страницами приложения и обработаем обратную навигацию с помощью предоставленной оболочкой кнопки возврата.

Цели

Выполнив эту работу, вы сможете:

- Создать новую XAML страницу
- Перемещаться на страницы второго уровня
- Передать параметры навигации
- Использовать платформенную кнопку возврата при переполнении стека переходов назад
- Внедрять стандартный метод запроса возврата

Системные требования

Чтобы выполнить эту работу, необходимо обладать следующим набором программных инструментов:

- Microsoft Windows 10
 - Microsoft Visual Studio 2015
 - Эмулятор Windows 10 Mobile
-

Настройка

Вы должны осуществить следующие шаги для подготовки своего компьютера к этой работе:

1. Установите Microsoft Windows 10.
 2. Установите Microsoft Visual Studio 2015. Выберите пользовательскую установку и убедитесь, что Инструменты разработки для приложений Windows выбраны из списка дополнительных функций.
 3. Установите Windows 10 Mobile Emulator.
-

Упражнения

Данная работа включает следующие упражнения:

1. Навигация по странице
 2. Работа с кнопкой "Назад"
-

Расчётное время для завершения работы: **От 30 до 45 минут.**

Упражнение 1: Навигация по странице

Навигация по странице является важной частью любого приложения. В этом упражнении вы создадите вторую страницу приложения, перейдете к ней с помощью метода Frame.Navigate и научитесь передавать данные для навигации.

Задача 1 – Создание пустого приложения UWP

Начнём с создания проекта из шаблона пустого приложения.

1. В новой версии Visual Studio 2015 выберите **File (Файл) -> New (Новый) -> Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed (Установленное) > Templates (Шаблоны) > Visual C# > Windows > Universal**, а затем выберите шаблон **Blank App (Universal Windows)**.
2. Назовите свой проект **SimpleNavigation** и выберите местоположение в файловой системе, где вы храните свои работы практического курса. Мы создали папку на диске **C** и переименовали ее в **HOL**, именно папку с этим названием вы будете видеть на скриншотах в ходе всего практического курса.

Не изменяйте настройки, установленные для **Create new solution (Создания нового решения)** и **Create directory for solution (Создания папки для решения)**. Вы можете снять галочки как с **Add to source control (Добавить в систему контроля версий)**, так и **Show telemetry in the Windows Dev Center (Отобразить телеметрию в Windows Dev Center)**, если не хотите использовать соответствующие возможности. Нажмите **OK** для создания проекта.

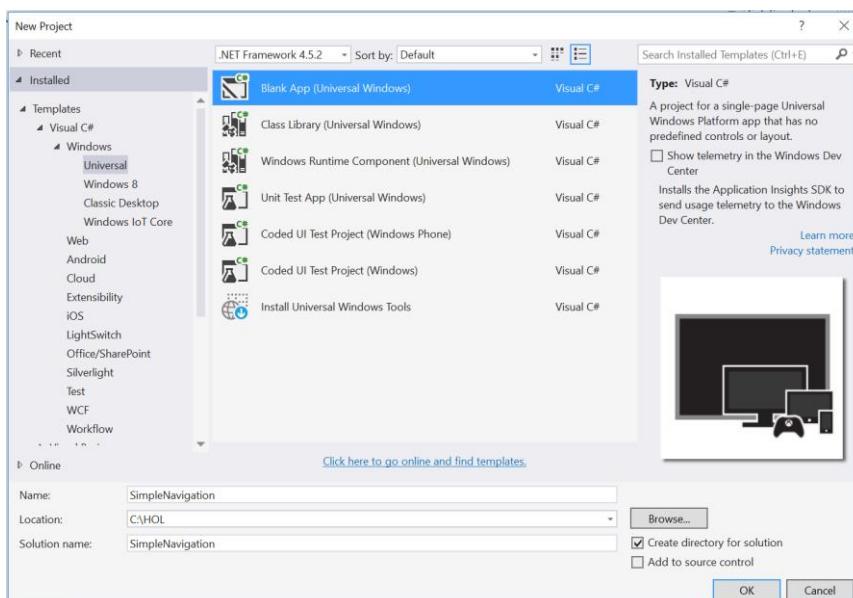


Рисунок 1

Создание нового проекта приложения в Visual Studio 2015.

- Настройте Solution Configuration (Текущую конфигурацию решения) на **Debug (Отладку)** и Solution Platform (Платформу решений) в соответствии с **x86**. Выберите **Local Machine (Локальный компьютер)** из выпадающего меню Debug Target (Цели отладки).

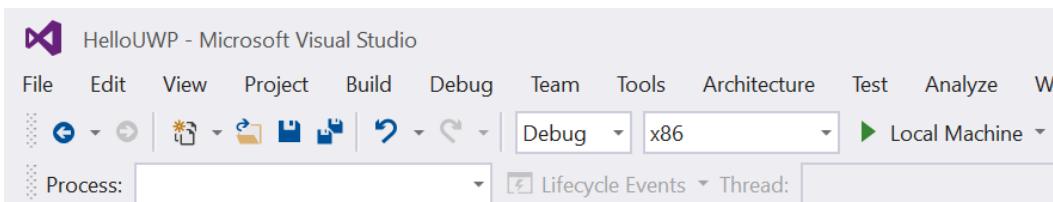


Рисунок 2

Сконфигурируйте свое приложение таким образом, чтобы оно запускалось на Local Machine (Локальном компьютере).

- Скомпилируйте и запустите своё приложение. Вы увидите окно Blank App со счетчиком частоты кадров, активированном по умолчанию для отладки.

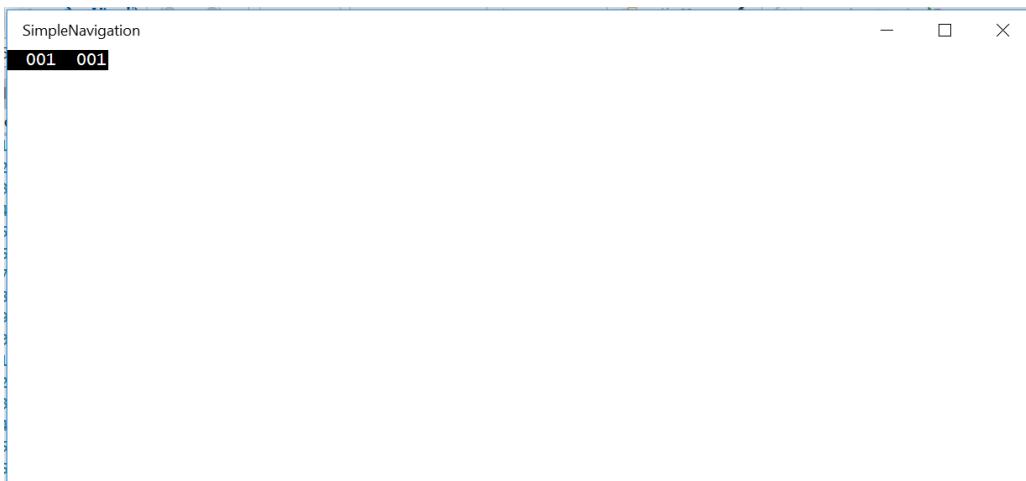


Рисунок 3

Пустое универсальное приложение, запущенное в режиме настольного компьютера.

Примечание: Счетчик частоты кадров является инструментом, используемым в процессе отладки, который помогает следить за производительностью вашего приложения. Он полезен для тех приложений, которые требуют интенсивной графической обработки, однако не подходит для простых приложений, которые будут создаваться вами на данный момент.

В шаблоне пустого приложения директива препроцессора активирует или отключает счетчик частоты кадров посредством **App.xaml.cs**. Счетчик частоты кадров может перекрывать или скрывать контент вашего приложения, если не свернуть его. При выполнении данных работ вы можете отключить его, установив значение **DebugSettings.EnableFrameRateCounter** как **False (Ложное)**.

- Вернитесь к Visual Studio и отключите отладку.

Задача 2 – Создание новой страницы

Перед тем, как вы сможете представить навигацию, вам нужно будет создать вторую страницу, на которую вы сможете перемещаться с главной страницы приложения.

1. В вашем решении SimpleNavigation кликните правой кнопкой мыши на название проекта в Solution Explorer (Обозреватель решения) и выберите **Add (Добавить) > New Item (Новый элемент)**.

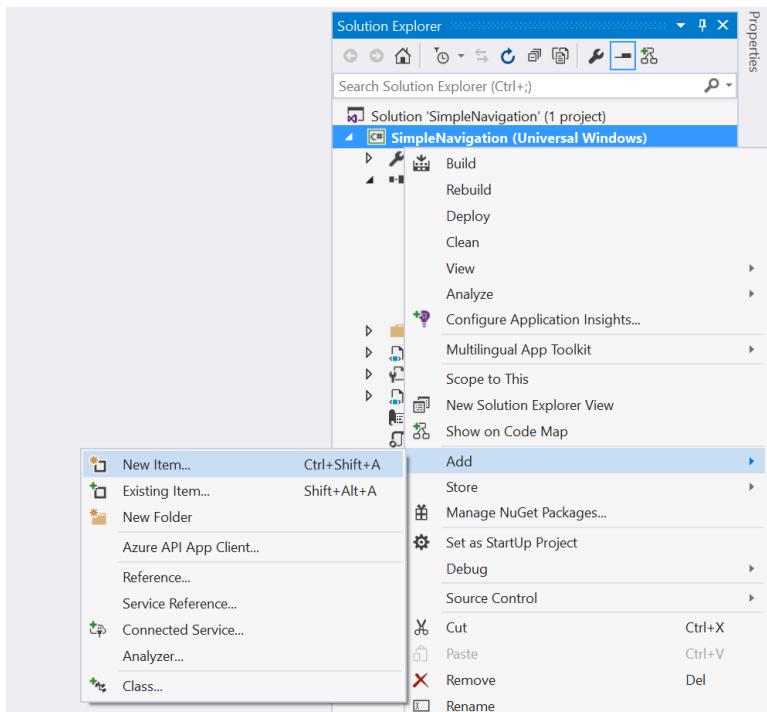


Рисунок 4

Добавление нового элемента в Solution Explorer (Обозреватель решения).

2. Выберите тип элемента **Blank Page (Чистая страница)** в списке элементов Visual C#. Назовите элемент **Page2.xaml** и кликните на кнопку **Add (Добавить)**.

Примечание: Будьте внимательны и не выберите по ошибке XAML View (вид XAML)! Хотя они и похожи, чистая страница и XAML view - это не одно и то же. Чистая страница включает и файл XAML, и связанный файл с выделенным кодом. Вид XAML не включает выделенный код. Мы будем изучать вид XAML в работе, посвященной адаптивным интерфейсам.

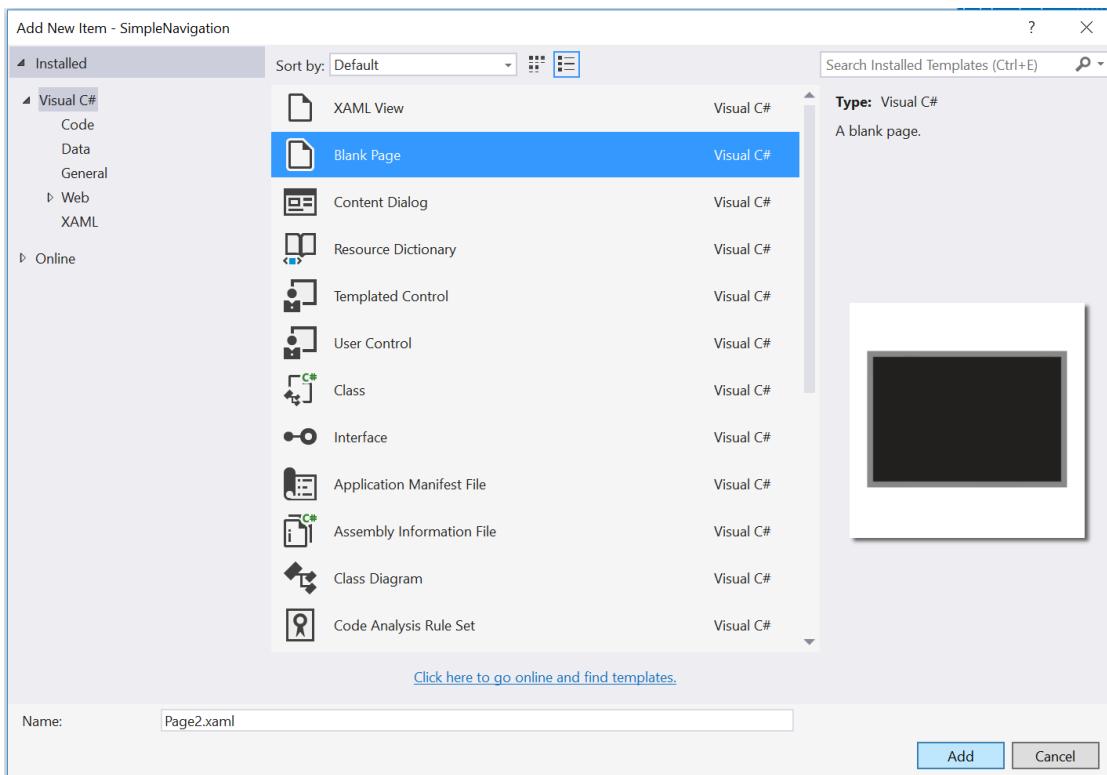


Рисунок 5

Добавление чистой страницы в проект *SimpleNavigation*.

3. Откройте **Page2.xaml** и добавьте текстовый блок **TextBlock** для отображения страниценного заголовка.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <TextBlock Text="Page 2" FontWeight="Light" FontSize="24" Margin="12"
/>
</Grid>
```

Примечание: ThemeResource - расширение для XAML разметки, которое позволяет вам ссылаться на стили XAML, определённые в словаре XAML Resource Dictionary. Расширение ThemeResource может динамически использовать другие ресурсы для отражения активной темы пользователя при запуске. Расширение StaticResource отличается тем, что оно не выполняет обновление во время запуска.

Дополнительную информацию по использованию расширения ThemeResource см. по ссылке <https://msdn.microsoft.com/en-us/library/windows/apps/dn263118.aspx>

4. Откройте **MainPage.xaml** и добавьте туда заголовок страницы. Мы используем простые страницы, таким образом, будет полезно знать, где мы находимся при использовании навигации.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
```

```
<TextBlock Text="Page Navigation" FontWeight="Light" FontSize="24" />
</Grid>
```

Задача 3 – Создание навигации

Сейчас, когда у вас есть две страницы в вашем проекте, вы можете перемещаться между ними. Навигация внутри приложения выполняется во фрейме, который служит как контейнер для ваших страниц. Когда ваше приложение запускается, корневой фрейм встраивается в App.xaml.cs и прикрепляется к окну. Фрейм важен в том смысле, что он управляет навигацией между страницами. В этой задаче вы создадите кнопку на главной странице, выполняющую переход к Странице 2.

1. Откройте **MainPage.xaml** и добавьте кнопку, которая выполняет навигацию на Страницу 2. Определите положение заголовка страницы и кнопку с помощью StackPanel, чтобы улучшить вёрстку страницы.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel HorizontalAlignment="Left" Margin="12">
        <TextBlock Text="Page Navigation" FontWeight="Light" FontSize="24" />
        <Button Content="Go to Page 2" Margin="0,12,0,0" />
    </StackPanel>
</Grid>
```

2. Добавьте событие нажатия на кнопку. Создайте метод **Button_Click**, чтобы обработать навигацию на следующем этапе.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel HorizontalAlignment="Left" Margin="12">
        <TextBlock Text="Page Navigation" FontWeight="Light" FontSize="24" />
        <Button Content="Go to Page 2" Margin="0,12,0,0"
               Click="Button_Click" />
    </StackPanel>
</Grid>
```

3. Сейчас нужно добавить обработчик **Button_Click** в **MainPage.xaml.cs**. Когда вы вызываете **Frame.Navigate**, фрейм загружает контент указанной страницы. Он принимает в качестве параметра тип объекта страницы, для которого вы хотите разместить ссылку в навигации, а второй дополнительный параметр используется для передачи параметра. Мы передадим параметр позже в этом упражнении, сейчас пока будем обходиться без него.

C#

```
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        Frame.Navigate(typeof(Page2));
    }
}
```

4. Создайте и запустите своё приложение на локальном компьютере. Когда вы кликните по кнопке **Go to Page 2 (Перейти к странице 2)** на главной странице, ваш фрейм будет перенаправлен на страницу 2. Мы не включили кнопку "Назад", таким образом, вы пока не сможете вернуться на главную страницу. Вы научитесь обрабатывать команды для возврата к предыдущей странице в следующем упражнении.

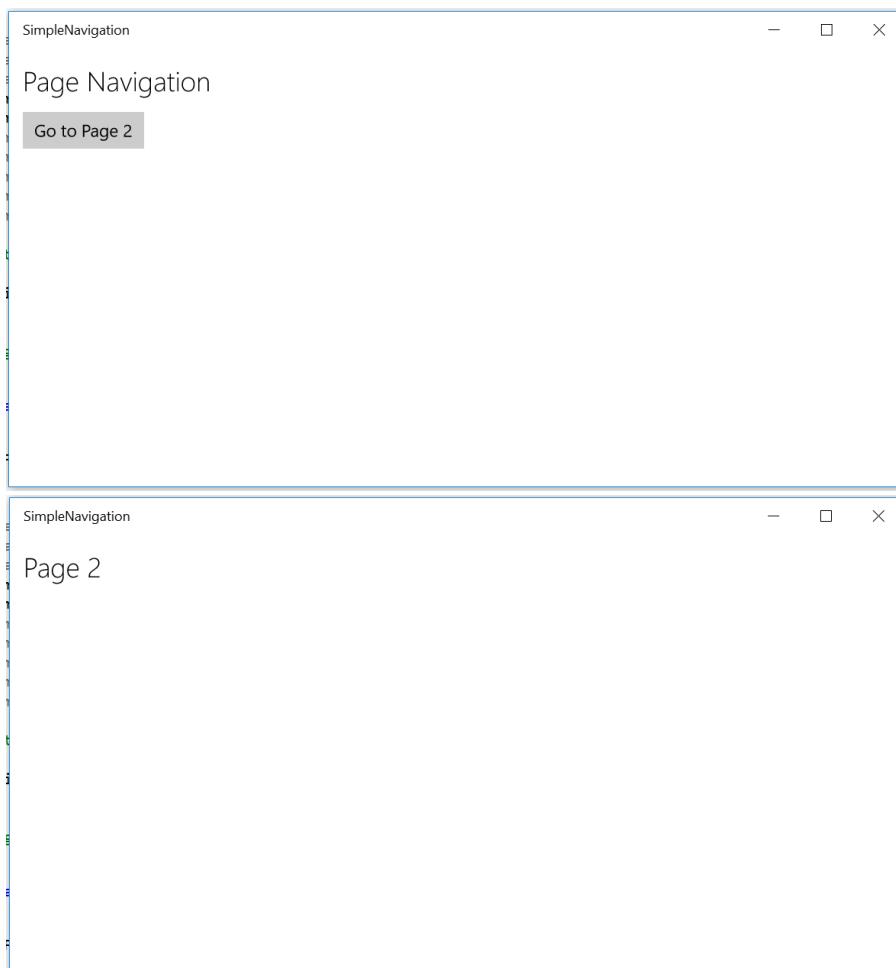


Рисунок 6

Кнопка **Go to Page 2 (Перейти к странице 2)** позволяет осуществлять переход на другую страницу в приложении.

5. Остановите отладку и вернитесь к Visual Studio.

Задача 4 – Передача параметра на страницу 2

Вы научились успешно перемещаться между страницами в своём приложении. Часто бывает полезно передать информацию на новую страницу при навигации. В этой задаче вы передадите параметр с главной страницы на страницу 2.

1. Добавьте текстовый блок **TextBox** в **MainPage.xaml**. Этот текстовый блок будет использоваться для ввода данных пользователями, которые вы передадите на Страницу 2 для отображения позже.

XAML

```
<StackPanel HorizontalAlignment="Left" Margin="12">
    <TextBlock Text="Page Navigation" FontWeight="Light" FontSize="24" />
    <TextBox x:Name="Message" Header="Enter a parameter to send to Page 2"
Width="300" Margin="0,12,0,0" />
    <Button Content="Go to Page 2" Margin="0,12,0,0"
Click="Button_Click" />
</StackPanel>
```

2. В свой обработчик нажатия кнопки **Button_Click** в выделенном коде добавьте второй дополнительный параметр для **Frame.Navigate**, чтобы передать текст в **TextBox** на страницу 2.

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    Frame.Navigate(typeof(Page2), Message.Text);
}
```

Примечание: Дополнительный параметр, который вы передаёте в **Frame.Navigate** не обязательно должен быть строкой, допускается любой объект, но он должен быть сериализуемым. Объект, который является сериализуемым, может преобразовываться в поток байт для хранения, чтобы сохранить его состояние и воспроизвести это состояние позже. Фрейм отслеживает историю приложения и параметры навигации, которые ему нужны, чтобы возобновить работу после приостановления, и он выполняет эту задачу, сериализуя параметры. Если в более сложных случаях вы захотите передать параметр, который не является сериализуемым, вы можете написать код, чтобы сохранять его самостоятельно.

Задача 5 – Отображение сообщения, переданного на страницу 2

Ваше сообщение будет передано как параметр, когда вы перемещаетесь на страницу 2. Однако, мы пока не сделали ничего, чтобы обработать сообщение на странице 2, так что оно

пока не отобразится. Давайте добавим всплывающее диалоговое окно, которое будет отображать сообщение.

1. Откройте **Page2.xaml.cs**. Создайте переопределение для метода **OnNavigatedTo()**.

C#

```
public Page2()
{
    this.InitializeComponent();
}
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);
}
```

Примечание: Конструктор страниц не будет вызываться каждый раз, когда вы переходите к странице, если страница уже была загружена. Метод **OnNavigatedTo()** вызывается каждый раз, когда вы переходите к странице, так что мы можем использовать его, чтобы запустить отображение входящего сообщения.

2. Добавьте пространство имен **Windows.UI.Popups** в выделенный код для страницы 2.

C#

```
using Windows.UI.Popups;
```

3. В функции **OnNavigatedTo()** входящий параметр передается в качестве аргументов **NavigationEventArgs**. Выведем его на экран. Поскольку функция показа диалогового окна – асинхронная, к заголовку функции надо будет добавить ключевое слово **async**.

C#

```
protected async override void OnNavigatedTo(NavigationEventArgs e)
{
    await new MessageDialog("You sent: " + e.Parameter).ShowAsync();

    base.OnNavigatedTo(e);
}
```

Примечание: Асинхронные методы позволяют приложению продолжить выполнение без ожидания операций, которые потенциально могут блокировать пользовательский интерфейс. Пользовательский интерфейс продолжит реагировать на действия пользователя, в то время как операция будет работать в фоновом режиме.

Асинхронный метод по традиции имеет имя, заканчивающееся на **Async**, и возвращает значение **Task** или **Task<T>**. При вызове с использованием ключевого слова **await**, рабочая среда приостанавливает содержащийся метод, и возвращает управление вызывающей программе вместе со значением задачи. Любые методы, вызывающие функции, использующие ключевое слово **await**, должны быть помечено с помощью ключевого слова **async**. Задача обычно выполняется асинхронно на пуле подпроцессов вместо основного потока приложения. Когда выполнение синхронного метода завершается, связанная задача

помечается как завершенная, и доступ к любым возвращаемым значениям может быть осуществлён посредством задачи.

Дополнительную информацию и примеры шаблонов асинхронного программирования см. по ссылке <https://msdn.microsoft.com/en-us/library/hh191443.aspx>

4. Скомпилируйте и запустите своё приложение. Наберите сообщение в текстовом поле на главной странице и перейдите к странице 2. Вы увидите, что на странице 2 появилось всплывающее окно с вашим сообщением.

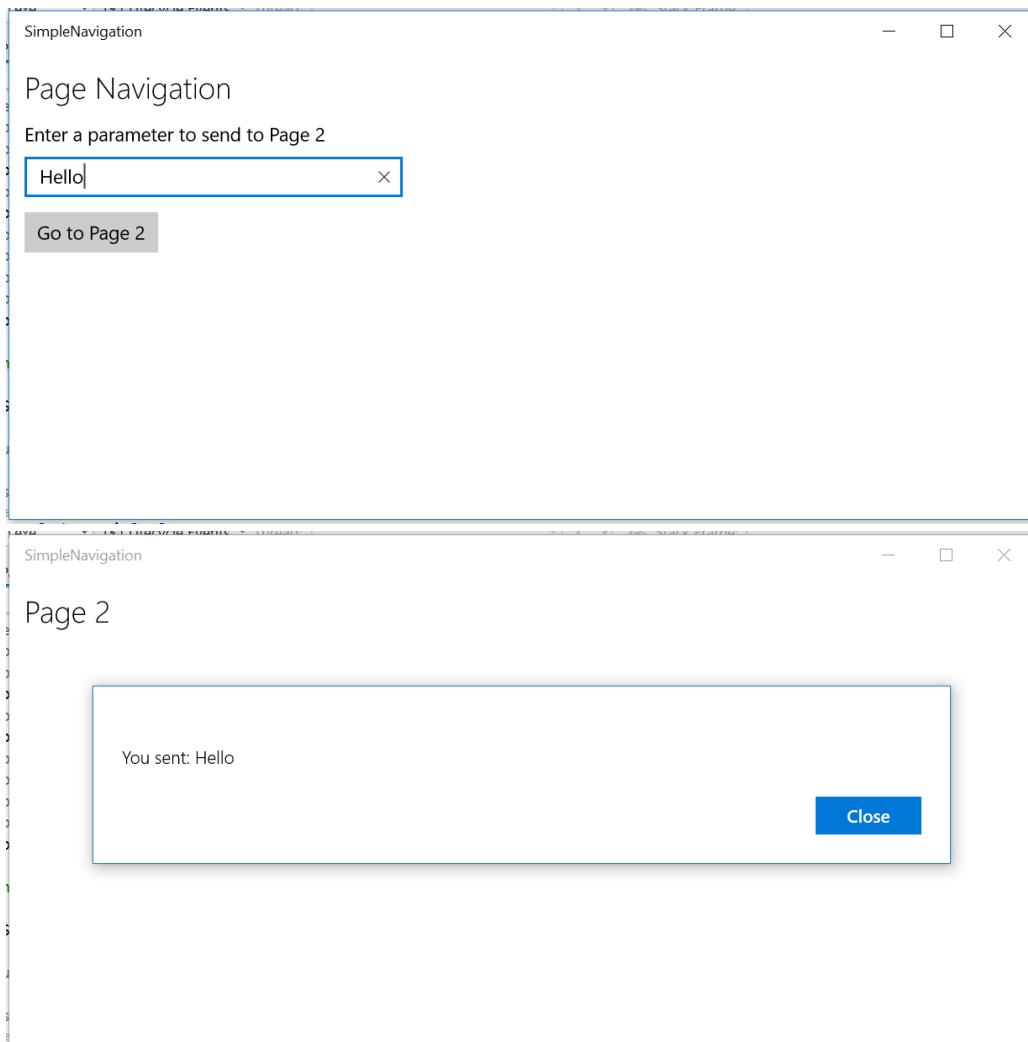


Рисунок 7

Сгенерированное пользователями сообщение передано на Страницу 2 как параметр.

5. Завершите отладку и вернитесь к Visual Studio.

Упражнение 2: Работа с кнопкой "Назад"

Приложения UWP выполняются на целом ряде устройств, которые отличаются тем, как они обращаются с обратной навигацией. Тогда как устройства под Windows 10 Mobile обычно имеют аппаратную кнопку "Назад", планшеты и настольные ПК обычно лишены подобных инструментов. Существует несколько способов обращения с возвратной навигацией внутри приложения. В этом упражнении вы изучите различия в навигации между семействами устройств и будете использовать предоставленную платформой кнопку "Назад", чтобы вернуться на главную страницу со второстепенной страницы.

Задача 1 – Проверка работы аппаратной кнопки "Назад"

Давайте посмотрим на существующий порядок работы аппаратной кнопки "Назад" на мобильном устройстве.

1. Переключите режим отладки на эмулятор мобильной платформы. Мы выбрали **Mobile Emulator 10.0.10240.0 720p 5 inch 1GB**.
2. Создайте и запустите приложение SimpleNavigation в эмуляторе. Перейдите на страницу 2. На странице 2 кликните или нажмите на аппаратную кнопку "Назад" эмулятора. Вместо того, чтобы вернуться на главную страницу, приложение завершит работу и вернёт вас в ваше последнее положение в стек переходов назад. В этом случае, если вы просто запустили эмулятор, последним положением будет экран запуска.
3. Откройте приложение Photos (Фото) с экрана запуска.
4. Кликните по аппаратной кнопке "Назад" в эмуляторе и удерживайте её, чтобы просмотреть открытые приложения. Используйте эту функцию, чтобы вернуться к своему приложению Simple Navigation.



Рисунок 8

Нажмите и удерживайте кнопку "Назад", чтобы перемещаться между открытыми приложениями.

5. В приложении Simple Navigation нажмите на обратную кнопку снова. Вы вернётесь к предыдущему приложению в стеке приложений, которое является приложением Фото, а не к главной странице приложения Simple Navigation, как вы могли бы ожидать. Если вы не реализуете пользовательский возврат назад, аппаратная кнопка "Назад" будет перемещаться через стек приложений по умолчанию.
Вы реализуете код, чтобы обращаться с пользовательским возвратом назад в **работе № 3** этого курса.

Задача 2 – Реализация программной кнопки "Назад"

Windows 10 предоставляет кнопку "Назад" в панели задач в режиме планшета, но по умолчанию кнопка "Назад" в интерфейсе операционной системы в режиме Рабочего Стола отсутствует. **SystemNavigationManager** предоставляет опции, чтобы включить метод **AppViewBackButton** для приложений, выполняющихся в режиме рабочего стола. Эта программная кнопка "Назад" отображается в панели заголовка приложения при работе в

режиме рабочего стола. Если вы напишите код для показа кнопки "Назад" операционной системой, он будет проигнорирован на мобильных устройствах, потому что операционная система ожидает наличия аппаратной кнопки "Назад", и проигнорирован при работе в режиме планшета. В этой задаче вы отобразите предоставляемую системой кнопку возврата в режиме рабочего стола, когда пользователь будет находиться на странице в приложении, где возможен возврат назад.

1. Откройте **App.xaml.cs**. Добавьте пространство имён **Windows.UI.Core**.

C#

```
using Windows.UI.Core;
```

2. Добавьте обработчик событий **rootFrame.Navigated** в конце переопределения **OnLaunched**. Этот обработчик событий будет запускать каждый раз имеющуюся навигацию на корневом фрейме и отобразит кнопку "Назад", если стек переходов назад приложения не будет пустым.

C#

```
// Убедитесь, что текущее окно активно
Window.Current.Activate();

rootFrame.Navigated += RootFrame_Navigated;
}

private void RootFrame_Navigated(object sender, NavigationEventArgs e)
{
    Frame rootFrame = Window.Current.Content as Frame;

    SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility
        = rootFrame.CanGoBack ? AppViewBackButtonVisibility.Visible :
                                         AppViewBackButtonVisibility.Collapsed;
}
```

Примечание: Мы хотим отображать кнопку "Назад" операционной системы, когда стек переходов назад не пустой. В этом коде устанавливается значение **AppViewBackButtonVisibility** по **AppViewBackButtonVisibility.Visible**, если значение **rootFrame.CanGoBack** - true, иначе используется значение **AppViewBackButtonVisibility.Collapsed**.

3. Измените платформу отладки на **Local Machine (Локальный компьютер)**. Создайте и запустите своё приложение и перейдите к странице 2. Кнопка "Назад" появится в панели заголовка в режиме Рабочего Стола.

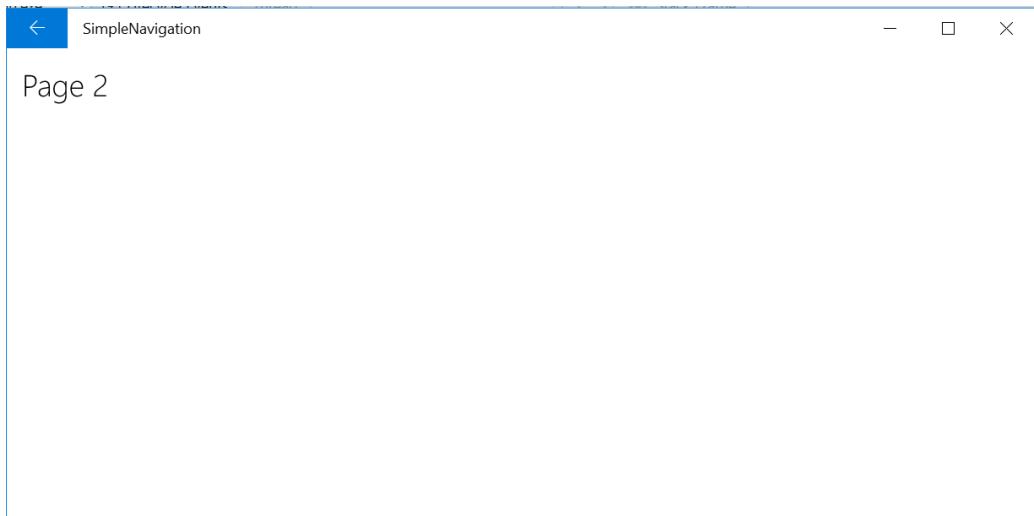


Рисунок 9

Значение AppViewBackButton - visible, кнопка отображается в панели заголовка в режиме Рабочего Стола.

4. Кликните по кнопке "Назад" в панели заголовка. Ничего не произойдёт. Хотя вы обеспечили видимость кнопки, она ещё не подключена для работы с навигацией.

Примечание: Цвет кнопки определяется выбранной пользователем цветовой схемой, которую пользователь может изменить в **Windows 10 Settings (Настройки Windows 10) > Personalization (Персонализация) > Colors (Цветовая схема)**. Если вы захотите управлять положением или стилем кнопки "Назад", можно реализовать пользовательский вариант кнопки "Назад" в самом приложении, что вы сделаете позднее в рамках данного курса.

5. Не закрывая приложение, используйте панель уведомлений Windows 10, чтобы переключиться в режим **Планшета**. Кнопка "Назад" появится в панели задач.

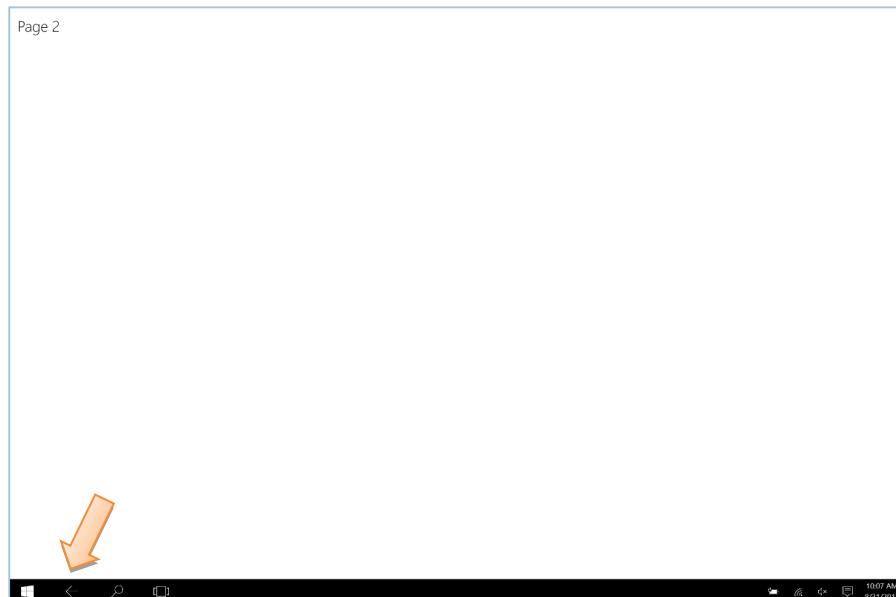


Рисунок 10

Кнопка "Назад" отображается в панели задач в режиме планшета.

6. Кликните по кнопке "Назад" в панели задач. Вы вернётесь в стек приложений. В отличие от кнопки в панели заголовка, кнопка "Назад" в режиме планшета демонстрирует то же поведение, что и аппаратная кнопка на мобильном устройстве. Эта кнопка отображается, даже если кнопка "Назад" оболочки в панели заголовка не включена.

Примечание: В режиме разделенного экрана существует стек переходов назад, доступный для каждой стороны экрана отдельно.

7. Используйте панель уведомлений, чтобы вернуться к режиму Настольного Компьютера. Остановите отладку и вернитесь к Visual Studio.
8. Запустите приложение снова, в этот раз в эмуляторе мобильной платформы. Когда вы будете перемещаться к странице 2, кнопка "Назад" оболочки не появится, потому что она игнорируется в режиме работы мобильной платформы.

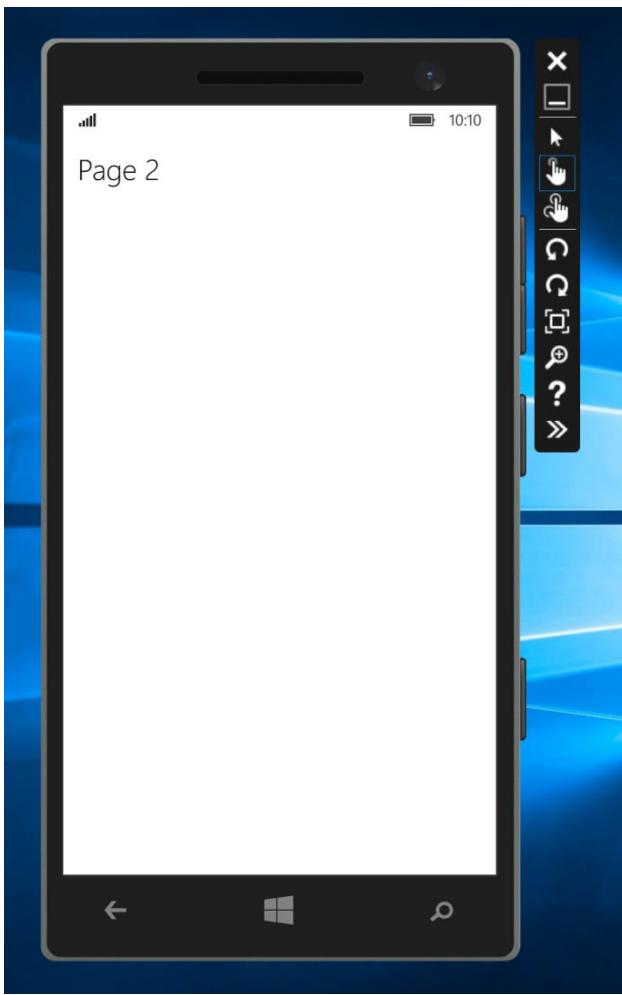


Рисунок 11

AppViewBackButton скрыт в мобильном режиме работы.

9. Завершите отладку и вернитесь к Visual Studio.

Задача 3 – Обработка стандартного запроса на возврат назад

На данный момент мы имеем интерфейс перехода назад, работающий как под Windows 10 Mobile, так и в настольном режиме Windows 10 и в режиме планшета. Давайте рассмотрим стандартную модель запроса на переход назад, чтобы обрабатывать подобные запросы.

1. В **App.xaml.cs** создайте и подпишитесь на событие **App_BackRequested**.

Примечание: Подписка на **App_BackRequested** должна производиться только после того, как вид был создан. Если вы попытаетесь выполнить этот код до того, как вид будет создан, **SystemNavigationManager.GetForCurrentView()** будет возвращать значение **null**.

C#

```
SystemNavigationManager.GetForCurrentView().BackRequested +=  
    App_BackRequested;  
  
    rootFrame.Navigated += RootFrame_Navigated;  
}  
  
private void RootFrame_Navigated(object sender, NavigationEventArgs e)  
{  
    Frame rootFrame = Window.Current.Content as Frame;  
    SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility  
=  
    rootFrame.CanGoBack ?  
        AppViewBackButtonVisibility.Visible :  
        AppViewBackButtonVisibility.Collapsed;  
};  
  
private void App_BackRequested(object sender, BackRequestedEventArgs e)  
{  
}
```

2. В обработчике событий **App_BackRequested** проверьте, было ли событие **BackRequested** обработано. Если нет, установите по умолчанию перемещение назад в пределах фрейма. Убедитесь, что вы установили для **e.Handled** значение **true**, после выполнения.

C#

```
private void App_BackRequested(object sender, BackRequestedEventArgs e)  
{  
    // Убедитесь, что никто ещё не обработал это событие  
    if (!e.Handled)  
    {  
        // По умолчанию установлен переход назад в пределах фрейма  
        Frame frame = Window.Current.Content as Frame;  
        if (frame.CanGoBack)  
        {  
            frame.GoBack();  
        }  
    }  
}
```

```

        // Сигнал обработан так, что система не осуществляет переход
        // назад
        // через стек приложений
        e.Handled = true;
    }
}

```

3. Скомпилируйте и запустите своё приложение. Перейдите к **странице 2**, затем используйте кнопку "Назад" оболочки, чтобы вернуться к главной странице.
4. Остановите отладку и вернитесь к Visual Studio.

Задача 4 – Отрисовка кнопки "Назад" в приложении

Если бы вы хотели иметь больше контроля над стилем и поведением кнопки "Назад", вы можете создать свою собственную кнопку, чтобы работать с навигацией при переходе назад.

1. Закомментируйте видимость кнопки "Назад" оболочки в **App.xaml.cs**.

C#

```
//rootFrame.Navigated += RootFrame_Navigated;
```

2. Добавьте кнопку в **Page2.xaml** и привяжите её стиль к **StaticResource NavigationBackButtonNormalStyle**. Установите положение кнопки и заголовка страницы в горизонтальное положение в **StackPanel** для выравнивания. Замените значение поля **TextBlock** заголовка на **VerticalAlignment="Top"**.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel Orientation="Horizontal">
        <Button Style="{StaticResource NavigationBackButtonNormalStyle}"
            VerticalAlignment="Top" />
        <TextBlock Text="Page 2" FontWeight="Light"
            FontSize="24" VerticalAlignment="Top" />
    </StackPanel>
</Grid>
```

3. Добавьте событие клика на кнопку и свяжите его с обработчиком **GoBack**. Вы создадите обработчик **GoBack** на следующем этапе.

XAML

```
<Button Style="{StaticResource NavigationBackButtonNormalStyle}"
    Click="GoBack" VerticalAlignment="Top" />
```

4. В выделенном коде страницы 2 добавьте функцию **GoBack()**, чтобы обработать событие перехода назад.

C#

```
private void GoBack()
{
```

```
        if (Frame.CanGoBack)
            Frame.GoBack();
    }
```

5. Скомпилируйте и запустите своё приложение. Когда вы будете перемещаться к странице 2, вы увидите свою пользовательскую кнопку "Назад" вместо кнопки "Назад" оболочки. Пользовательская кнопка вернёт вас на главную страницу.

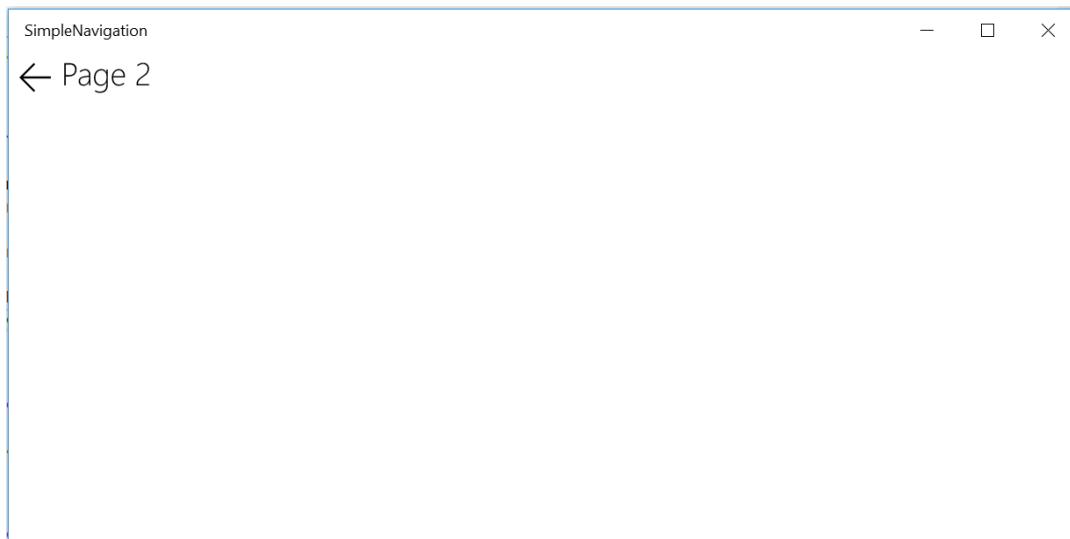


Рисунок 12

Пользовательская кнопка возврата на странице 2.

6. Завершите отладку и вернитесь к Visual Studio.

Задача 5 – Управление видимостью кнопки "Назад" в приложении

Давайте сделаем пользовательскую кнопку возврата видимой, только когда в стеке переходов назад есть что-то.

1. Добавьте закрытое поле **Visibility (Видимость)** в выделенный код Страницы 2.

C#

```
public sealed partial class Page2: Page
{
    public Visibility CanGoBack;
```

2. В переопределении страницы 2 **OnNavigatedTo** проверьте, может ли фрейм быть возвращён, и настройте поле видимости соответственно.

C#

```
protected async override void OnNavigatedTo(NavigationEventArgs e)
{
    if (Frame.CanGoBack)
        CanGoBack = Visibility.Visible;
    else
        CanGoBack = Visibility.Collapsed;
```

```
        await new MessageDialog("You sent: " + e.Parameter).ShowAsync();
    }
```

3. Свяжите видимость своей кнопки возврата с полем **CanGoBack** на **Странице 2.xaml**.

XAML

```
<Button Style="{StaticResource NavigationBackButtonNormalStyle}"
       Click="{x:Bind GoBack}" Visibility="{x:Bind CanGoBack}"
       VerticalAlignment="Top" />
```

4. В **App.xaml.cs** настройте первоначальную страницу запуска для своего приложения, чтобы она указывала на **Page2** вместо **MainPage**.

C#

```
if (rootFrame.Content == null)
{
    // Когда стек навигации не может быть восстановлен, перемещайтесь к
    // первой странице,
    // конфигурируя новую страницу, передавая требуемую информацию
    // как параметр навигации
    rootFrame.Navigate(typeof(Page2)), e.Arguments);
}
```

5. Скомпилируйте и запустите своё приложение. Пользовательская кнопка "Назад" не появится на странице 2, потому что в стек переходов назад пуст.



Рисунок 13

Пользовательская кнопка "Назад" не появляется, когда стек переходов назад пуст.

6. Отключите отладку и вернитесь к **App.xaml.cs**. Замените страницу запуска обратно на **MainPage**.

C#

```
if (rootFrame.Content == null)
{
    // Когда стек навигации не может быть восстановлен, перемещайтесь к
    // первой странице, конфигурируя новую страницу, передавая требуемую
    // информацию как параметр навигации
    rootFrame.Navigate(typeof(MainPage), e.Arguments);
}
```

7. Скомпилируйте и запустите своё приложение. Перейдите на страницу 2. Пользовательская кнопка "Назад" снова появится, когда стек будет заполнен.
 8. Завершите отладку и вернитесь к Visual Studio.
-

Краткий обзор

В этой работе вы узнали о фреймах, навигации между страницами и поведении при переходе назад в пределах приложения UWP. Хотя опции навигации варьируются для разных устройств, вы научились создавать приложения, которые одинаково хорошо ведут себя на настольном компьютере, планшете и мобильном устройстве, подстраиваясь под принятый для данного форм-фактора стиль навигации.



Лабораторный практикум

Создание адаптивного
пользовательского интерфейса

Октябрь 2015 года



Обзор

Приложения UWP могут выполняться на разных семействах устройств, которые существенно различаются в размере экранов и наборе функций. Чтобы обеспечить оптимальный пользовательский опыт на всех устройствах, дизайн должен быть адаптирован под устройство. Адаптивный UI может получать информацию об устройстве, на котором запущено приложение, и отображать макет соответственно характеристикам этого устройства.

Адаптивный (adaptive) интерфейс отличается от отзывчивого (responsive), например, он может предоставить индивидуализированный макет для каждого семейства устройств или размера экрана. Отзывчивый интерфейс отображает отдельный, гибкий дизайн, который выглядит корректно на всех устройствах. Один из недостатков такого пользователяского интерфейса - более медленная загрузка, элементы загружаются для всех устройств и разрешений, даже несмотря на то, что они, возможно, не потребуются. Адаптивный UI может быть основан на отзывчивом, однако вы также имеете возможность отображать уникальные виды для устройств, которые имеют мало общего друг с другом. Например, изображение на Xbox может полностью отличаться от вида приложения на рабочем столе компьютера и вида мобильного приложения, потому что способы взаимодействия пользователя с интерфейсом различаются.

В этом курсе мы переработаем фиксированный макет в адаптивный UI. Первоначально пользовательский интерфейс будет соответствовать более-менее стандартной практике. Затем мы создадим отдельный вид, чтобы обработать уникальные элементы дизайна для мобильного устройства.

Цели

Настоящий курс научит вас:

- Превращать фиксированный макет в адаптивный UI
- Создавать визуальные состояния приложения, чтобы поддерживать узкие, средние и широкие экраны
- Использовать класс `RelativePanel`, чтобы изменять состояние контента
- Использовать схему настройки, чтобы лучше приспосабливать стили для небольших экранов
- Использовать вид XAML, чтобы создавать уникальный вид для отдельных семейств устройств

Системные требования

Чтобы выполнить эту работу, необходимо обладать следующим набором программных инструментов:

- Microsoft Windows 10
 - Microsoft Visual Studio 2015
 - Эмулятор Windows 10 Mobile
-

Настройка

Вы должны осуществить следующие шаги для подготовки своего компьютера для этой работы:

1. Установите Microsoft Windows 10.
 2. Установите Microsoft Visual Studio 2015. Выберите пользовательскую установку и убедитесь, что Инструменты разработки для приложений Windows выбраны из списка дополнительных функций.
 3. Установите Windows 10 Mobile Emulator.
-

Упражнения

Эта практическая работа включает в себя следующие упражнения:

1. Переход к адаптивному пользовательскому интерфейсу
 2. Класс RelativePanels с визуальными состояниями
 3. Адаптивный пользовательский интерфейс с видами XAML
-

Расчётное время для завершения курса: **От 30 до 45 минут.**

Упражнение 1: Переход к адаптивному пользовательскому интерфейсу

При создании приложения использование фиксированного макета может показаться самым приемлемым решением из-за его простоты. Однако фиксированный дизайн не совсем подходит для приложений UWP, которые могут отображаться на различных по размеру и ориентации экранах. В этом упражнении вы создадите простой фиксированный макет, чтобы показать изображение и связанные метаданные. Затем вы преобразуете фиксированный макет в адаптивный.

Задача 1 – Создание приложения UWP

Начнём с создания проекта из шаблона пустого приложения.

1. В новой версии Visual Studio 2015 выберите **File (Файл) -> New (Новый) -> Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed (Установленное) > Templates (Шаблоны) > Visual C# > Windows > Universal**, а затем выберите шаблон **Blank App** приложения (Universal Windows).
2. Назовите свой проект **AdaptiveUI** и выберите местоположение файловой системы, в которое будет осуществлено сохранение результатов прохождения Лабораторного практикума. Мы создали папку на диске **C** и переименовали ее в **HOL**, именно папку с этим названием вы будете видеть на скриншотах в ходе всего практического курса.

Не изменяйте настройки, установленные для **Create new solution (Создания нового решения)** и **Create directory for solution (Создания папки для решения)**. Вы можете снять галочки как с **Add to source control (Добавить в систему контроля версий)**, так и **Show telemetry in the Windows Dev Center (Отобразить телеметрию в Windows Dev Center)**, если не хотите использовать соответствующие возможности. Нажмите **OK** для создания проекта.

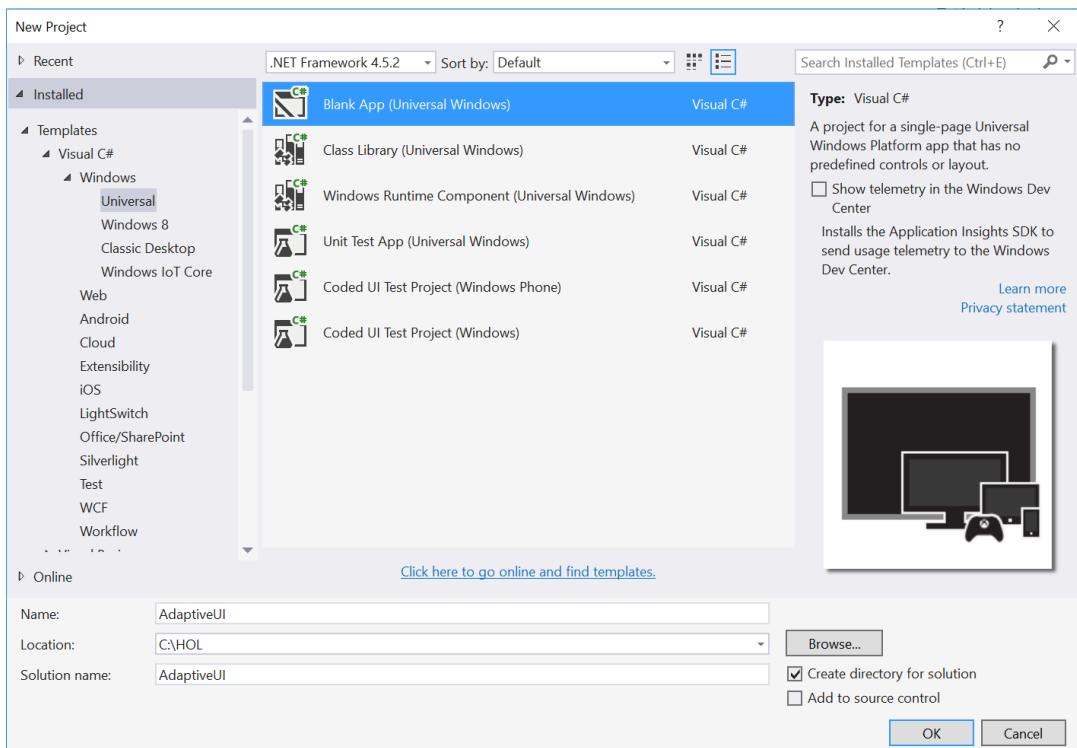


Рисунок 1

Создание нового проекта приложения в Visual Studio 2015.

3. Настройте Solution Configuration (Текущую конфигурацию решения) на **Debug (Отладка)** и Solution Platform (Платформа решений) на **x86**. Выберите **Local Machine (Локальный компьютер)** из выпадающего меню Debug Target (Цели отладки).

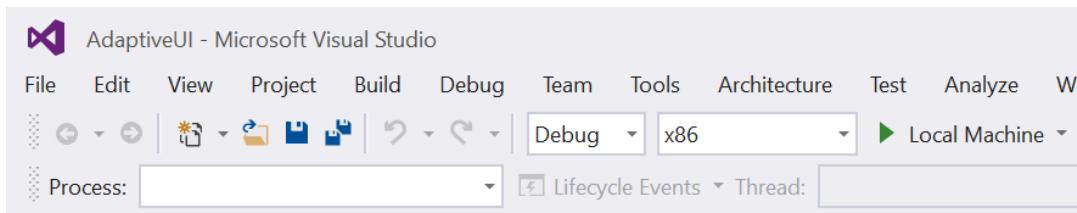


Рисунок 2

Сконфигурируйте свое приложение таким образом, чтобы оно запускалось на Local Machine (Локальном компьютере).

4. Скомпилируйте и запустите своё приложение. Вы увидите пустое окно приложения со счётчиком скорости кадров, активированном по умолчанию в режиме отладки.

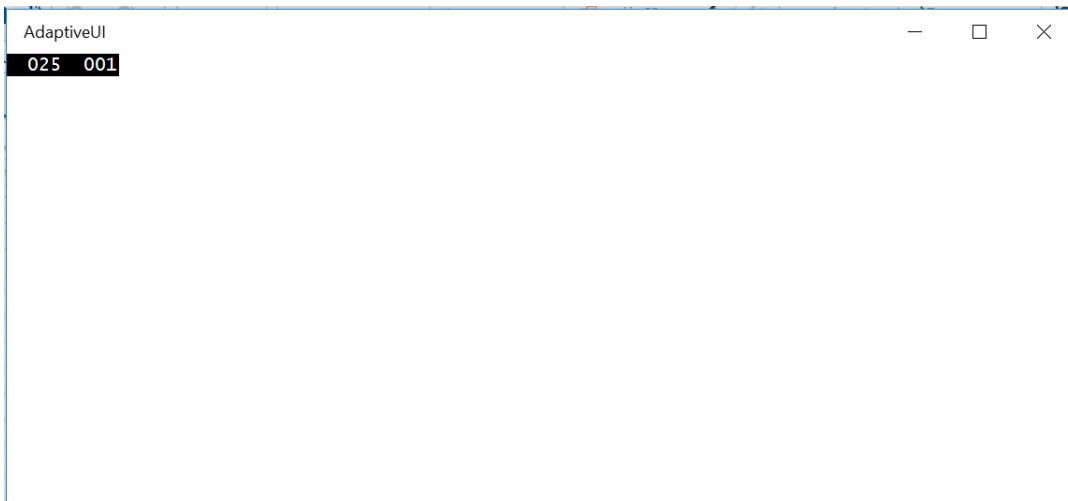


Рисунок 3

Пустое универсальное приложение, запущенное в режиме настольного компьютера.

Примечание: Счетчик частоты кадров является инструментом, используемым в процессе отладки, который помогает следить за производительностью вашего приложения. Он полезен для тех приложений, которые требуют интенсивной графической обработки, однако не подходит для простых приложений, которые будут создаваться вами в данном практическом курсе.

В шаблоне пустого приложения директива препроцессора для активации или отключения счетчика частоты кадров находится на **App.xaml.cs**. Счётчик скорости кадра может перекрываться или может скрыть ваше содержание приложения, если вы оставляете его. При выполнении данных работ вы можете отключить его, установив значение **DebugSettings.EnableFrameRateCounter** как **False (Ложное)**.

5. Вернитесь к Visual Studio и остановите отладку.

Задача 2 – Создание фиксированного макета

В качестве отправного пункта мы создадим основной фиксированный макет, содержащий простой контент. Содержание будет состоять из большого изображения и связанных метаданных: имени пользователя, аватара, имени изображения, даты и описания. Мы отформатируем содержание в две колонки.

1. Откройте **MainPage.xaml**. Добавьте два класса **ColumnDefinitions** к сетке с фиксированной шириной **500 эффективных пикселей**.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="500" />
        <ColumnDefinition Width="500" />
    </Grid.ColumnDefinitions>
</Grid>
```

- Кликните правой кнопкой мыши на папке **Assets (Ресурсы)** и выберите **Add (Добавить) > Existing Item (Существующий элемент)**. Используйте диалоговое окно, чтобы переместиться к папке **Assets** и выбрать **airtime.jpg** и **avatar.jpg**. Добавьте их в свой проект.
- Вернитесь к **MainPage.xaml**. Добавьте изображение **airtime** в первую колонку.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="500" />
        <ColumnDefinition Width="500" />
    </Grid.ColumnDefinitions>
    <Image Grid.Column="0" Source="Assets/airtime.jpg"
        Stretch="UniformToFill" />
</Grid>
```

- Добавьте аватар и текстовые метаданные во второй колонке.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="500" />
        <ColumnDefinition Width="500" />
    </Grid.ColumnDefinitions>
    <Image Grid.Column="0" Source="Assets/airtime.jpg"
        Stretch="UniformToFill"
        />
    <StackPanel Grid.Column="1" Background="LightBlue">
        <Image Source="Assets/avatar.jpg" Width="100" Height="100"
            HorizontalAlignment="Left" />
        <TextBlock Text="phutureproof" />
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France." />
    </StackPanel>
</Grid>
```

- Скомпилируйте и запустите своё приложение. Для некоторых форматов экрана содержание будет четко умещаться на экране. Измените размер своего окна, чтобы увидеть поведение при меньшем и большем размерах окна. При меньших размерах экрана вторая колонка обрезается, а при больших размерах контент окружается пустым пространством.

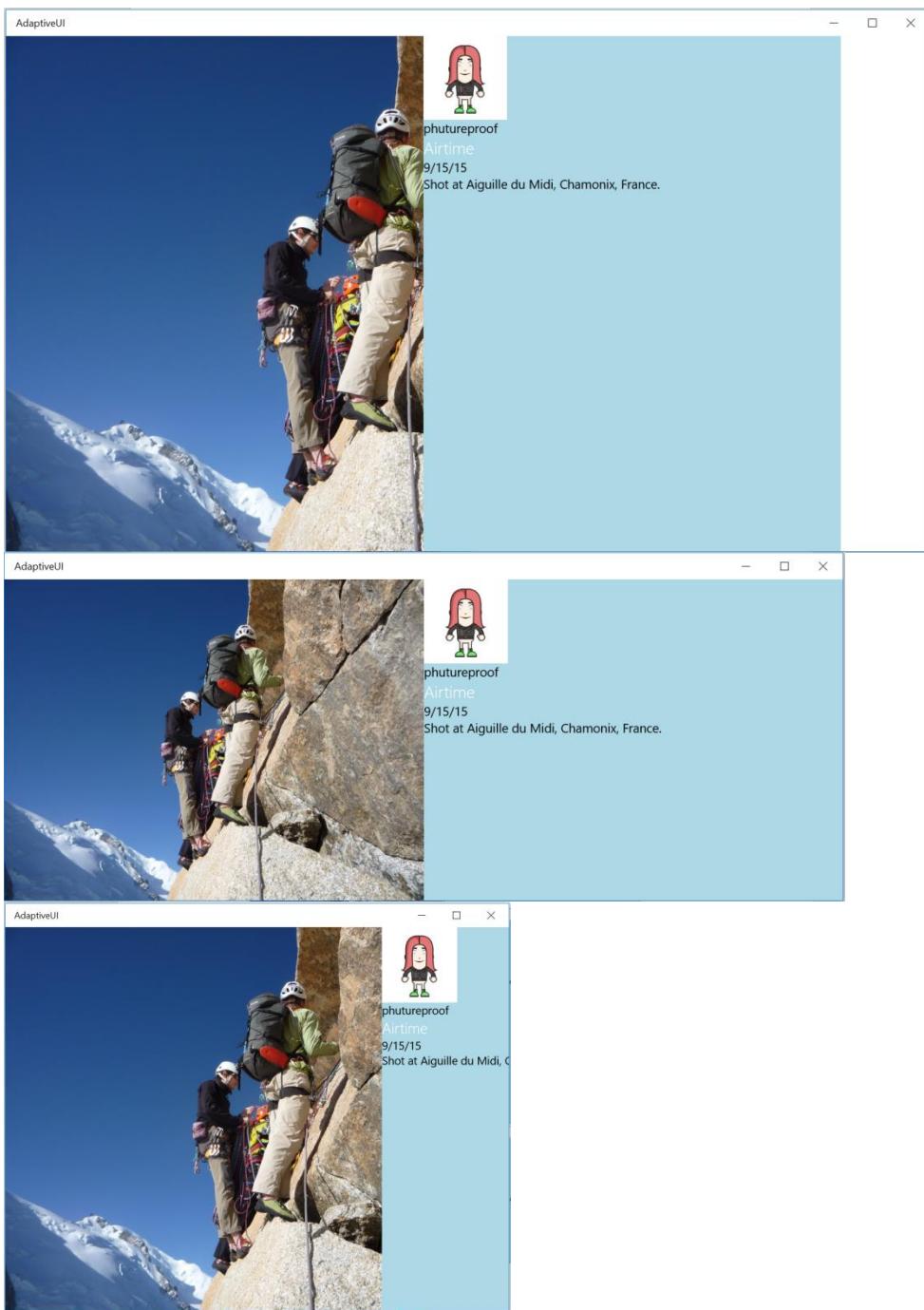


Рисунок 4

Колонки фиксированной ширины при разных размерах окна.

6. Остановите отладку и вернитесь к Visual Studio.

Задача 2 – Адаптация макета для различных размеров экрана

Сейчас, когда мы увидели недостатки фиксированного макета для приложения UWP, давайте сделаем его адаптивным. Мы будем следовать шести принципам гибкого дизайна:

- Reposition – репозиционирование, изменение положения
- Resize - изменение размера
- Reflow – изменение обтекания
- Reveal – открытие некоторых частей UI при увеличении размера
- Replace – замена
- Re-architect – перепроектирование, различные варианты дизайна

Примечание: Более подробную информацию об адаптивном дизайне в приложениях UWP см. по ссылке <https://msdn.microsoft.com/en-us/library/windows/apps/dn958435.aspx>.

1. Добавьте определения строк к сетке. Удалите атрибуты фиксированной ширины столбца и используйте атрибуты x:Name. Мы будем использовать строки и столбцы, чтобы изменять положения контента в сетке.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition x:Name="LeftCol" />
        <ColumnDefinition x:Name="RightCol" />
    </Grid.ColumnDefinitions>
```

2. Создадим адаптивный триггер, который будет срабатывать на различных разрешениях экрана. Добавьте три визуальных состояния по условию **MinWindowWidth**, соответствующих разрешениям 320, 548 и 1024 точки по горизонтали.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="VisualStateGroup">
            <VisualState x:Name="VisualStateMin320">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="320"/>
                </VisualState.StateTriggers>
            </VisualState>
            <VisualState x:Name="VisualStateMin548">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="548"/>
                </VisualState.StateTriggers>
            </VisualState>
            <VisualState x:Name="VisualStateMin1024">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="1024"/>
                </VisualState.StateTriggers>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
```

Примечание: Ваше приложение UWP будет запускаться на устройствах с самыми разными экранами. Полезным подходом при работе с группами устройств со сходными, но не идентичными характеристиками, является использование некоторых опорных точек. Некоторые рекомендованные точки – 320, 720, 1024 и 320, 548, 1024 в зависимости от того, как ваш UI адаптируется через визуальные состояния. То, какие именно точки вы выберете в качестве опорных, зависит от вашего приложения и конкретного дизайна, который вы создаёте. Необходимо всегда тестировать своё приложение с помощью множества различных экранов.

Адаптивные триггеры в UWP включают **MinWindowWidth** и **MinWindowSize**. Мы будем использовать триггер **MinWindowWidth** в этом упражнении.

3. Используйте директивы **x:Name** для именования элементов (Hero и элемент StackPanel, содержащий контент метаданных), чтобы мы могли ссылаться на них в установщике **VisualState**. Расположите изображение героя в первой строке и столбце таблицы. Изображение останется в первом столбце и строке независимо от визуального состояния, таким образом, мы можем оставить его как статическую настройку. Контент метаданных будет перемещаться в зависимости от изменения ширины экрана. Удалите привязку **Grid.Column** из элемента StackPanel **MetaData**

XAML

```
<Image x:Name="Hero" Grid.Column="0" Grid.Row="0"
Source="Assets/airtime.jpg"
Stretch="UniformToFill" />
<StackPanel x:Name="Metadata" Background="LightBlue">
```

4. Добавьте установщики визуального состояния, чтобы изменить поведение столбцов. Пока что, два визуальных состояния с меньшим размером экрана будут содержать одни и те же правила. Мы реализуем разные правила для них в следующем упражнении.

XAML

```
<VisualState x:Name="VisualStateMin320">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="320"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin548">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="548"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
    </VisualState.Setters>
</VisualState>
```

Примечание: Обратите внимание на синтаксис, используемый в установщиках VisualState, для установки таких свойств как (Grid.ColumnSpan) - свойство находится в скобках.

Такие свойства называются привязанными свойствами, и они обычно используются для наследования свойств дочерними элементами, т.е. позволяют дочернему элементу установить свойства родителя. Например, дочерние элементы для Grid могут использовать привязанные свойства, чтобы указать элементам Grid.Row и Grid.Column на необходимость их размещения в нужной строке и столбце сетки.

Этот метод является важной практикой для установки свойств, определяющих макет в каждом визуальном состоянии. Хотя некоторые значения могут переноситься без изменений из одного визуального состояния в другое, в то время как пользователь меняет размер окна, при запуске приложения окно может быть любого размера, так что важные свойства должны присутствовать в каждом визуальном состоянии, чтобы получить необходимый результат.

5. Отцентрируем изображение Hero с выравниванием по вертикали и горизонтали.

Поскольку размер окна изменился, оно будет скрывать и показывать края изображения в зависимости от формата. Центрируя изображение, вы обеспечиваете большую вероятность того, что изображение останется на виду.

XAML

```
<Image x:Name="Hero" Grid.Column="0" Grid.Row="0"  
Source="Assets/airtime.jpg"  
Stretch="UniformToFill" VerticalAlignment="Center"  
HorizontalAlignment="Center" />
```

6. Добавьте перенос текста для поля описания.

XAML

```
<TextBlock Text=" Shot at Aiguille du Midi, Chamonix, France "  
TextWrapping="WrapWholeWords" />
```

7. Добавим немного свободного пространства вокруг элемента Metadata с помощью Padding.

XAML

```
<StackPanel x:Name="Metadata" Background="LightBlue" Padding="12">
```

8. Без фиксированной ширины по умолчанию каждая колонка занимает 50% ширины окна приложения. Установите для левой колонки значение 66,6%, а для правой колонки - 33,3% для визуального состояния в 1024 пикселя, чтобы обеспечить больше места для изображения.

XAML

```
<VisualState x:Name="VisualStateMin1024">  
    <VisualState.StateTriggers>  
        <AdaptiveTrigger MinWindowWidth="1024"/>  
    </VisualState.StateTriggers>  
    <VisualState.Setters>
```

```

<Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
<Setter Target="Hero.(Grid.RowSpan)" Value="2" />
<Setter Target="Metadata.(Grid.Column)" Value="1" />
<Setter Target="Metadata.(Grid.Row)" Value="0" />
<Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
<Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
<Setter Target="LeftCol.Width" Value="2*" />
<Setter Target="RightCol.Width" Value="1*" />
</VisualState.Setters>
</VisualState>
```

Примечание: Звёздочка в присвоении ширины говорит элементу Grid, что значение ширины выражено как пропорция доступного пространства. Ваш контент охватывает оба столбца в меньших визуальных состояниях, так что не требуется устанавливать правила для ширины столбцов в этих состояниях.

9. Скомпилируйте и запустите своё приложение. Когда вы уменьшите окно, метаданные будут перемещаться в первый столбец под изображение героя. При большем размере метаданные отобразятся во втором столбце.

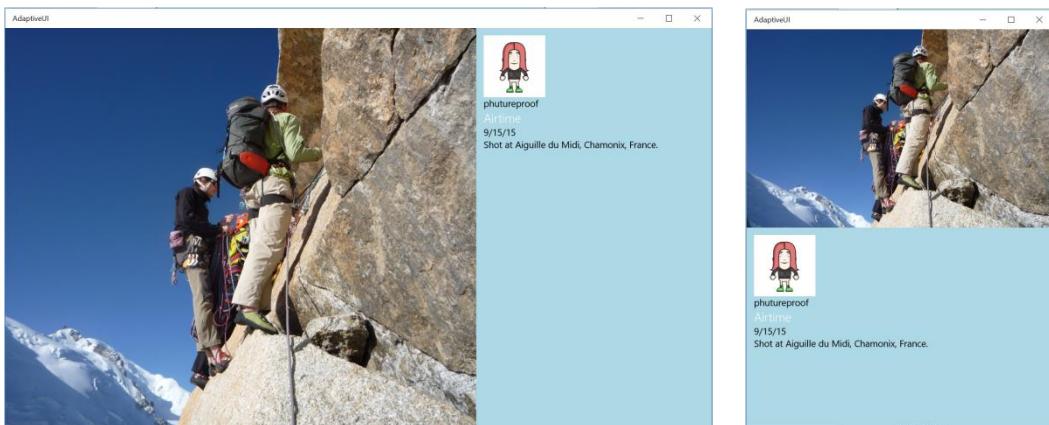


Рисунок 5

Метаданные перемещается под изображения при небольшом размере окна.

10. Остановите отладку и вернитесь к Visual Studio.

Упражнение 2: Класс RelativePanel с визуальными состояниями

В этом упражнении вы добавите класс RelativePanel, чтобы обработать семантически сгруппированный контент и ещё больше усовершенствовать отзывчивость дизайна.

Задача 1 – Добавление класса RelativePanel

Метаданные фотографии сгруппированы во всех визуальных состояниях, что делает удобным использование класса RelativePanel.

Примечание: RelativePanel позволяет вам определить расположение его дочерних элементов относительно друг друга. В то время как расположение элементов относительно друг друга могут оставаться теми же в разных визуальных состояниях, вы также можете модифицировать их для разных размеров окна. Более подробную информацию о RelativePanel см. по ссылке:

<https://msdn.microsoft.com/library/windows/apps/windows.ui.xaml.controls.relativepanel.aspx>

1. Замените элемент **StackPanel** с именем Metadata на **RelativePanel**, сохранив все прежние атрибуты без изменений.

XAML

```
<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image Source="Assets/avatar.jpg" Width="100" Height="100"
HorizontalAlignment="Left" />
    <TextBlock Text="phutureproof" />
    <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
        Text="Airtime" />
    <TextBlock Text="9/15/15" />
    <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France."
        TextWrapping="WrapWholeWords" />
</RelativePanel>
```

2. Заключите элементы TextBlock, показывающие заголовок изображения, дату и описание в в **StackPanel**. Эти блоки будут привязаны друг к другу, поэтому будет легче разместить их в StackPanel вместе, чем по отдельности.

XAML

```
<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image Source="Assets/avatar.jpg" Width="100" Height="100"
HorizontalAlignment="Left" />
    <TextBlock Text="phutureproof" />
    <StackPanel>
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
```

```

        <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France."
            TextWrapping="WrapWholeWords" />
    </StackPanel>
</RelativePanel>
```

- Присвойте атрибуты **x:Name** дочерним элементам класса **RelativePanel**. Эти элементы будут изменять свое расположение относительно друг друга.

XAML

```

<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100"
    Height="100"
        HorizontalAlignment="Left" />
    <TextBlock x:Name="Username" Text="phutureproof" />
    <StackPanel x:Name="Description" >
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France."
            TextWrapping="WrapWholeWords" />
    </StackPanel>
</RelativePanel>
```

- Используйте **RelativePanel.Below** и **RelativePanel.AlignHorizontalCenterWith** для размещения имени пользователя относительно аватара пользователя. Положение этих элементов относительно друг друга будет оставаться таким же во всех визуальных состояниях.

XAML

```

<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100"
    Height="100"
        HorizontalAlignment="Left" />
    <TextBlock x:Name="Username" RelativePanel.Below="Avatar"
        RelativePanel.AlignHorizontalCenterWith="Avatar"
    Text="phutureproof"
        />
    <StackPanel x:Name="Description" >
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text=" Shot at Aiguille du Midi, Chamonix, France."
            TextWrapping="WrapWholeWords" />
    </StackPanel>
</RelativePanel>
```

- Добавьте установщики визуальных состояний, чтобы определить размещение описания, связанного с аватаром пользователя. Для состояний **VisualStateMin1024** и **VisualStateMin548** есть свободное пространство для отображения описания справа от аватара пользователя. Для состояния **VisualStateMin320** описание можно поместить под

изображением. Также будем изменять границы (margins) для лучшего визуального восприятия.

XAML

```
<VisualState x:Name="VisualStateMin320">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="320"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="" />
        <Setter Target="Description.(RelativePanel.Below)" Value="Username" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin548">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="548"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
        <Setter Target="LeftCol.Width" Value="2*" />
        <Setter Target="RightCol.Width" Value="1*" />
    </VisualState.Setters>
</VisualState>
```

```

        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar"
/>
        <Setter Target="Description.(RelativePanel.Below)" Value="" />
        <Setter Target="Description.Margin" Value="12,0,0,0" />
    </VisualState.Setters>
</VisualState>
```

Примечание: В некоторых визуальных состояниях свойства класса RelativePanel устанавливаются равными пустой строке. Пустая строка стирает настройки, которые могут иначе сохраняться от других визуальных состояний. Без этих установщиков некоторые нежелательные относительные положения могут наследоваться различными визуальными состояниями.

6. В дополнение к внесению изменений в расположение элементов макета, установщики в визуальном состоянии могут переопределять такие стили, как размер шрифта и отступ, для лучшего отображения на различных устройствах. Удалите атрибут шрифта из текстового блока TextBlock, отображающего имя изображения, и добавьте директиву **x:Name**, чтобы позволить вам обращаться к данному элементу.

XAML

```
<TextBlock x:Name="ImageName" Foreground="White" FontWeight="Light"
Text="Airtime" />
```

7. Добавьте установщики для изменения размера шрифта в зависимости от VisualState.

XAML

```

<VisualState x:Name="VisualStateMin320">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="320"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="" />
        <Setter Target="Description.(RelativePanel.Below)" Value="Username" />
        <Setter Target="Description.Margin" Value="0,12,0,0" />
        <Setter Target="ImageName.FontSize" Value="20" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin548">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="548"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
```

```

<Setter Target="Hero.(Grid.RowSpan)" Value="1" />
<Setter Target="Metadata.(Grid.Column)" Value="0" />
<Setter Target="Metadata.(Grid.Row)" Value="1" />
<Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
<Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
<Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar" />
</VisualState.Setters>
</VisualState>
<VisualState x:Name="ViewStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
        <Setter Target="LeftCol.Width" Value="2*" />
        <Setter Target="RightCol.Width" Value="1*" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar" />
        </VisualState.Setters>
    </VisualState>
    <Setter Target="Description.(RelativePanel.Below)" Value="" />
    <Setter Target="Description.Margin" Value="12,0,0,0" />
    <Setter Target="ImageName.FontSize" Value="20" />
</VisualState>

```

8. Скомпилируйте и запустите своё приложение. Измените размер окна, чтобы понаблюдать, как меняется макет с классом `RelativeLayout`.

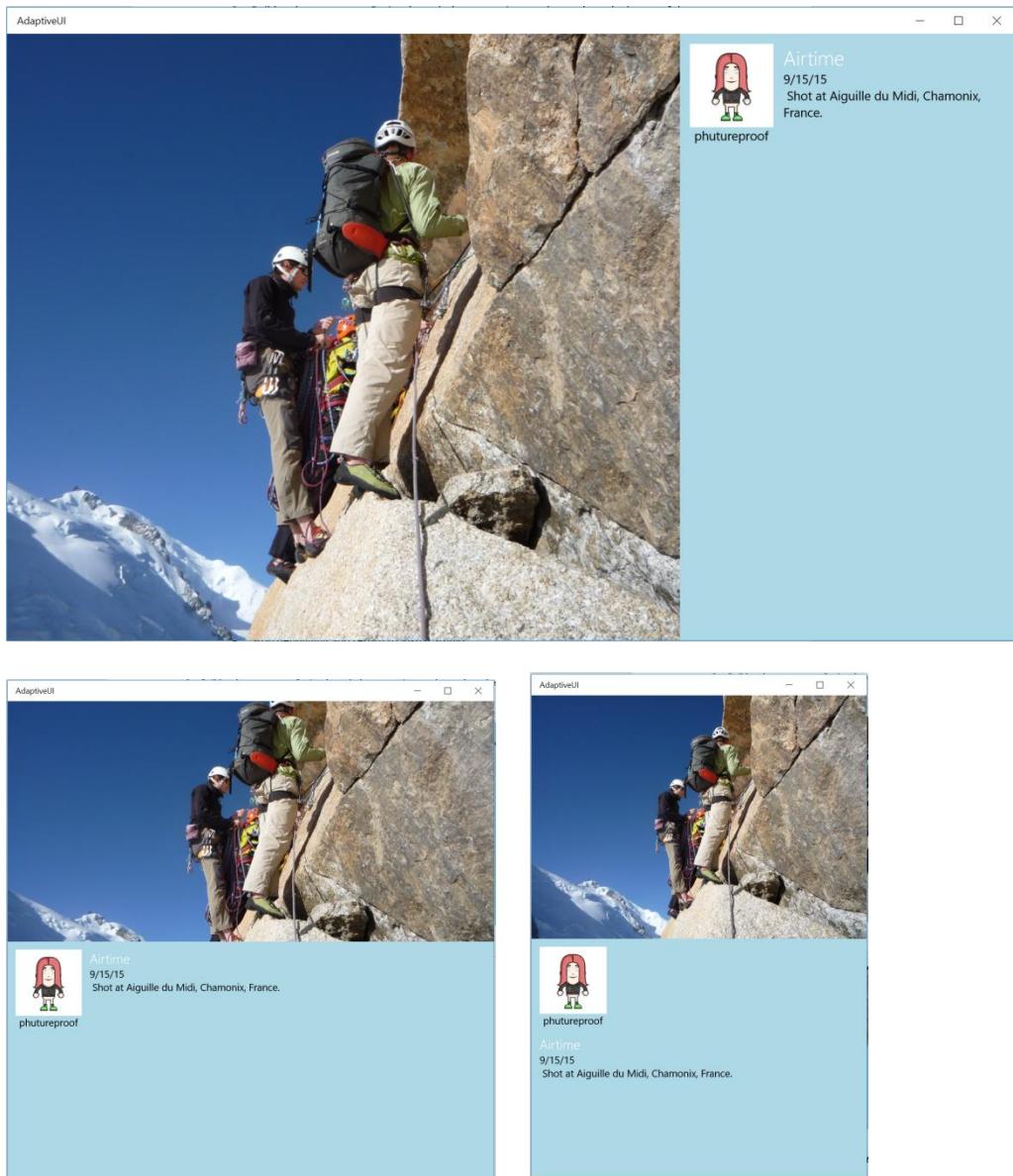


Рисунок 6

Контент с использованием класса RelativePanel перемещается внутри окна приложения при изменении размера.

9. Остановите отладку и вернитесь к Visual Studio.

Упражнение 3: Адаптивный пользовательский интерфейс на основе XAML Views

В некоторых случаях не имеет смысла использовать адаптивный дизайн для различных размеров экрана, и может быть проще начать с отдельных разметок XAML для каждого типа устройств. Виды XAML (XAML Views) позволяют вам реализовать такой подход. В этом упражнении мы добавим функцию скроллинга в приложение и создадим отдельный вид главной страницы MainPage XAML для мобильного устройства.

Задача 1 – Добавление ScrollViewer в панель метаданных

Прежде чем мы будем создавать новый вид XAML, давайте понаблюдаем, как ScrollViewer будет вести себя на существующей странице MainPage.

- Добавьте к классу **RelativePanel** с именем Metadata **ScrollViewer**, и перенесите директиву **x:Name** для метаданных в ScrollViewer.

XAML

```
<ScrollViewer x:Name="Metadata" VerticalScrollBarVisibility="Auto">
    <RelativePanel Background="LightBlue" Padding="12">
        <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100"
Height="100" HorizontalAlignment="Left" />
        <TextBlock x:Name="Username" Text="phutureproof"
            RelativePanel.Below="Avatar"
            RelativePanel.AlignHorizontalCenterWith="Avatar" />
        <StackPanel x:Name="Description">
            <TextBlock x:Name="ImageName" Foreground="White"
                FontWeight="Light" Text="Airtime" />
            <TextBlock Text="9/15/15" />
            <TextBlock Text=" Shot at Aiguille du Midi, Chamonix, France."
                TextWrapping="WrapWholeWords" />
        </StackPanel>
    </RelativePanel>
</ScrollViewer>
```

- Скопируйте более длинную строку в подпись изображения.

XAML

```
<TextBlock Text="Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Nulla imperdiet pulvinar nunc. In et gravida ipsum. Morbi congue consequat  
ullamcorper. Integer ornare porta convallis. Pellentesque habitant morbi  
tristique senectus et netus et malesuada fames ac turpis egestas. Vivamus  
sem nisi, ornare vel laoreet vel, accumsan facilisis tortor. Pellentesque  
ut nunc in leo vehicula pretium et at quam. Aliquam euismod id purus nec  
ultrices. Aliquam sed nisl at erat maximus finibus in sed urna. Nulla  
ullamcorper vehicula ex, in porta ante ullamcorper id. Phasellus a enim  
vitae odio ultricies semper. Suspendisse fermentum, erat in sodales  
accumsan, lacinia urna aliquam nisi, sed ultricies dolor orci quis ligula."  
TextWrapping="WrapWholeWords" />
```

3. Скомпилируйте и запустите своё приложение. Измените размер окна приложения до высоты, при которой появляется полоса прокрутки. Полоса прокрутки появляется только для метаданных, что означает, что второй столбец прокручивается при большем размере окна, а вторая строка прокручивается при меньшем размере окна.

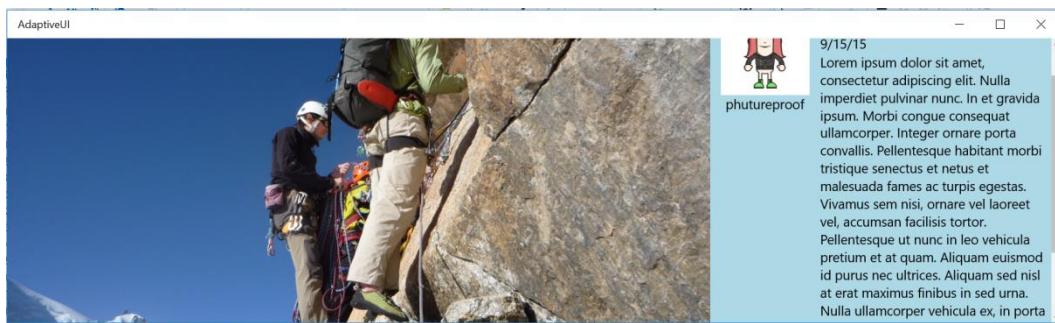


Рисунок 7

Метаданные прокручиваются, в то время как изображение остаётся на месте.

4. Остановите отладку и вернитесь в Visual Studio.

Задача 2 – Создание вида XAML

Использование прокрутки для меньших визуальных состояний - неидеальное решение. Гораздо лучше было бы, если вся страница прокручивалась на мобильных устройствах. Мы могли бы обернуть Grid в ScrollViewer, но тогда изображение героя также прокручивалось бы при большом размере окна. В этой задаче мы создадим отдельный вид для мобильного устройства, чтобы было удобнее управлять прокруткой, не меняя порядок работы приложения на настольном компьютере.

1. Щелкните правой кнопкой мыши на Solution Explorer (Обозреватель решений). Затем **Add (Добавить) > New Folder (Новая папка)**. Присвойте папке имя **DeviceFamily-Mobile**.

Примечание: Папка и вид должны называться определенным образом, чтобы XAML View правильно работал. Папки должны называться **DeviceFamily-FamilyName**, а страницы внутри должны иметь такие же имена, как и исходные страницы, которые они будут заменять.

Кроме того, вы можете добавить файлы прямо в ту же папку, где находятся исходные страницы XAML, и присвоить им следующие имена **OriginalPage.DeviceFamily-FamilyName.xaml**.

- Щёлкнитесь правой кнопкой мыши на папке и выберите **Add (добавить) > New Item (Новый элемент)** для добавления **вида XAML (XAML View)**. Назовите его **MainPage.xaml**.

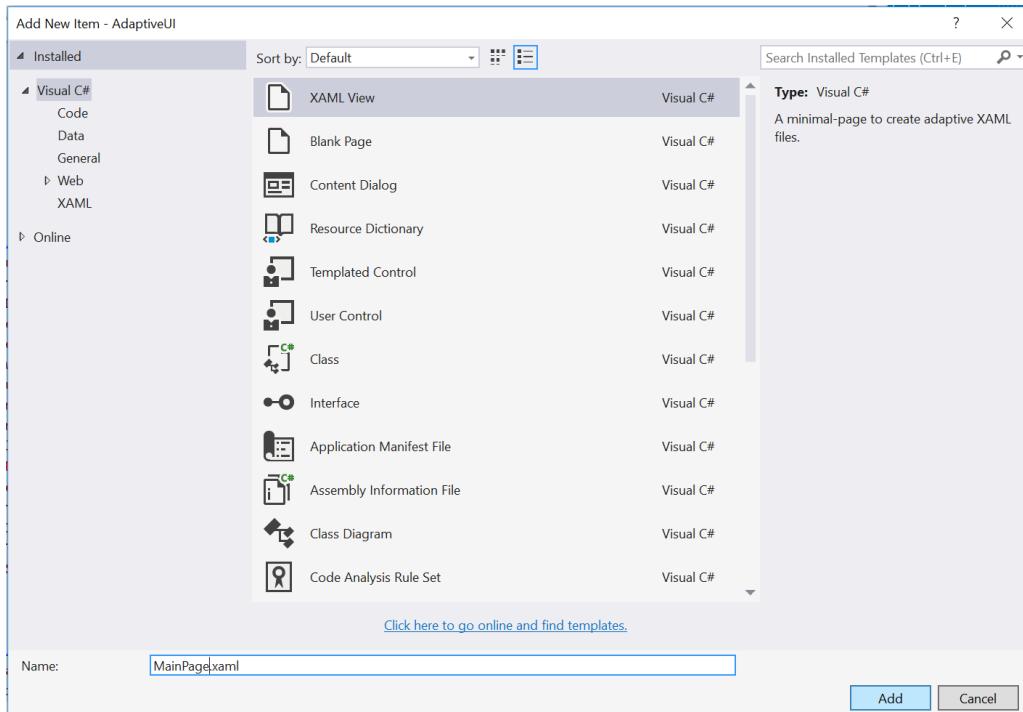


Рисунок 8
Добавление вида XAML.

Примечание: Вид XAML отличается от пустой страницы тем, что он не генерирует файл с C#-кодом. Мы будем использовать такой же выделенный код MainPage.xaml.cs для мобильных устройств и настольного компьютера, так что мы создадим вид XAML вместо страницы.

- Скопируйте содержание своего исходного файла **MainPage.xaml** в новый файл **MainPage.xaml**.
- В новом файле **MainPage.xaml** удалите все визуальные состояния и **VisualStateManager**.
- Удалите определения столбцов и привязку столбцов.
- Удалите атрибуты **метаданных x:Name** и переместите класс **ScrollViewer** за пределы кода **Grid**.

XAML

```
<ScrollViewer VerticalScrollBarVisibility="Auto">
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
```

```

<Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
</Grid.RowDefinitions>
<Image x:Name="Hero" Source="Assets/airtime.jpg"
    Stretch="UniformToFill" HorizontalAlignment="Center"
    VerticalAlignment="Center" />
<RelativePanel Background="LightBlue" Padding="12">
    <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100"
        Height="100" HorizontalAlignment="Left" />
    <TextBlock x:Name="Username" Text="phutureproof"
        RelativePanel.Below="Avatar"
        RelativePanel.AlignHorizontalCenterWith="Avatar" />
    <StackPanel x:Name="Description">
        <TextBlock x:Name="ImageName" Foreground="White"
            FontWeight="Light" Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text="Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Nulla imperdiet pulvinar nunc. In et gravida ipsum. Morbi
congue consequat ullamcorper. Integer ornare porta convallis. Pellentesque
habitante morbi tristique senectus et netus et malesuada fames ac turpis
egestas. Vivamus sem nisi, ornare vel laoreet vel, accumsan facilisis
tortor. Pellentesque ut nunc in leo vehicula pretium et at quam. Aliquam
euismod id purus nec ultrices. Aliquam sed nisl at erat maximus finibus in
sed urna. Nulla ullamcorper vehicula ex, in porta ante ullamcorper id.
Phasellus a enim vitae odio ultricies semper. Suspendisse fermentum, erat
in sodales accumsan, lacus urna aliquam nisi, sed ultricies dolor orci quis
ligula."/>
            TextWrapping="WrapWholeWords" />
    </StackPanel>
</RelativePanel>
</Grid>
</ScrollView>

```

- Добавьте привязку **Grid.Row** для изображения **Hero** и класса **RelativePanel**.

XAML

```

<Image x:Name="Hero" Grid.Row="0" Source="Assets/airtime.jpg"
    Stretch="UniformToFill" HorizontalAlignment="Center"
    VerticalAlignment="Center" />
<RelativePanel Grid.Row="1" Background="LightBlue" Padding="12">

```

- Добавьте размер шрифта как атрибут для элемента **ImageName**.

XAML

```

<TextBlock x:Name="ImageName" FontSize="20" Foreground="White"
    FontWeight="Light" Text="Airtime" />

```

- Присвойте для **RelativePanel.Below="Username"** в блок для текста **Description**.

XAML

```

<StackPanel x:Name="Description" RelativePanel.Below="Username">

```

10. Запустите свое приложение в эмуляторе мобильной платформы. Нажмите на контент, чтобы увидеть полосу прокрутки. И изображение, и метаданные будут прокручиваться.



Рисунок 9

Адаптивный мобильный вид приложения, запущенное на эмуляторе мобильной платформы.

11. Остановите отладку и вернитесь к Visual Studio.

12. Вы также можете запустить приложение на локальном компьютере. Вы увидите, что первоначальный вид будет использоваться при запуске приложения на локальном компьютере.

Краткий обзор

Адаптивный интерфейс является важной частью дизайна приложений UWP. В этом курсе мы следовали принципам адаптивного дизайна для перемещения и обновления контента для его лучшего отображения на различных экранах. Мы также создали отдельный вид для мобильного устройства, чтобы обеспечить новые возможности, недоступные для адаптивного дизайна.



Лабораторный практикум

Живые плитки и уведомления

Октябрь 2015 г.



Обзор

Windows 10 предоставляет уникальные и привлекательные способы для взаимодействия с пользователями, которые выходят за рамки привычных приложений. Теперь вы можете задать контент для «Живой плитки» посредством использования новой адаптивной разметки, которая позволит максимально эффективно учитывать размер плитки и плотность экрана. Всплывающие уведомления (Toast) могут включать интерактивные и адаптивные элементы, изображения, а также способны синхронизироваться с живыми плитками.

Цели

Данная лабораторная работа научит вас:

- Задавать графические элементы и фоновый цвет для плиток по умолчанию
 - Использовать BadgeUpdateManager для обновления счетчика плитки
 - Создавать адаптивные шаблоны для живых плиток
 - Обновлять живые плитки
 - Поддерживать плитки разного размера
 - Создавать интерактивные уведомления с возможностью для пользователя выбирать действие
 - Обрабатывать действия уведомлений с помощью фоновой активации приложения
-

Системные требования

Для прохождения данной лабораторной работы вам понадобятся:

- Microsoft Windows 10
 - Microsoft Visual Studio 2015
-

Настройка

Вы должны выполнить следующие действия для подготовки компьютера:

1. Установить Microsoft Windows 10.
 2. Установить Microsoft Visual Studio 2015.
-

Упражнения

Данный практикум включает следующие упражнения:

1. Настройка плитки по умолчанию
 2. Управление обновлениями плиток
 3. Запуск интерактивного всплывающего уведомления
-

Расчетное время для завершения лабораторной работы: **45-60 минут.**

Упражнение 1: Настройка плитки по умолчанию

Приложения Windows 10 используют выбранный вами логотип (изображение) для отображения плитки по умолчанию. В данном упражнении мы добавим новый логотип (изображение) с целью создания простых плиток и обновления счетчика (бейджа) плитки при использовании BadgeUpdateManager.

Задача 1 – Создание пустого универсального Windows-приложения

Мы начнем с создания проекта на основе шаблона Blank App (Пустого приложения).

1. В новой версии Visual Studio 2015 выберите **File (Файл) -> New (Новый) -> Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed (Установленное) > Templates (Шаблоны) > Visual C# > Windows > Universal**, а затем выберите шаблон **Blank App (Universal Windows)**.
2. Назовите свой проект "**TilesAndNotifications**" и выберите место в файловой системе, в котором будет храниться ваш проект для данного Лабораторного практикума. На диске **C:** создана папка под именем "**HOL**", мы отсылаемся на нее в примерах снимков экрана.

Оставьте без изменения выбранные опции для **Create new solution** (Создание нового решения) и **Create directory for solution** (Создание папки для решения). Вы можете снять галочки с **Add to source control** (Добавить в систему контроля версий) и **Show telemetry in the Windows Dev Center** (Отобразить телеметрию в Windows Dev Center), если не хотите управлять версиями своего проекта или использовать инструмент Application Insights. Нажмите **OK** для создания проекта.

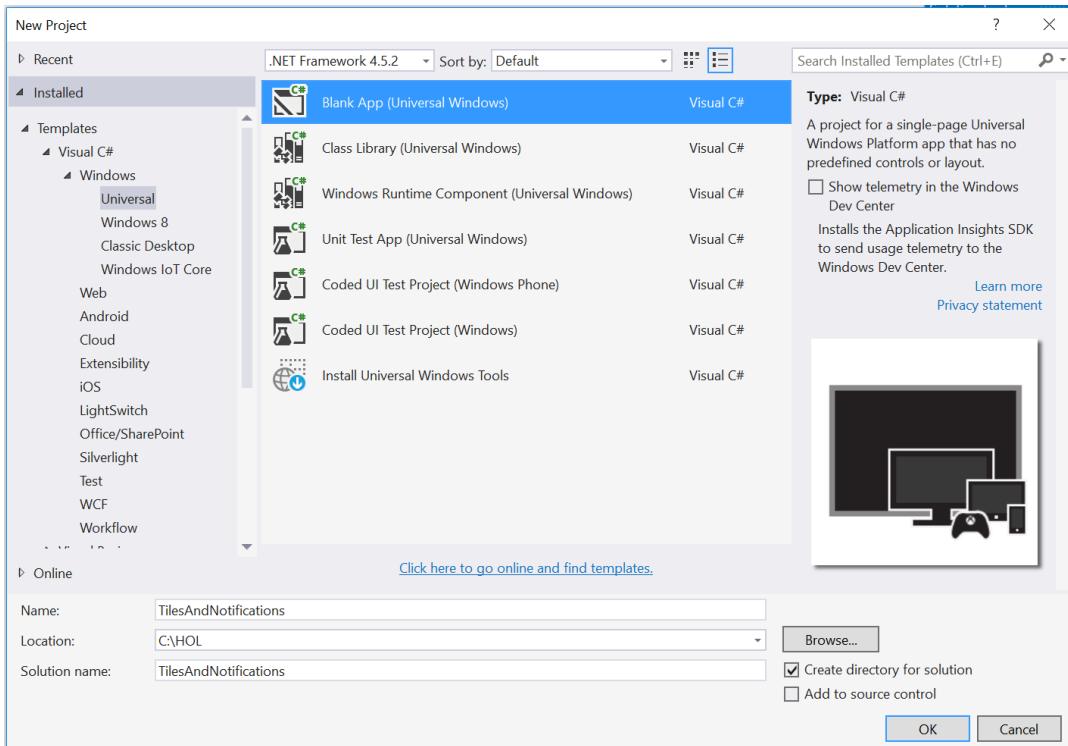


Рисунок 1

Создайте в Visual Studio 2015 новый проект Blank App.

- Настройте Solution Configuration (Текущую конфигурацию решения) на **Debug** (Отладку) и Solution Platform (Платформу решений) в соответствии с **x86**. Выберите **Local Machine** (Локальный компьютер) из выпадающего меню Debug Target (Цели отладки).

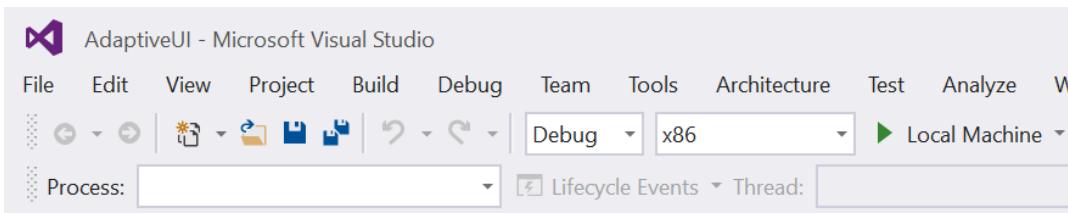


Рисунок 2

Настройте свое приложение таким образом, чтобы оно запускалось на Local Machine (Локальном компьютере).

- Создайте и запустите свое приложение. Вы увидите окно Blank App со счетчиком частоты кадров, активированным по умолчанию для отладки.

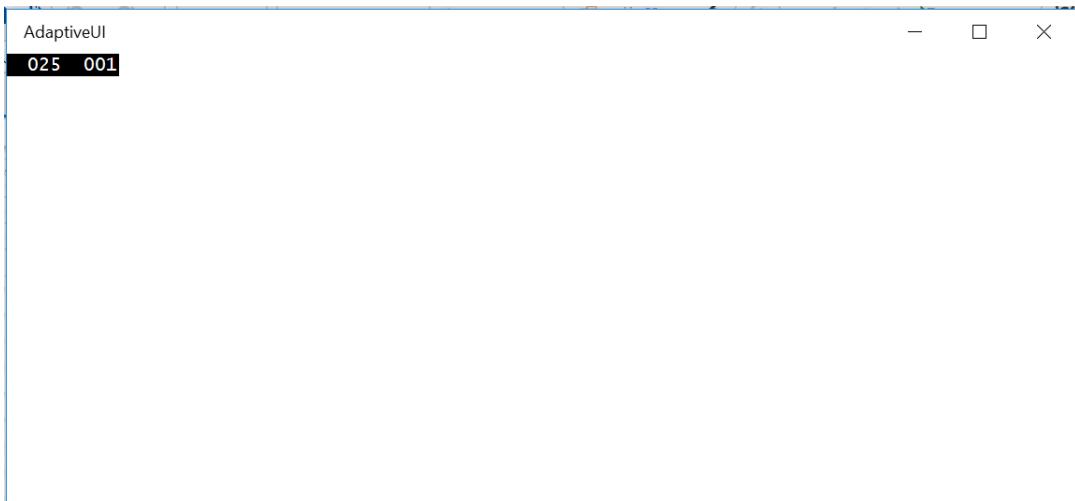


Рисунок 3

Пустое универсальное приложение в десктопном режиме.

Примечание: Счетчик частоты кадров является инструментом отладки, который помогает следить за производительностью вашего приложения. Он полезен для тех приложений, которые требуют интенсивной графической обработки, однако не подходит для простых приложений, которые будут создаваться вами на данный момент.

В шаблоне Blank App директива препроцессора, которая активирует или отключает счетчик частоты кадров, указана в **App.xaml.cs**. Счетчик частоты кадров может перекрывать или скрывать контент вашего приложения, если его не отключить. При выполнении данной работы вы можете отключить его, отметив **DebugSettings.EnableFrameRateCounter** как **false**.

Вернитесь к Visual Studio и остановите отладку.

Задача 2 – Импорт визуальных элементов

Когда вы создаете новый проект, он включает папку Assets с шаблонными изображениями плиток. Чтобы выделить свое приложение, вам нужно заменить шаблонные изображения, представленные в шаблоне Blank App, на собственные логотипы. Визуальные элементы можно указывать с учетом различных коэффициентов масштаба для экранов с разной плотностью пикселей. Если вы указываете изображения только для одного масштаба, то рекомендуется выбирать изображение в масштабе 200 (200% от исходного размера для типичного устройства с 96 dpi), что должно привести к хорошим результатам, поскольку Windows масштабирует изображения, подстраиваясь под плотности пикселей экранов различных устройств. Для достижения наилучших результатов при различной плотности экрана вам стоит предоставить изображения, созданные для нескольких масштабных коэффициентов.

В рамках данной задачи мы добавим в приложение изображения для масштаба 200 с целью их отображения на плитках по умолчанию.

1. Откройте **Package.appxmanifest** в редакторе манифеста и перейдите на вкладку **Visual Assets**. Выберите пункт **Tile Images and Logos** на панели слева.

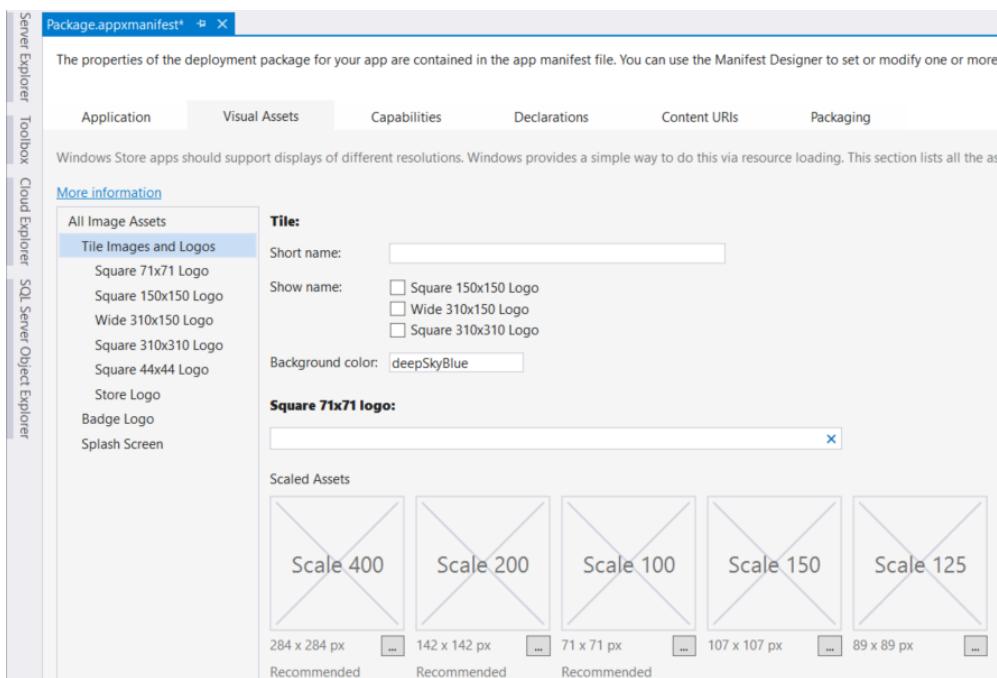


Рисунок 4

Выбор пункта Tile Images and Logos на вкладке Visual Assets в редакторе манифеста.

2. В поле Background color (Цвет фона) введите "deepskyblue".

Примечание: Если вы оставите поле Background прозрачным, ваши плитки унаследуют акцентный цвет, выбранный пользователем для системы в настройках **Windows 10 Settings (Настройки Windows 10) > Personalization (Персонализация) > Colors (Цвета)**. В этом упражнении мы зададим конкретный цвет фона, чтобы было легче видеть изображения плиток в редакторе манифеста.

3. В разделе **Square71x71Logo** используйте символ многоточия под изображением для масштаба 200 (**scale 200**), чтобы открыть диалоговое окно **Select Image** (Выбор изображения). Перейдите в папку **Lab Assets** и выберите файл **Square71x71Logo.scale-200.png**. Нажмите **Open**, чтобы добавить изображение в качестве логотипа.

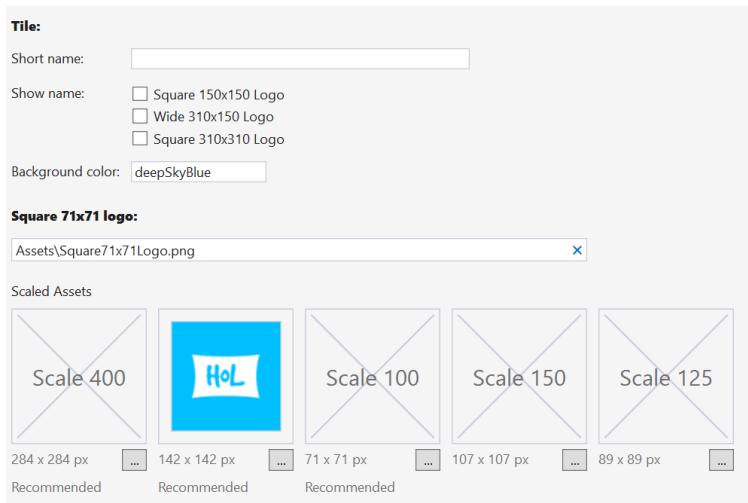


Рисунок 5

Изображение для Square71x71Logo в редакторе манифеста.

4. Повторите **шаг 3**, чтобы добавить изображения **Square150x150Logo.scale-200.png**, **Wide310x150Logo.scale-200.png**, **Square310x310Logo.scale-200.png** и **Square44x44Logo.scale-200.png**. Когда дойдете до **StoreLogo.png**, добавьте изображение в **масштабе 100**. В каждом случае, если вы заменяете одно из шаблонных изображений, существующих в проекте, Visual Studio уточнит у вас, действительно ли вы хотите заменить. Нажмите Yes.
5. Выберите пункт **Splash Screen** в панели слева. В настройках экрана загрузки установите цвет фона также в значение **deepskyblue**. Используйте кнопку с многоточием под масштабом 200, чтобы выбрать **SplashScreen.scale-200.png** и папки Lab Assets.
6. Соберите и запустите приложение. После разворачивания, найдите приложение в меню Пуск (Start). Щелкните правой кнопкой по имени приложения и выберите **Pin to Start** (**закрепить на экране Пуск**).

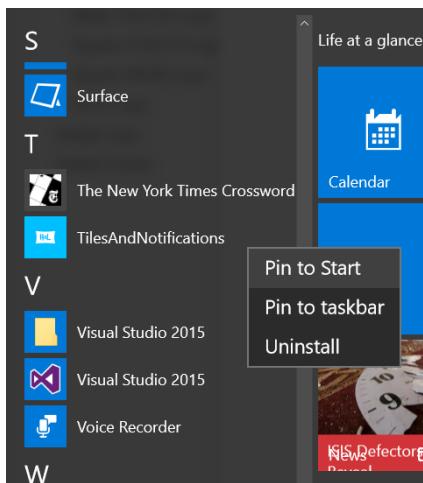


Рисунок 6

Закрепление приложения в меню Пуск.

7. Ваша плитка по умолчанию появится в меню Пуск уже с ваши логотипом и соответствующим цветом фона, указанными в манифесте. Нажмите правой кнопкой мыши на плитке и поменяйте размер, чтобы посмотреть, как плитка масштабируется.

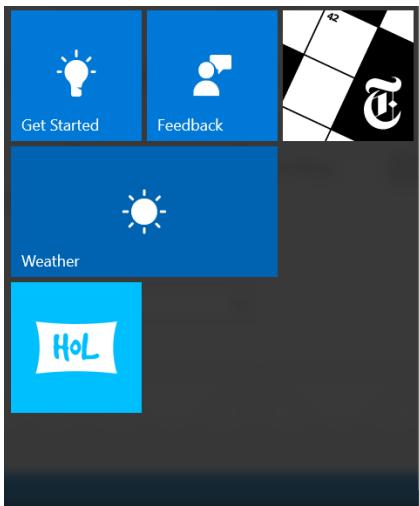


Рисунок 7

Плитка закрепленного приложения

8. Остановите отладку и вернитесь в Visual Studio.

Задача 3 – Обновление счетчика на плитке

Счетчик на плитке – это отличный способ для отображения информации, которую можно было бы легко считать взглядом, например, количество новых элементов в приложении. Мы посмотрим, как обновить бейдж со счетчиком для плитки по умолчанию, это быстрый и простой способ добавить дополнительную ценность присутствию вашего приложения на экране Пуск.

1. Щёлкните правой кнопкой мыши по имени проекта и выберите **Add (Добавить) > New Folder (Новая папка)**. Назовите папку **Services**.
2. Щелкните правой кнопкой мыши по папке и выберите **Add (Добавить) > Class (Класс)**. Назовите файл с классом **TileService.cs**.
3. Откройте файл **TileService.cs** и сделайте класс публичным (**public**).

```
C#  
namespace TilesAndNotifications.Services  
{  
    public class TileService  
    {  
    }  
}
```

4. Добавьте статичный метод для обновления счетчика на плитке. Добавьте с помощью `using` ссылки на `Windows.Data.Xml.Dom` и `Windows.UI.Notifications`.

C#

```
using Windows.Data.Xml.Dom;
using Windows.UI.Notifications;

namespace TilesAndNotifications.Services
{
    public class TileService
    {
        static public void SetBadgeCountOnTile(int count)
        {
            // Update the badge on the real tile
            XmlDocument badgeXml =
BadgeUpdateManager.GetTemplateContent(BadgeTemplateType.BadgeNumber);

            XmlElement badgeElement =
(XmlElement)badgeXml.SelectSingleNode("//badge");
            badgeElement.SetAttribute("value", count.ToString());

            BadgeNotification badge = new BadgeNotification(badgeXml);

            BadgeUpdateManager.CreateBadgeUpdaterForApplication().Update(badge);
        }
    }
}
```

5. Вернитесь в файл `MainPage.xaml`. Создайте кнопку для обновления счетчика

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Button Click="UpdateBadge" VerticalAlignment="Top" Margin="12">Update
    Badge Count</Button>
</Grid>
```

6. В коде для `MainPage.xaml.cs` добавьте свойство `_count` типа `int` и метод `UpdateBadge()` для вызова сервиса `TileService`. Добавьте пространство имен `TilesAndNotifications.Services`.

C#

```
public MainPage()
{
    this.InitializeComponent();
}
int _count;
private void UpdateBadge (object sender, RoutedEventArgs e)
{
    _count++;
    TileService.SetBadgeCountOnTile(_count);
}
```

- Соберите и запустите приложение на локальной машине. После развертывания снова найдите ваше приложение в меню Пуск.

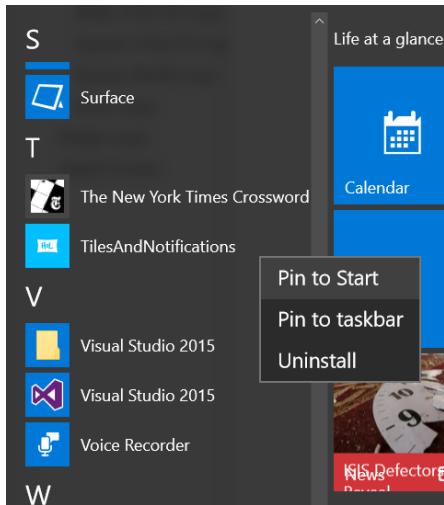


Рисунок 8

Закрепление приложения (если вы его открепили).

- В запущенном приложении нажмите на кнопку **Update Badge Count**. Когда вы вернетесь к свое живой плитке на экране Пуск, вы увидите, что значение счетчика обновилось до 1. Этот счетчик будет увеличиваться каждый раз, когда вы вызываете метод **UpdateBadge()**.
- Остановите отладку и вернитесь в Visual Studio.

Упражнение 2: Создание адаптивных живых плиток

Живые плитки в Windows 10 используют адаптивные шаблоны для показа контента с учетом параметров устройства и плотности экрана. Хотя шаблоны прежних версий все еще совместимы с живыми плитками, адаптивные шаблоны предоставляют вам большую свободу в плане выбора, как контент будет отображаться на разных устройствах. Группы и подгруппы позволяют вам семантически связывать контент в пределах плитки. В настоящем упражнении мы создадим адаптивный макет и используем его для показа статичных данных на плитке.

Задача 1 – Добавление модели

В типичном приложении живая плитка отображает имеющиеся в самом приложении данные. В рамках данной задачи мы создадим класс как набросок статических данных с целью их отображения на плитках.

- Щелкните правой кнопкой мыши по имени проекта и создайте папку **Models**.
- Добавьте класс **PrimaryTile.cs** в папку Models.
- Откройте класс **PrimaryTile** и сделайте его публичным. Добавьте статичные данные в виде строковых полей, чтобы задать данные, которыми мы будем заполнять плитки.

C#

```
namespace TilesAndNotifications.Models
{
    public class PrimaryTile
    {
        public string time { get; set; } = "8:15 AM, Saturday";
        public string message { get; set; } = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.";
        public string message2 { get; set; } = " At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.";
        public string branding { get; set; } = "name";
        public string appName { get; set; } = "HoL";
    }
}
```

Задача 2 – Построение XML-описания плитки

Адаптивное описание плитки задается в XML. В рамках настоящей задачи мы сгенерируем XML, необходимый для отображения текстового контента из PrimaryTile на маленькой и средней живых плитках.

- В файле **TileService.cs** добавьте пространства имён **TilesAndNotifications.Models** и **System.Xml.Linq**.

C#

```
using TilesAndNotifications.Models;
using System.Xml.Linq;
```

- Добавьте метод **CreateTiles** для генерации XML под маленькую и среднюю плитки.

C#

```
public static Windows.Data.Xml.Dom.XmlDocument CreateTiles (PrimaryTile primaryTile)
{
    XDocument xDoc = new XDocument(
        new XElement("tile", new XAttribute("version", 3),
            new XElement("visual",
                // Small Tile
                new XElement("binding", new XAttribute("branding",
primaryTile.branding), new XAttribute("displayName", primaryTile.appName),
new XAttribute("template", "TileSmall"),
                    new XElement("group",
                        new XElement("subgroup",
```

```

        new XElement("text", primaryTile.time, new
XAttribute("hint-style", "caption")),
            new XElement("text", primaryTile.message, new
XAttribute("hint-style", "captionsubtle"), new XAttribute("hint-wrap",
true), new XAttribute("hint-maxLines", 3))
        )
    )
),
// Medium Tile
new XElement("binding", new XAttribute("branding",
primaryTile.branding), new XAttribute("displayName", primaryTile.appName),
new XAttribute("template", "TileMedium"),
new XElement("group",
new XElement("subgroup",
new XElement("text", primaryTile.time, new
XAttribute("hint-style", "caption")),
new XElement("text", primaryTile.message, new
XAttribute("hint-style", "captionsubtle"), new XAttribute("hint-wrap",
true), new XAttribute("hint-maxLines", 3)))
)
)
)
);
Windows.Data.Xml.Dom.XmlDocument xmlDoc = new
Windows.Data.Xml.Dom.XmlDocument();
xmlDoc.LoadXml(xDoc.ToString());
return xmlDoc;
}

```

Примечание: Существует несколько элементов, которые вы можете включить в свою схему адаптивной плитки. Для каждого типа элементов вы можете выбрать один из предопределенных стилей. Дополнительная информация доступна в документации <https://msdn.microsoft.com/en-us/library/windows/apps/Mt186446.aspx>.

Дополнительные инструкции и примеры можно найти в статье: [Схема адаптивных плиток и документация](#).

3. Откройте MainPage.xaml. Добавьте новую кнопку после кнопки обновления счетчика, она будет обновлять основную плитку. Мы расположим кнопки в панели StackPanel, чтобы облегчить компоновку.

XAML

```

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel>
        <Button Click="UpdateBadge" VerticalAlignment="Top"
Margin="12">Update Badge Count</Button>

```

```

        <Button Click="UpdatePrimaryTile" VerticalAlignment="Top"
Margin="12">Update Primary Tile</Button>
    </StackPanel>
</Grid>

```

4. В файле кода для MainPage добавьте ссылки на пространства имен **TilesAndNotifications.Models** и **Windows.UI.Notifications**.

C#

```

using TilesAndNotifications.Models;
using Windows.UI.Notifications;

```

5. Добавьте метод **UpdatePrimaryTile()** в файле MainPage.xaml.cs.

C#

```

private void UpdatePrimaryTile(object sender,
Windows.UI.Xaml.RoutedEventArgs e)
{
    var xmlDoc = TileService.CreateTiles(new PrimaryTile());

    var updater = TileUpdateManager.CreateTileUpdaterForApplication();
    TileNotification notification = new TileNotification(xmlDoc);
    updater.Update(notification);
}

```

Примечание: Каждый раз обновляя плитку, вы на самом деле создаете новый экземпляр такой плитки. При создании плитки в приложении с динамичными данными, вы можете передать наиболее актуальные сведения для отображения на плитке.

6. Соберите и запустите приложение. Закрепите приложение на экране Пуск, если это еще не сделано.
7. Нажмите кнопку **Update Primary Tile** в запущенном приложении, чтобы начать обновление плитки. Откройте меню Пуск на вашем устройстве и подождите, пока изображение плитки по умолчанию обновит свой вид. Измените размер плитки, чтобы убедиться, что обновление затрагивает как маленькую, так и среднюю плитки.

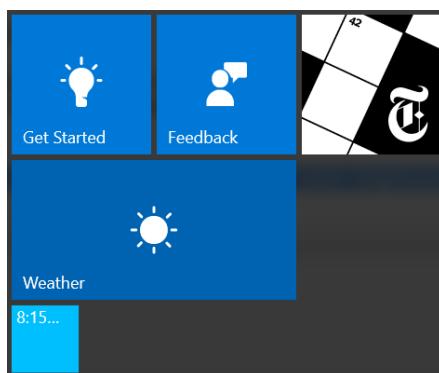


Рисунок 9

Маленькая живая плитка.

8. Измените размер плитки на **средний**. Обратите внимание, что теперь при обновлении плитки показывается больше информации из адаптивного шаблона.

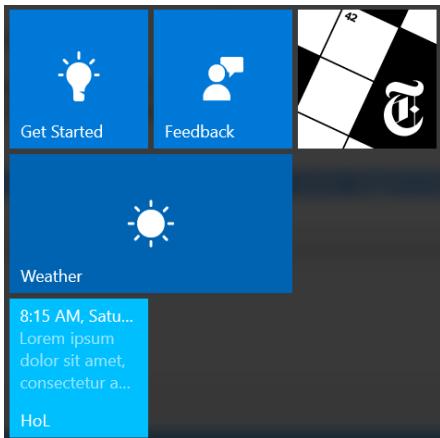


Рисунок 10
Средняя живая плитка.

9. Остановите отладку и вернитесь в Visual Studio.

Задача 3 – Создание адаптивного шаблона для широкой и большой плиток.

- Добавьте широкую и большую плитки к методу `CreateTiles()`. Плитки большего размера имеют дополнительное пространство для отображения помимо текста также и изображений. В нашем случае мы покажем изображение, уже присутствующее в папке Assets. Убедитесь, что добавили запятую после кода для создания XML-описания средней плитки, чтобы продолжить список.

```
C#
),
// Wide Tile
new XElement("binding", new XAttribute("branding", primaryTile.branding),
new XAttribute("displayName", primaryTile.appName), new
XAttribute("template", "TileWide"),
    new XElement("group",
        new XElement("subgroup",
            new XElement("text", primaryTile.time, new XAttribute("hint-
style", "caption")),
            new XElement("text", primaryTile.message, new XAttribute("hint-
style", "captionsubtle"), new XAttribute("hint-wrap", true), new
XAttribute("hint-maxLines", 3)),
            new XElement("text", primaryTile.message2, new
XAttribute("hint-style", "captionsubtle"), new XAttribute("hint-wrap",
true), new XAttribute("hint-maxLines", 3))
        ),
        new XElement("subgroup", new XAttribute("hint-weight", 15),
            new XElement("image", new XAttribute("placement", "inline"),
new XAttribute("src", "Assets/StoreLogo.png"))
        )
)
```

```

        )
),

//Large Tile
new XElement("binding", new XAttribute("branding", primaryTile.branding),
new XAttribute("displayName", primaryTile.appName), new
XAttribute("template", "TileLarge"),
    new XElement("group",
        new XElement("subgroup",
            new XElement("text", primaryTile.time, new XAttribute("hint-
style", "caption")),
            new XElement("text", primaryTile.message, new XAttribute("hint-
style", "captionsubtle"), new XAttribute("hint-wrap", true), new
XAttribute("hint-maxLines", 3)),
            new XElement("text", primaryTile.message2, new
XAttribute("hint-style", "captionsubtle"), new XAttribute("hint-wrap",
true), new XAttribute("hint-maxLines", 3))
        ),
        new XElement("subgroup", new XAttribute("hint-weight", 15),
            new XElement("image", new XAttribute("placement", "inline"),
new XAttribute("src", "Assets/StoreLogo.png"))
        )
    )
)

```

Примечание: Чтобы отобразить широкую и большую живые плитки, вы должны иметь изображения WideLogo и Square310x310Logo, определяемые в вашем манифесте приложения. Мы добавили данные изображения в Упражнении 1.

- Соберите и запустите приложение. Нажмите на кнопку Update Primary Tile. Измените размер закрепленной плитки приложения на **Wide (Широкая)** и **Large (Большая)**. Обратите внимание, что, хотя в обоих случаях есть потенциальная возможность отобразить как **message**, так и **message2**, большая плитка с большой вероятностью будет иметь место для дополнительного текста.

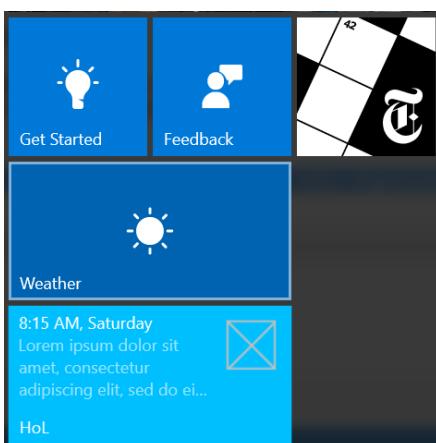


Рисунок 11
Широкая живая плитка

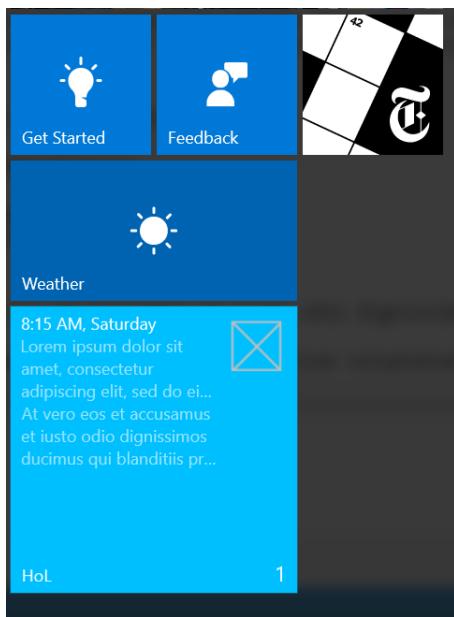


Рисунок 12

Большая живая плитка имеет место для большего количества контента.

Примечание: Большие плитки не доступны на устройствах с Windows 10 Mobile.

3. Остановите отладку и вернитесь в Visual Studio.

Упражнение 3: Запуск интерактивного уведомления (Toast)

Помимо адаптивных плиток, в Windows 10 также обновились уведомления – теперь они тоже адаптивные, а также интерактивные. Всплывающие уведомления могут включать контент, встроенные изображения и возможность ввода данных пользователем. Уведомления могут открыть приложение для выполнения задачи или запустить фоновый сервис без запуска основного приложения.

В данном упражнении мы создадим уведомление с управляющими кнопками, которое обновит данные в приложении через фоновый сервис. Приложение будет содержать в качестве примера to-do элемент, состоящий из чек-бокса и описания. Уведомление позволит отметить элемент как выполненный или незавершенный, и это проявится при обновлении основной страницы приложения.

Задача 1 – Создание сервиса для уведомления

XML для уведомления устроен очень похоже на XML плитки, который мы создавали в прошлом упражнении. Мы создадим класс сервиса для создания XML-описания всплывающего уведомления и используем его из кода MainPage.

- Щелкните правой кнопкой мыши на папку **Services** и выберите **Add > Class**. Назовите класс **ToastService.cs**.
- Сделайте класс публичным (**public**) и статичным (**static**), также добавьте пространства имён **System.Xml.Linq** и **Windows.Data.Xml.Dom**.

C#

```
using System.Xml.Linq;
using Windows.Data.Xml.Dom;

namespace TilesAndNotifications.Services
{
    public static class ToastService
```

- Добавьте метод **CreateToast()** для создания и загрузки XmlDocument.

C#

```
public static class ToastService
{
    public static XmlDocument CreateToast()
    {
        var xDoc = new XDocument(
            new XElement("toast",
                new XElement("visual",
                    new XElement("binding", new XAttribute("template",
"ToastGeneric"),
                        new XElement("text", "To Do List"),
                        new XElement("text", "Is the task complete?")
                    ) // binding
                ), // visual
                new XElement("actions",
                    new XElement("action", new XAttribute("activationType",
"background"),
                        new XAttribute("content", "Yes"), new
XAttribute("arguments", "yes")),
                    new XElement("action", new XAttribute("activationType",
"background"),
                        new XAttribute("content", "No"), new
XAttribute("arguments", "no"))
                ) // actions
            )
        );
        var xmlDoc = new XmlDocument();
        xmlDoc.LoadXml(xDoc.ToString());
        return xmlDoc;
    }
}
```

Данный код создает следующее описание всплывающего уведомления с помощью адаптивной XML-схемы:

```

<toast>
    <visual>
        <binding template="ToastGeneric">
            <text>To Do List</text>
            <text>Is the task complete?</text>
        </binding>
    </visual>
    <actions>
        <action activationType="background" content="Yes" arguments="yes" />
        <action activationType="background" content="No" arguments="no" />
    </actions>
</toast>

```

Примечание: Кнопки **action** пока ни с чем не связаны но они понадобятся нам далее в упражнении. Значение **background** для **activationType** означает, что действие не запустит приложение при активации. Подробнее XML-схема для уведомлений описана в документации: <https://msdn.microsoft.com/en-us/library/windows/apps/br230849.aspx>

4. Сохраните и закройте ToastService.
5. Откройте **MainPage.xaml**. Добавьте определение строки в сетку (Grid) и второй StackPanel в нижней строке.

XAML

```

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <StackPanel Grid.Row="0" Margin="12">
        <TextBlock Text="Adaptive Tiles" FontSize="20" FontWeight="Light" />
        <Button Click="UpdateBadge" VerticalAlignment="Top" Margin="12">Update Badge Count</Button>
        <Button Click="UpdatePrimaryTile" VerticalAlignment="Top" Margin="12">Update Primary Tile</Button>
    </StackPanel>
    <StackPanel Grid.Row="1" Margin="12">
        </StackPanel>
</Grid>

```

6. Добавьте название секции и кнопку Notify в новую StackPanel. Метод Notify() мы создадим на следующем шаге.

XAML

```

<StackPanel Grid.Row="1" Margin="12">
    <TextBlock Text="Interactive Toast" FontSize="20" FontWeight="Light" />
    <Button Click="Notify" Margin="12">Notify</Button>
</StackPanel>

```

7. Добавьте пространства имен **Windows.UI.Notifications** и **TilesAndNotifications.Services** в код MainPage.

C#

```
using TilesAndNotifications.Services;
```

8. Создайте метод **Notify()** в коде страницы.

C#

```
private void Notify(object sender, RoutedEventArgs e)
{
    var xmlDoc = ToastService.CreateToast();
    var notifier = ToastNotificationManager.CreateToastNotifier();
    var toast = new ToastNotification(xmlDoc);
    notifier.Show(toast);
}
```

Соберите и запустите приложение на локальной машине. Используйте кнопку **Notify** для отправки всплывающего уведомления. При появлении всплывающего уведомления вы можете щелкнуть по управляемым кнопкам, но ничего не произойдет. Мы реализуем данные действия в последующей задаче.

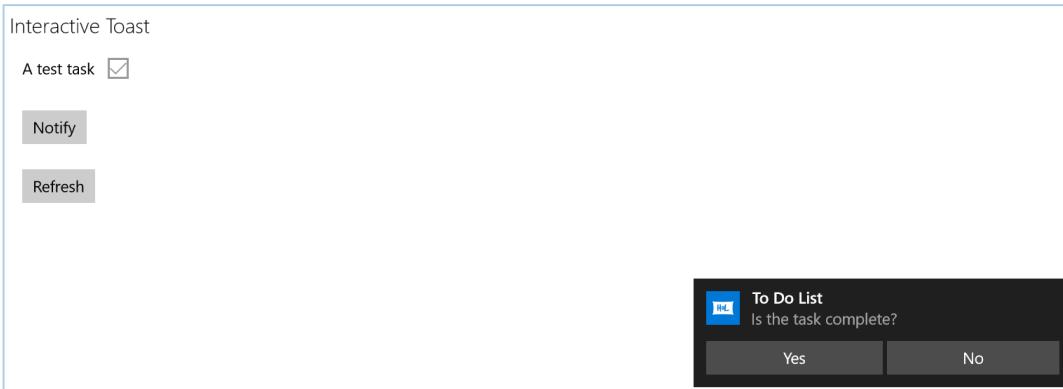


Рисунок 13

Уведомление с кнопками действия.

9. Остановите отладку и вернитесь в Visual Studio.

Задача 2 – Создание модели данных и вспомогательных методов

Прежде, чем мы сможем реализовать действия в уведомлениях, нам понадобится простой класс и вспомогательная конструкция, чтобы симулировать to-do элемент. Мы создадим их в библиотеке классов, чтобы сделать доступными из любого проекта в решении. Как приложение, так и фоновый сервис, определяемые в Windows Runtime компоненте, будут требовать доступа к модели.

1. Щёлкнитесь правой кнопкой мыши на Solution Explorer (Обозреватель решений). Затем Add (Добавить) > New Project (Новый проект). Выберите тип проекта: **Visual C# >**

Windows > Universal > Class Library (Universal Windows). Назовите его **TasksAndNotifications.Library**.

2. Удалите **Class1.cs**, используя контекстное меню обозревателя. Если необходимо, подтвердите, что действительно хотите удалить.
3. Щелкните правой кнопкой мыши на проект **TilesAndNotifications.Library** и выберите **Add (Добавить) > Existing Item (Существующий элемент)**. Найдите папку **Lab Assets** в директории практикума и выберите файлы **ToDoTask.cs** и **ToDoTaskFileHelper.cs**. Добавьте в вашу библиотеку классов.
4. Откройте **ToDoTask.cs**. В файле вы увидите, что простой ToDo-элемент состоит из **Id**, **Description** и флага **IsComplete**. Также в классе есть два метода для управления сериализацией в и из JSON.

Примечание: Мы будем хранить данные ToDo в виде файла в формате JSON, чтобы сделать их доступными для фоновой задачи, которая может иметь доступ к файлам в пакете приложений. Позднее в рамках данного упражнения вы создадите фоновую задачу.

5. Откройте **ToDoTaskFileHelper.cs**. Данный вспомогательный код сохраняет сериализованный JSON в файл и считывает его назад из файла при необходимости.
6. Вернитесь к своему приложению в Solution Explorer. Щелкните правой кнопкой мыши на папку **References** и добавьте **TilesAndNotifications.Library** как ссылку.
7. Щелкните правой кнопкой мыши на папку **Assets** и выберите **Add > Existing Item**. Найдите папку **Lab Assets** и добавьте стартовый файл **task.json**.
8. Щелкните правой кнопкой мыши на файл **task.json** и откройте его свойства (**Properties**). Установите его свойство **Build Action** в значение **Content**, и свойство **Copy to Output Directory** в значение **Copy Always**. Закройте панель свойств.

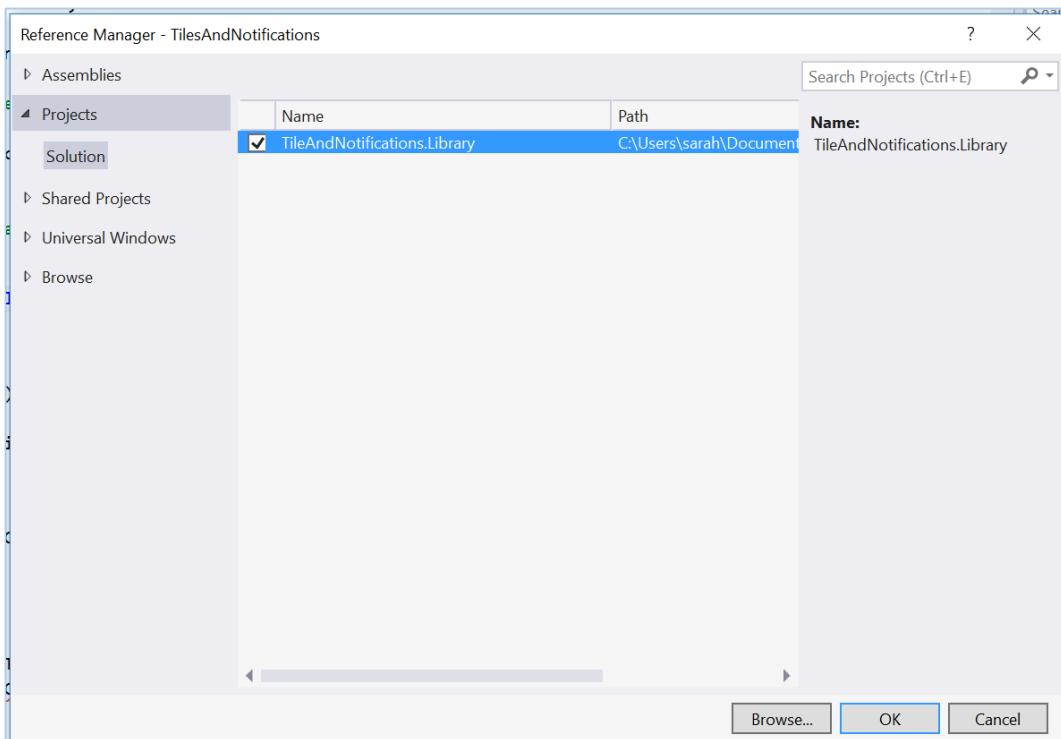


Рисунок 14
Добавление библиотеки классов по ссылке.

Задача 3 – Добавление элемента ToDo и реализация INotifyPropertyChanged

Мы отобразим элемент ToDo на странице MainPage с использованием чек-бокса для отображения статуса IsComplete. Задача ToDo будет связана со свойством в коде. Чтобы наш интерфейс понимал, когда наша фоновая задача вносит изменения в значение ToDo (позднее в данном упражнении), мы реализуем интерфейс **INotifyPropertyChanged** для CurrentToDoTask.

1. Откройте **MainPage.xaml.cs**. Добавьте пространства имен **TilesAndNotifications.Library** и **System.ComponentModel**.

```
C#  
using TilesAndNotifications.Library;  
using System.ComponentModel;
```

2. Добавьте приватное свойство для ToDoTask и публичное свойство для доступа к нему.

```
C#  
private int _count;  
private ToDoTask _currentToDoTask;  
  
public MainPage()  
{  
    InitializeComponent();  
    Loaded += MainPage_Loaded;  
}
```

```

public ToDoTask CurrentToDoTask
{
    get { return _currentToDoTask; }
    set
    {
        _currentToDoTask = value;
    }
}

```

3. Реализуйте **INotifyPropertyChanged** для сообщения в UI, когда **CurrentToDoTask** изменяется. Мы свяжем свойство с UI далее.

C#

```

public sealed partial class MainPage : Page, INotifyPropertyChanged
{
    ...
    public ToDoTask CurrentToDoTask
    {
        get { return _currentToDoTask; }
        set
        {
            _currentToDoTask = value;
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(CurrentToDoTask)));
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;
}

```

Примечание: `INotifyPropertyChanged` предоставляет обобщенные уведомления об изменениях свойств подписчикам, которые обычно свяжены с изменяющимся значением. В данном примере наш UI должен знать, когда вызванный всплывающим уведомлением фоновый сервис обновляет элемент `ToDo`. Подробнее `INotifyPropertyChanged` описан тут: <https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.data.inotifypropertychanged>

4. Откройте `MainPage.xaml` добавьте элементы управления `TextBlock` и `CheckBox` для отображения `ToDo`-задачи и ее статуса. Добавьте кнопку для обновления данных. Мы добавим метод обработки нажатия в следующем шаге.

XAML

```

<StackPanel Grid.Row="1" Margin="12">
    <TextBlock Text="Interactive Toast" FontSize="20" FontWeight="Light" />
    <StackPanel Orientation="Horizontal" Margin="12">
        <TextBlock x:Name="Description" VerticalAlignment="Center"
Text="{x:Bind CurrentToDoTask.Description, Mode=OneWay}"/>
        <CheckBox Margin="12,0,0,0" IsChecked="{x:Bind
CurrentToDoTask.IsComplete, Mode=OneWay}" IsEnabled="False" />
    
```

```
</StackPanel>
<Button Click="Notify" Margin="12">Notify</Button>
<Button Click="{x:Bind Refresh}" Margin="12">Refresh</Button>
</StackPanel>
```

5. Вернитесь к коду MainPage. Добавьте асинхронный метод **Refresh()** для чтения последнего значения задачи из данных в JSON-файле.

C#

```
private async void Refresh()
{
    var json = await ToDoTaskFileHelper.ReadToDoTaskJsonAsync();
    CurrentToDoTask = ToDoTask.FromJson(json);
}
```

6. Подпишитесь на Loaded-событие и вызовите метод Refresh() при загрузке страницы.

C#

```
public MainPage()
{
    InitializeComponent();
    Loaded += MainPage_Loaded;
}

...

private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    Refresh();
}

private async void Refresh()
{
    var json = await ToDoTaskFileHelper.ReadToDoTaskJsonAsync();
    CurrentToDoTask = ToDoTask.FromJson(json);
}
```

7. Соберите и запустите приложение. Вы увидите тестовую задачу, с по умолчанию выделенным чек-боксом IsComplete.

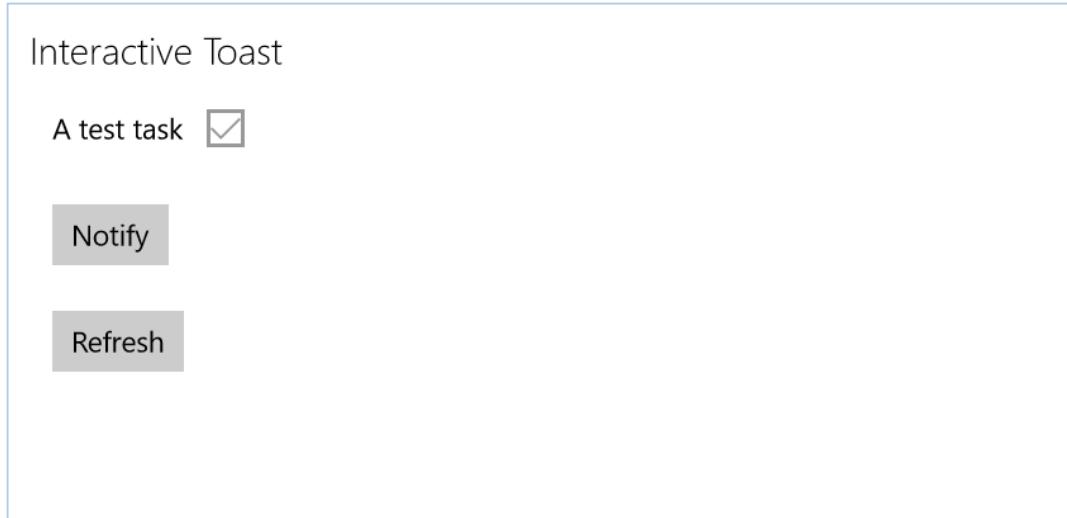


Рисунок 15

Элемент *ToDo*-задачи.

8. Остановите отладку и вернитесь в Visual Studio.

Задача 4 – Создание фонового сервиса

В этой задаче мы создадим фоновый сервис как часть компонента для Windows Runtime.

1. Щелкните правой кнопкой мыши на Solution Explorer и выберите Add > New Project. Добавьте проект типа **Visual C# > Windows > Universal > Windows Runtime Component (Universal Windows)**. Назовите его **BackgroundTasks**.
2. Щелкните правой кнопкой мыши на папку **References** в проекте **BackgroundTasks** и выберите **Add > Reference**. Добавте проект **TilesAndNotifications.Library** по ссылке. Библиотека даст доступ к **ToDoTask** и вспомогательному коду.
3. Щелкните правой кнопкой мыши и **переименуйте Class1.cs** в **ToastUpdateTask.cs**. При запросе согласитесь выполнить переименование всех ссылок с **Class1** на **ToastUpdateTask**.
4. Откройте **ToastUpdateTask.cs** и добавьте пространства имён **Windows.ApplicationModel.Background**, **Windows.UI.Notifications** и **TilesAndNotifications.Library**.

C#

```
using Windows.ApplicationModel.Background;
using Windows.UI.Notifications;
using TilesAndNotifications.Library;
```

5. Реализуйте интерфейс **IBackgroundTask**. Создайте и завершите deferral внутри метода **Run**.

C#

```
namespace BackgroundTasks
{
    public sealed class ToastUpdateTask : IBackgroundTask
    {
        public async void Run(IBackgroundTaskInstance taskInstance)
        {
            var deferral = taskInstance.GetDeferral();

            deferral.Complete();
        }
    }
}
```

6. Преобразуйте `taskInstance.TriggerDetails` в `ToastNotificationActionTriggerDetails`.

Данные детали передаются из действия **Yes** в уведомлении. Если детали не пусты, прочитайте JSON-задачу и установите ее флаг `IsComplete` в `true`.

C#

```
namespace BackgroundTasks
{
    public sealed class ToastUpdateTask : IBackgroundTask
    {
        public async void Run(IBackgroundTaskInstance taskInstance)
        {
            var deferral = taskInstance.GetDeferral();

            var details = taskInstance.TriggerDetails as
ToastNotificationActionTriggerDetail;
            if (details != null)
            {
                string arguments = details.Argument;
                // this is where you would retrieve any user input
                var userInput = details.UserInput;

                var json = await
ToDoTaskFileHelper.ReadToDoTaskJsonAsync();
                var task = ToDoTask.FromJson(json);

                task.IsComplete = arguments == "yes";

                await ToDoTaskFileHelper.SaveToDoTaskJson(taskToJson());
            }

            deferral.Complete();
        }
    }
}
```

Примечание: Дополнительную информацию об аргументах и деталях триггеров уведомлений можно найти тут: <https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.notifications>

Задача 5 – Регистрация фонового сервиса (задачи)

До запуска приложения необходимо зарегистрировать фоновый сервис в приложении TilesAndNotifications и объявить его в манифесте.

1. Вернитесь к своему проекту приложения. Откройте пакет манифеста в редакторе и перейдите на вкладку **Declarations**. Выберите **BackgroundTasks** из списка **Available Declarations (Доступные объявления)** и нажмите для **Add** добавления объявления.
2. Установите тип **task type** в значение **System Event** и **Entry point** в значение **BackgroundTasks.ToastUpdateTask**. Сохраните и закройте манифест.

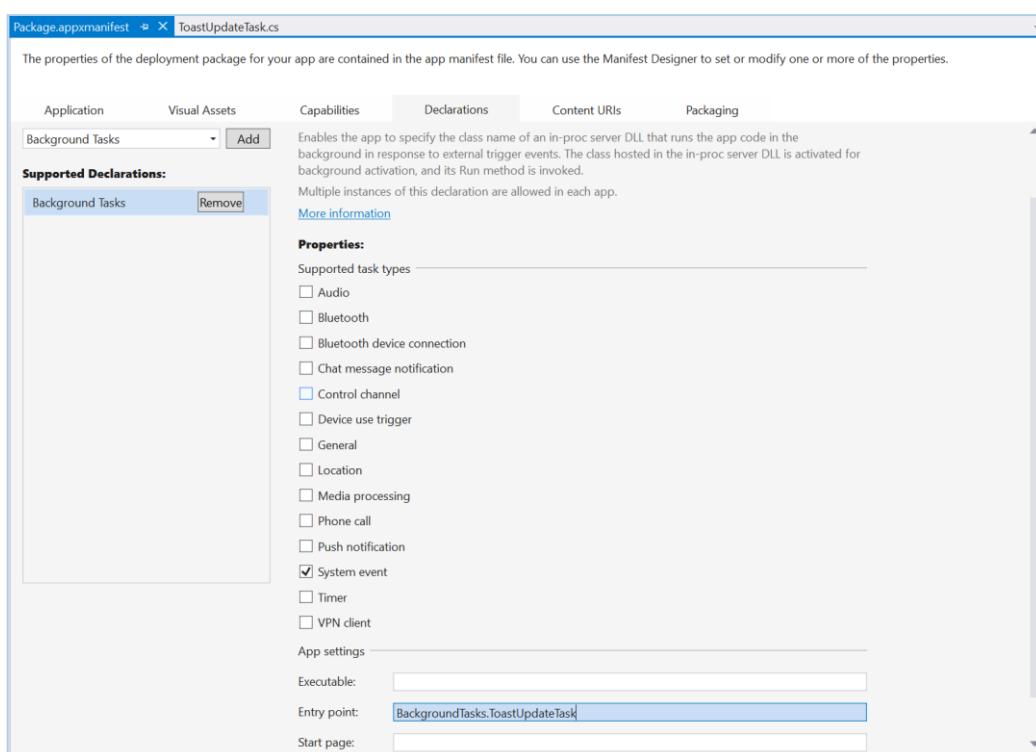


Рисунок 16

Объявление фоновой задачи в манифесте приложения.

3. В вашем проекте **TilesAndNotifications** щелкните правой кнопкой мыши на папку **References** и выберите **Add > Reference**. Добавьте проект **BackgroundTasks** по ссылке.
4. Откройте **App.xaml.cs**. Добавьте пространства имен **Windows.ApplicationModel.Background** и **BackgroundTasks**.

C#

```
using Windows.ApplicationModel.Background;
using BackgroundTasks;
```

5. Добавьте **async**-метод **RegisterBgTask** для управления регистрацией сервиса.

C#

```
private async void RegisterBgTask(string taskName, Type taskType, bool
isInternetRequired = false)
{
    var backgroundAccessStatus = await
BackgroundExecutionManager.RequestAccessAsync();

    if (backgroundAccessStatus ==
BackgroundAccessStatus.AllowedMayUseActiveRealTimeConnectivity ||
        backgroundAccessStatus ==
BackgroundAccessStatus.AllowedWithAlwaysOnRealTimeConnectivity)
    {
        // Check to see if the task has already been registered
        if (BackgroundTaskRegistration.AllTasks.Any(t =>
t.Value.Name == taskName))
        {
            return;
        }

        // Register the toast update task
        var taskBuilder = new BackgroundTaskBuilder
        {
            Name = taskName,
            TaskEntryPoint = taskType.FullName
        };

        taskBuilder.SetTrigger(new
ToastNotificationActionTrigger());
        var registration = taskBuilder.Register();
    }
}
```

6. Вызовите метод **RegisterBgTask** из конструктора приложения и передайте ему тип (**typeof**) **ToastUpdateTask**.

C#

```
public App()
{
    this.InitializeComponent();
    this.Suspending += OnSuspending;
    RegisterBgTask("ToastUpdateTask", typeof(ToastUpdateTask));
}
```

7. Соберите и запустите приложение. Используйте кнопку **Notify** отправки всплывающего уведомления. Если ваша задача была уже завершена, выберите **Нет** в качестве

действия. Используйте кнопку Refresh, чтобы обновить статус задачи. Чек-бокс будет отображать выбор, сделанный в уведомлении.

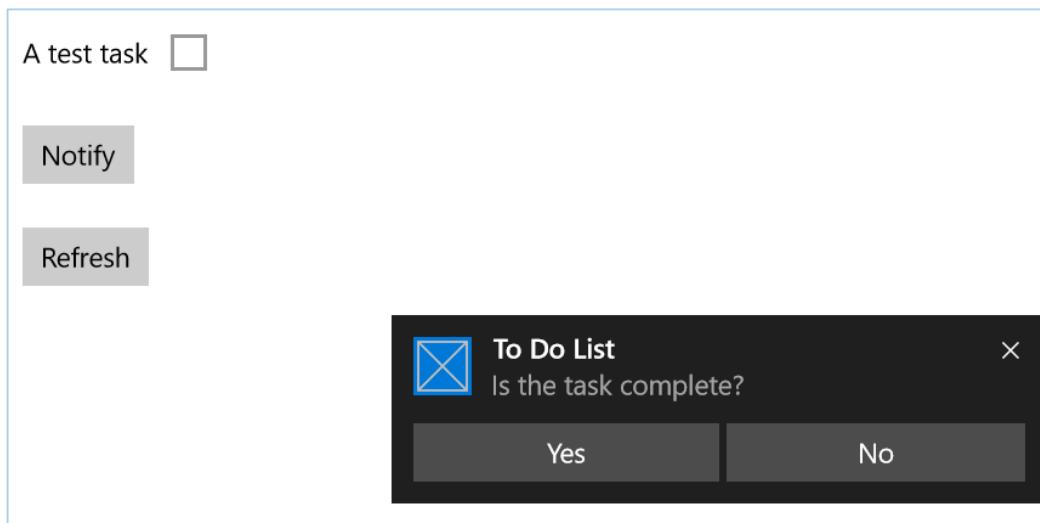


Рисунок 17
Изменение статуса задачи из уведомления.

8. Остановите отладку и вернитесь в Visual Studio.

Резюме

В рамках данной лабораторной работы вы сумели добавить в свое приложение собственные изображения в качестве графических ресурсов и использовать их для создания разнообразных плиток по умолчанию, которые могут отображать счетчик уведомлений. Вы узнали о новой адаптивной схеме плитки и научились создавать живые плитки (маленькие, средние, широкие и большие), а затем поработали над интерактивными всплывающими уведомлениями (toast) с фоновой активацией.



Лабораторный практикум

Веб-приложения в Windows

Октябрь 2015 г.



Overview

Следя за релизом Windows 10, набор инструментов «Windows Bridge» и, в частности, Project Westminster открывают доступ к UWP-платформе широкому спектру разработчиков с разным опытом за плечами, включая iOS-разработчиков, разработчиков классических Windows-приложений и веб-разработчиков. Мост Windows для веба позволяет вам легко перейти от вашего кода в вебе в пространство приложений, фактически опубликовав ваш отзывчивый сайт в Магазине Windows. Веб-приложения и хостящиеся веб-приложения имеют доступ к вызову UWP API напрямую из JavaScript, что позволяет интегрироваться с такими возможностями, как живые плитки, активные уведомления, контакты, голосовые команды для Cortana и встраиваемыми покупками Магазина Windows.

Хостящиеся веб-приложения немедленно отображают изменения, сделанные в коде веб-сайта, позволяя вам легко поддерживать контент в актуальном состоянии.

После создания хостящегося веб-приложения для Windows 10 вы можете заинтересоваться расширением на другие платформы. ManifoldJS – это новый фреймворк с открытым кодом, который генерирует хостящиеся веб-приложения для популярных платформ.

Цели

Данная лабораторная научит вас:

- Создавать хостящиеся веб-приложения
- Отправлять уведомления из хостящегося веб-приложения
- Получать доступ к камере из хостящегося веб-приложения
- Задавать собственные изображения для плиток и экрана загрузки
- Обновлять живые плитки из хостящегося веб-приложения
- Генерировать хостящиеся веб-приложения для других платформ с помощью ManifoldJS

Системные требования

Для выполнения лабораторной работы вам потребуются:

- Microsoft Windows 10
- Microsoft Visual Studio 2015

Опциональные требования

Если вы хотите также сделать опциональные задачи, вам потребуются:

- The Node Package Manager (npm)
 - ManifoldJS
-

Установка

Вы должны осуществить следующие шаги для подготовки своего компьютера к данной лабораторной работе:

1. Установите Microsoft Windows 10.
2. Установите Microsoft Visual Studio 2015. Выберите опцию настройки установки (custom install) и убедитесь, что инструменты разработки универсальных Windows-приложений выбраны в списке опций.
3. Опционально: установите npm.
4. Опционально: установите ManifoldJS

Инструкции и ссылки на установку npm и ManifoldJS приведены в Упражнении 3: Задаче 1.

Упражнения

Данный практикум включает следующие упражнения:

1. Создание хостящегося веб-приложения
 2. Поддержка дополнительных платформ и устройств с помощью ManifoldJS (опционально)
-

Расчетное время для завершения курса: **От 30 до 45 минут.**

Упражнение 1: Создание хостящегося веб-приложения

Имея уже размещенный в сети отзывчивый веб-сайт, вы можете создать хостящееся веб-приложение для Магазина Windows за считанные минуты. В данном упражнении мы создадим хостящееся веб-приложение, используя сайт Codepen.io в качестве примера. Codepen позволяет вам вводить и выполнять собственный код на JavaScript, CSS и HTML. Как только вы дадите доступ к Windows Runtime для вашего приложения, Codepen будет для вас отличным способом познакомиться с интеграцией с API без необходимости размещать специальный сайт на сервере. Мы вызовем уведомление из хостящегося приложения и добавим возможность сделать снимок через камеру.

Задача 1 – Создание пустого универсального Windows-приложения на JavaScript

Мы начнем с создания проекта на основе шаблона UWP Blank App JavaScript.

1. Откройте Visual Studio 2015, используйте **File > New > Project**, чтобы открыть диалог создания нового проекта. Перейдите в **Installed > Templates > JavaScript** и выберите шаблон **Blank App (Universal Windows)**.
2. Назовите ваш проект **HostedWebApp** и выберите место для сохранения вашего проекта практикум. Мы создали папку на диске **C:**, названную **HOL** – вы увидите ее на снимках экрана по ходу лабораторной работы.
3. Оставьте без изменения опцию **Create directory for solution**. Вы можете снять выделение с опции **Add to source control** если не собираетесь управлять версиями проекта. Нажмите **OK** для создания проекта.

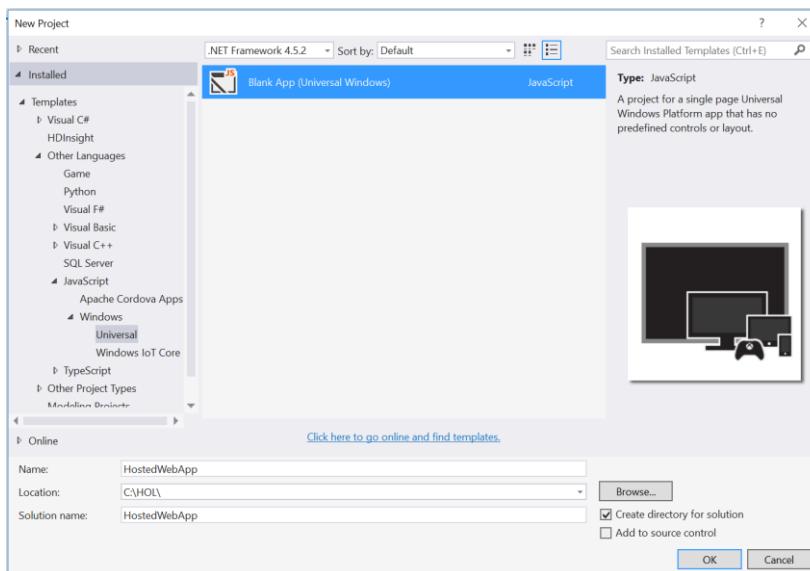


Рисунок 1

Создание нового пустого проекта в Visual Studio 2015.

4. Установите конфигурацию решения в **Debug** (Отладка) и целевую и **x86**. Выберите **Local Machine** (локальная машина) из выпадающего меню Debug Target справа от кнопки начала отладки.

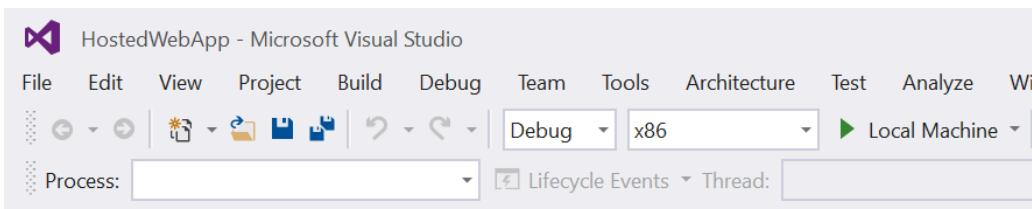


Рисунок 2

Настройте запуск приложения на локальной машине.

Примечание: -- это кнопка начала отладки.

5. Используйте кнопку начала отладки, чтобы собрать и запустить приложение. Вы увидите черный фон приложения с надписью “Content goes here.”

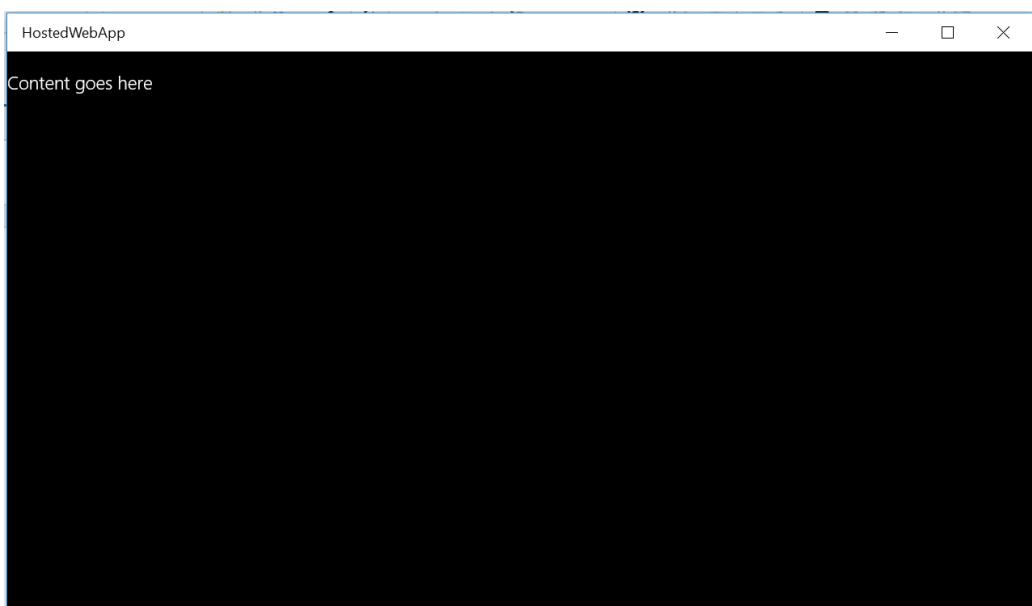


Рисунок 3

Пустое универсальное приложение на JavaScript, запущенное в десктопном режиме.

6. Остановите отладку и вернитесь Visual Studio.

Задача 2 – Добавление Codepen в веб-приложение

Хостящееся веб-приложение представляет собой обертку вокруг веб-сайта, предоставляющего контент, она использует движок Edge для рендеринга. Вы можете ограничить набор просматриваемых страниц конкретным сайтом или сайтами и формировать переход по остальным ссылкам во внешний браузер по умолчанию. В данной задаче мы отобразим сайт Codepen в виде хостящегося веб-приложения и запустим всплывающее уведомление из приложения.

1. Удалите папки **css**, **js**, и **WinJS**, а также файл **default.html** из проекта **HostedWebApp**.

Замечание: при создании веб-приложения с исключительно хостящимся контентом вы можете удалить папки **css**, **js** и **WinJS** а также файл **default.html**. Мы удалили данные файлы, но вы можете решить их сохранить и использовать для реализации оффлайн-страниц в вашем приложении.

Мы оставили папку **images**, так как в ней размещены изображения приложения, например, экран загрузки и логотип для магазина, которые по-прежнему имеют смысл в контексте хостящегося приложения.

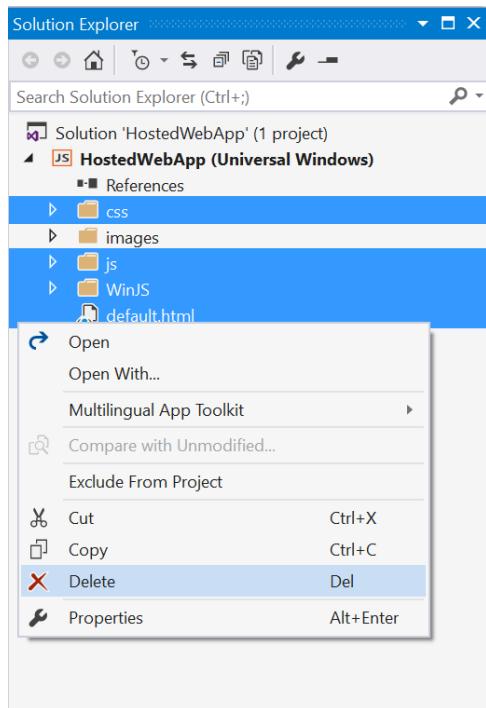


Рисунок 4

Удалите папки и файлы, которые не нужны в хостящемся приложении.

2. Откройте файл **package.appxmanifest** из вашего проекта **HostedWebApp** в редакторе манифеста.
3. На вкладке **Application** измените **Start page** на
<http://codepen.io/seksenov/pen/wBbVyb/?editors=101>.

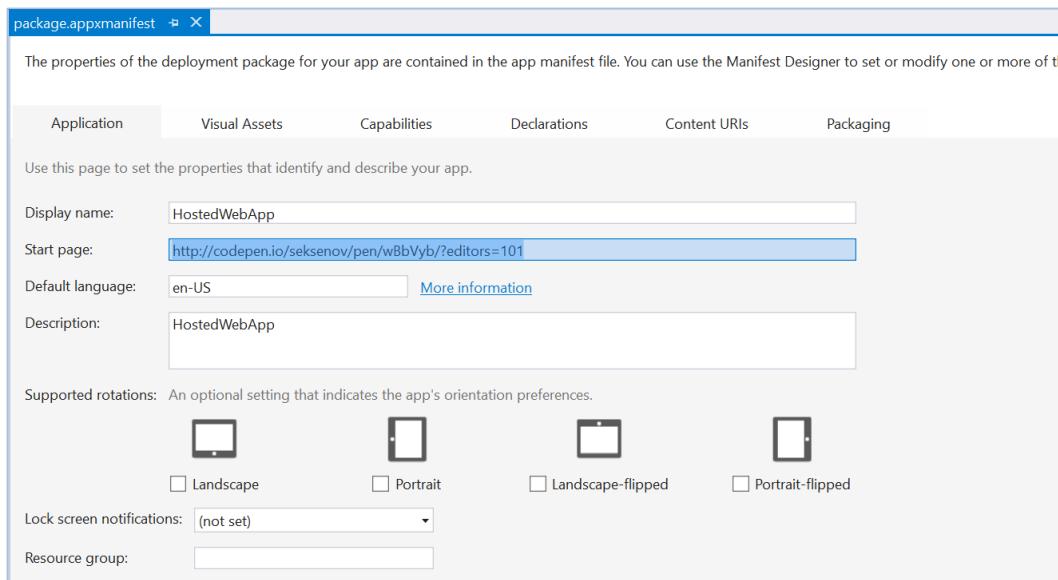


Рисунок 5

Установка *Codepen* в качестве стартовой страницы

4. Перейдите на вкладку **Content URIs** и добавьте <http://codepen.io/seksenov/pen/wBbVyb/?editors=101> в поле URI. Оставьте **Rule** установленным в **include** и поставьте **WinRT Access** в значение **All**.

Примечание: Правила для ссылок на контент приложения (ACURS, Application Content URI Rules) указывают страницы, которые хостятся или разрешены в вашем приложении. В частности, вы можете решить, что пользователи могутходить по сайту Codepen внутри вашего приложения, но внешние ссылки должны открывать в браузере. Подобные включения и исключения позволяют вам контролировать границы вашего приложения и предотвращают скатывание к сценарию, напоминающему стандартный веб-браузер. Контентные ссылки также дают возможность включить или выключить доступ к Windows Runtime для различных частей приложения и решить, какой уровень доступа нужно дать: **None**, **All**, или **Allow for web only**.

Чтобы указать внешнюю ссылку используйте протокол **http(s)://**. Чтобы задать локальную ссылку, используйте протокол **ms-appx-web:///**.

5. Добавьте **http://*.codepen.io/** как дополнительную контентную ссылку с теми же настройками для **Rule** и **WinRT Access**, что и в первой ссылке. Звездочка указывает покрытие всех возможных поддоменов.

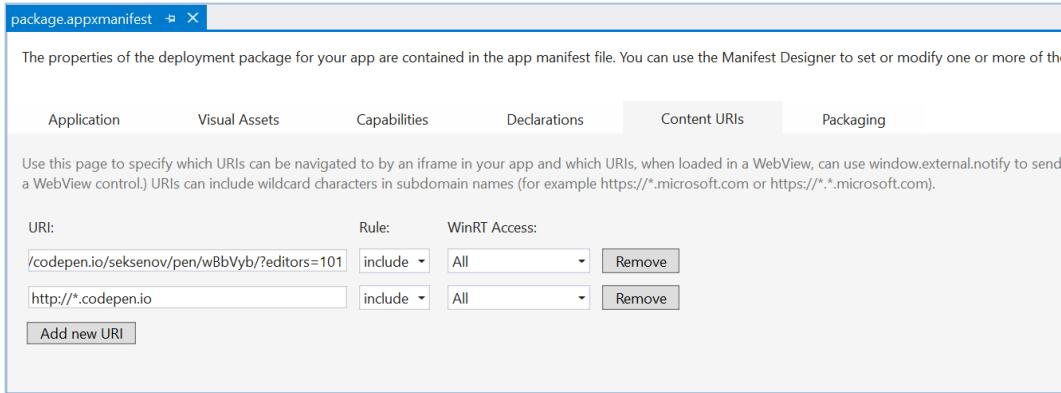


Рисунок 6

Добавление контентных ссылок для Codepen

- Соберите и выполните приложение. Вы увидите, что сайт Codepen появился внутри окна приложения уже с некоторым кодом и контентом на HTML и JavaScript. Код генерации уведомления уже присутствует в приложении.

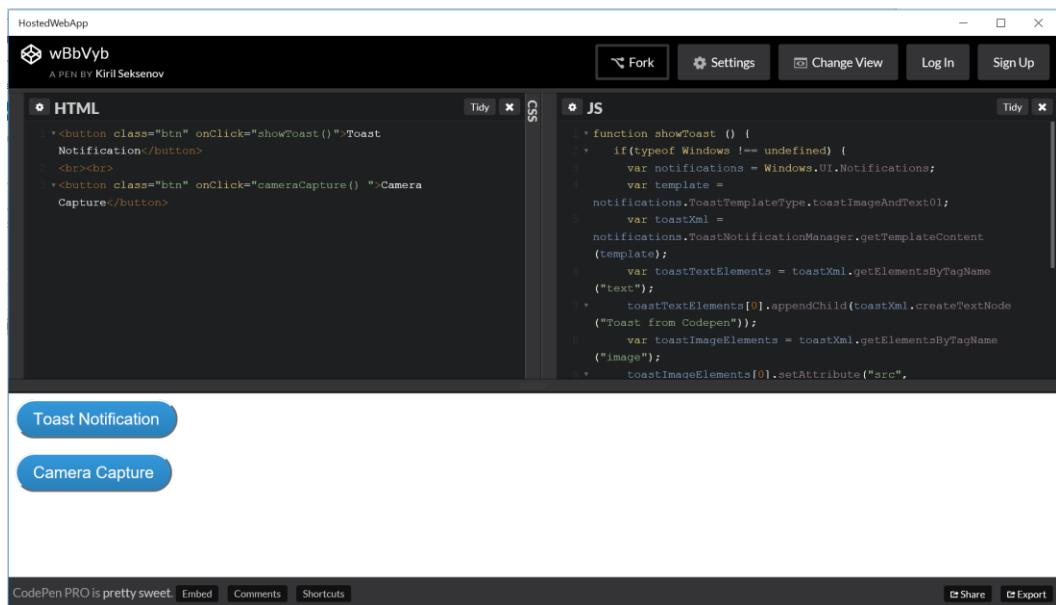


Рисунок 7

Codepen в качестве хостящегося веб-приложения.

- Используйте кнопку «Toast Notification» для отправки уведомления в вашу систему. Мы установили свойство **WinRT Access** в значение **All** для **Codepen**, поэтому код на JavaScript в панели имеет права на доступ к Windows API, включая возможность отправки уведомлений.

Примечание: Мы используем Codepen в качестве удобного инструмента для вставки налету собственного кода на JavaScript с возможностью сохранить на удаленном сервере и немедленного его отображения для тестирования в приложении. В реальном сценарии метод `showToast()` будет работать как часть скрипта внутри веб-проекта, который вы

размещаете на некотором сервере, чтобы он начал работать. Вы можете создать отдельный проект для хостящегося веб-приложения и использовать контентные ссылки для отображения сайта в приложении.

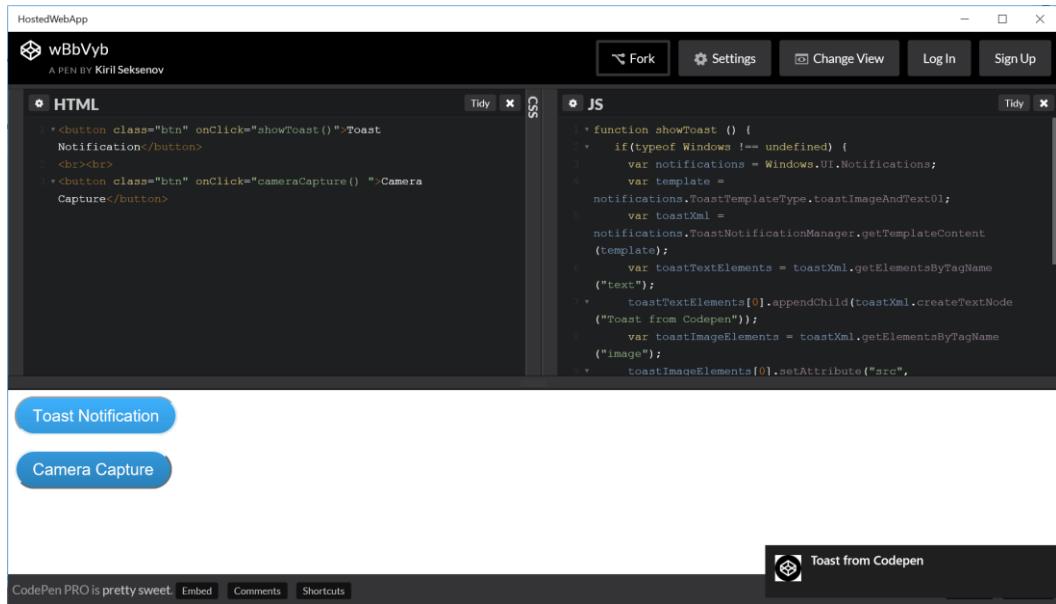


Рисунок 8

Уведомление, сгенерированное хостящимся веб-приложением

Задача 3 – Включение съемки камерой

Хотя кнопка «Camera Capture» уже определена в разметке HTML, сама съемка камерой еще не реализована. Чтобы это сделать мы добавим немного кода.

1. Во все еще работающем приложении добавьте новую функцию после метода `systemAlertCommandInvokedHandler()` в панели **JS**, чтобы отработать съемку камерой:

JavaScript

```
function cameraCapture() {  
    if (typeof Windows != 'undefined') {  
        var captureUI = new Windows.Media.Capture.CameraCaptureUI();  
        //Set the format of the picture to be captured (.png, .jpg, ...)  
        captureUI.photoSettings.format =  
            Windows.Media.Capture.CameraCaptureUIPhotoFormat.png;  
        //Pop up the camera UI to take a picture  
        captureUI.captureFileAsync(  
            Windows.Media.Capture.CameraCaptureUIMode.photo).then(  
                function(capturedItem) {  
                    // Do something with the picture  
                });  
    }  
}
```

2. Используйте кнопку **Camera Capture** чтобы открыть интерфейс камеры. Если система запросит разрешение на доступ к камере, нажмите **Yes**. После открытия интерфейса камеры вы можете снять кадр с помощью кнопки камеры с правой стороны окна.

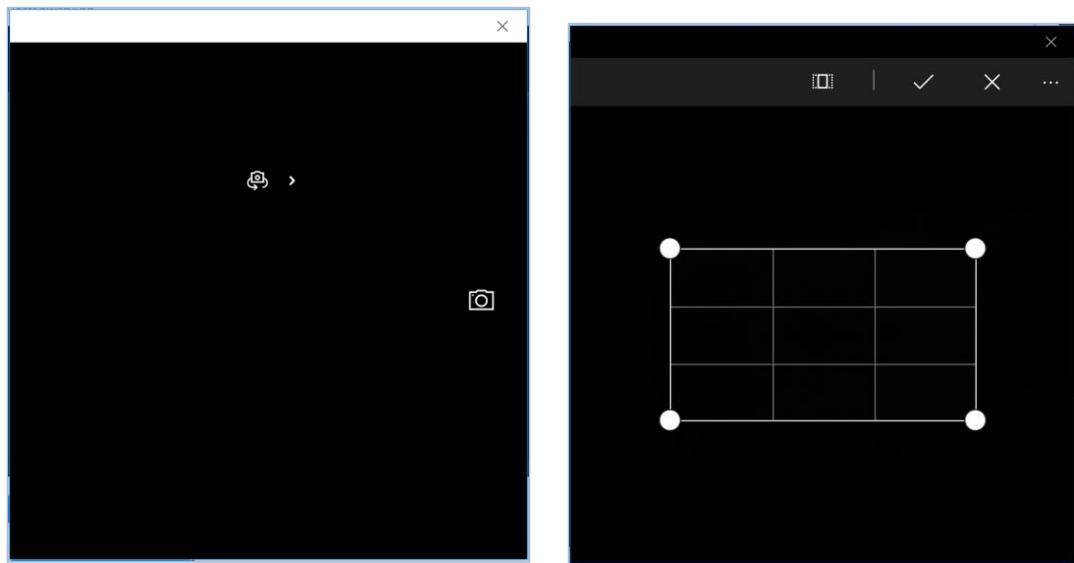


Рисунок 9

Снятие изображения с помощью открытой камеры, режим редактирования фотографии.

Примечание: мы не сохраняем фотографию внутри блока **then**, поэтому любая съемка носит временный характер.

3. Остановите отладку и вернитесь в Visual Studio.

Задача 4 – Добавление живых плиток

Также, как и с другими UWP-приложениями, вы можете определить изображения для хостящихся веб-приложений. В рамках данной задачи мы добавим изображения для экрана загрузки, средней плитки и меню Пуск. После настройки вида плитки по умолчанию мы создадим и обновим живую плитку из приложения.

1. Откройте **package.appxmanifest** в редакторе манифеста и выберите вкладку **Visual Assets**.
2. Используйте многоточие под **Square 71x71 logo** в масштабе **Scale 200**, чтобы открыть диалог выбора изображения. Перейдите в папку **Lab Assets** в директории практикума и выберите файл **Square71x71Logo.scale-200.png**. Нажмите **Open**, чтобы заменить картинку по умолчанию выбранным логотипом. Если нужно, подтвердите, что хотите перезаписать файл в проекте.

- Повторите шаг 2 с **Square150x150Logo** в масштабе **Scale 200**, **Square44x44Logo** в масштабе **Scale 200**, и **Splash screen logo** в масштабе **Scale 200** в соответствии с пунктами в манифесте.

Примечание: мы добавили изображения логотипов в демонстрационных целях. Для более глубокого погружения и дополнительных рекомендаций рекомендуем посмотреть лабораторную работу про работу с живыми плитками и уведомлениями.

- Выберите раздел **All Image Assets**, измените поля **Background color** для плитки и экрана загрузки на значение **deepSkyBlue**. Промотайте вниз, чтобы убедиться, что новые изображения отображаются в манифесте.

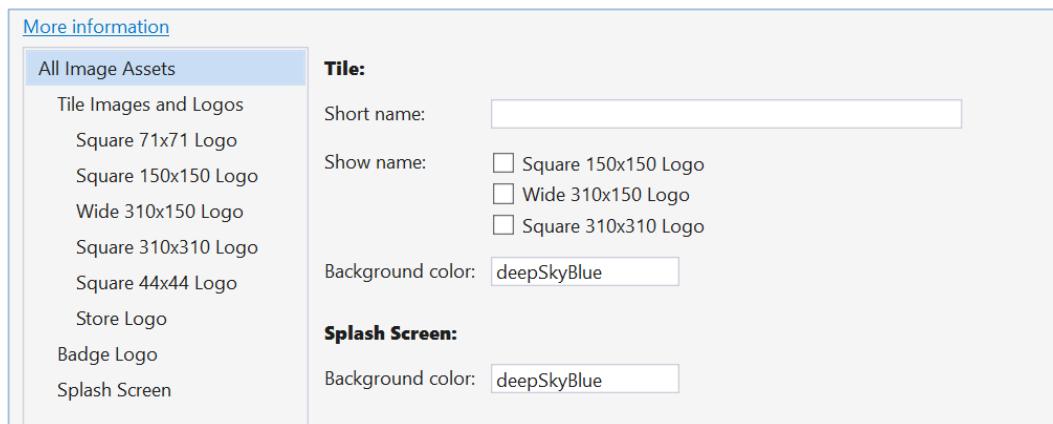


Рисунок 10

Установка цвета фона для плиток и экрана загрузки.

- Соберите и запустите приложение. При загрузке приложения вы увидите новый экран загрузки с голубым фоном и белым логотипом практикума по центру.

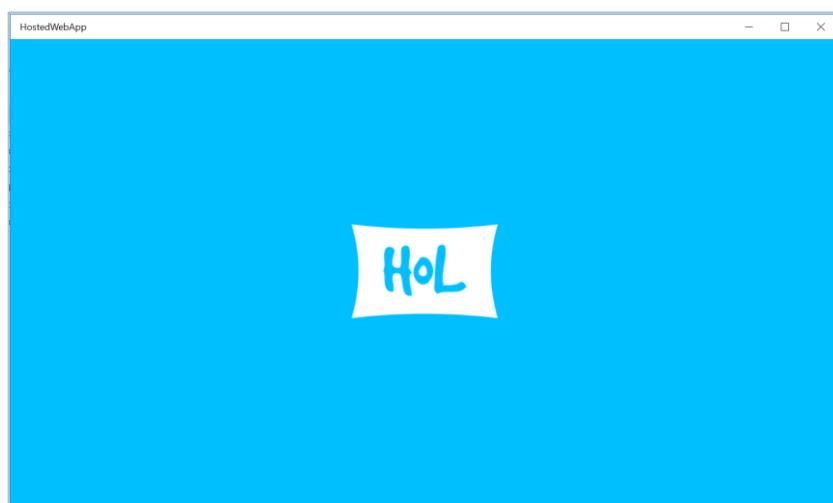


Рисунок 11

Новый собственный экран загрузки для хостящегося веб-приложения.

6. Пока приложение работает, найдите приложение **HostedWebApp** в списке всех приложений в меню Пуск. Щелкните правой кнопкой мыши по имени приложения и закрепите его в меню Пуск. Если плитка приложения стала любого другого размера, кроме среднего, поменяйте размер на средний. Вы увидите голубой фон с белым логотипом HoL.



Рисунок 12
Средняя плитка по умолчанию

7. Вернитесь к запущенному приложению. В панели HTML добавьте кнопку, которая вызывает метод **updateTile()** по событию click.

HTML

```
<button class="btn" onClick="cameraCapture() ">Camera Capture</button>
<br><br>
<button class="btn" onClick="updateTile() ">Update Tile</button>
```

HostedWebApp
wBbVyb
A PEN BY KirillSeksenov

HTML

```
<button class="btn" onClick="showToast()">Toast Notification</button>
<br><br>
<button class="btn" onClick="cameraCapture() ">Camera Capture</button>
<br><br>
<button class="btn" onClick="updateTile() ">Update Tile</button>
```

Toast Notification
Camera Capture
Update Tile

CodePen PRO is pretty sweet. [Embed](#) [Comments](#) [Shortcuts](#)

Рисунок 13
Кнопка обновления плитки в панели HTML.

8. Добавьте функцию **updateTile()** в панели JS после функции cameraCapture().

JavaScript

```
function updateTile() {
    if (typeof Windows !== 'undefined' && typeof Windows.UI !== 'undefined'
&&
        typeof Windows.UI.Notifications !== 'undefined')
    {
        console.log('Attempting to update the tile');
        var notifications = Windows.UI.Notifications,
            tile =
        notifications.TileTemplateType.tileSquare150x150PeekImageAndText01,
            tileContent = notifications.TileUpdateManager.getTemplateContent(
                tile),
            tileText = tileContent.getElementsByName('text'),
            tileImage = tileContent.getElementsByName('image');

        tileText[0].appendChild(tileContent.createTextNode('Demo
Message'));
        tileImage[0].setAttribute('src',
            'http://unsplash.it/150/150/?random');
        tileImage[0].setAttribute('alt', 'Random demo image');

        var tileNotification = new
            notifications.TileNotification(tileContent);
        var currentTime = new Date();
        tileNotification.expirationTime = new Date(currentTime.getTime() +
20
            * 1000);

        notifications.TileUpdateManager.createTileUpdaterForApplication(
            ).update( tileNotification);

    }
    else {
        //alternate behavior
    }
}
```

Примечание: каждый раз, когда в рамках хостящегося контента мы добавляем метод, требующий доступ к API платформы, мы также проверяем, если тип (**typeof**) **Windows** определен. Это условие равно true, если сайт работает в формате хостящегося веб-приложения на устройстве с Windows 10. Дополнительно метод **updateTile()** также проверяет доступность **Windows.UI** и **Windows.UI.Notifications** прежде, чем к ним обращаться.

Для простоты мы используем старый шаблон для создания плитки. О настройке адаптивного шаблона можно узнать в лабораторной работе про плитки и уведомления.

9. Нажмите на кнопку **Update Tile**, чтобы вызвать обновление живой плитки. Когда вы откроете меню Пуск, вы увидите, что изображение сменяется и ваша плитка становится живой. Через несколько секунд поверх картинки появится текст.

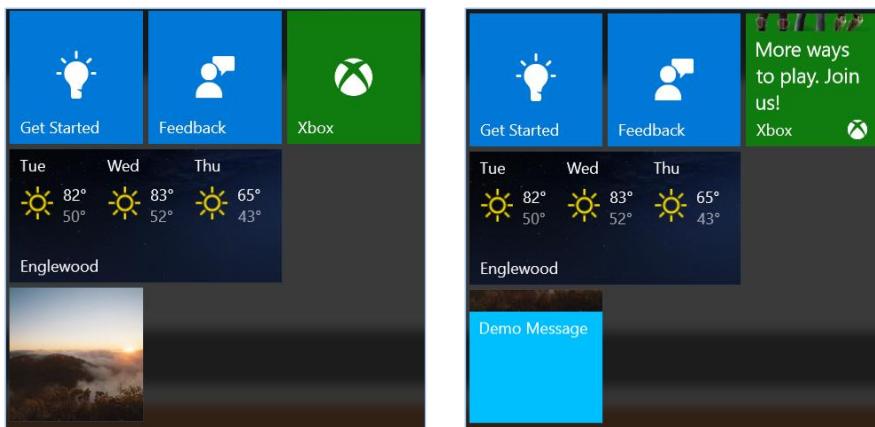


Рисунок 14

Контент живой плитки

10. Подождите 20 секунд, наблюдая за плиткой. По истечении времени плитка вернется в изначальное состояние.

Примечание: внутри метода updateTile() мы отправляем сообщение через console.log. Если что-то пошло не так, посмотрите окно JavaScript Console в Visual Studio, чтобы проверить сообщение там. Если в консоли написано **Attempting to update the tile**, возможно, у вас синтаксическая ошибка в методе.

11. Остановите отладку и вернитесь в Visual Studio.

Задача 5 – Дополнительные возможности

Помимо сценариев, которые мы посмотрели в лабораторной работе, хостящиеся приложения могут интегрироваться со множеством других возможностей из богатого набора Windows 10. Хотя мы не сможем осветить подробно их все в лабораторной, есть несколько сценариев, которые могут вас заинтересовать. Дополнительная информация доступна по ссылке:

<http://microsoftedge.github.io/WebAppsDocs/en-US/win10/HWAfeatures.htm>.

Голосовые команды для Кортаны

Вы можете интегрировать хостящееся веб-приложение с Кортаной, указав файл Voice Command Definition (VCD) в вашем html-коде, используя мета-тег. После регистрации, вы можете использовать VCD для голосовой активации приложения, запуска фоновых сервисов и реализации других способов взаимодействия с Кортаной.

Гибридные приложения

Если вы хотите, чтобы пользователи могли иметь доступ к вашему приложению в оффлайн-режиме, вы можете создать гибридное приложение. Гибридное приложение может отдавать как хостящийся контент, так и локальный, размещенный внутри пакета или локального хранилища.

Брокер веб-аутентификации

Выше хостящееся веб-приложение может использовать преимущества брокера веб-аутентификации для обработки входа пользователей, если вы опираетесь на такие интернет-протоколы, как OpenID или OAuth. URI для веб-аутентификации может быть также определен через мета-тег внутри любой HTML-страницы приложения.

Декларация свойств приложения

Для предоставления доступа к микрофону и другим возможностям устройства и ресурсам, вы должны объявить соответствующие возможности в манифесте приложения. Это можно сделать через редактор или вручную через XML. Подробности доступны в документации: <https://msdn.microsoft.com/en-us/library/windows/apps/br211477.aspx>.

Упражнение 2: Поддержка дополнительных платформ и устройств с ManifoldJS (опциональное)

Хостящиеся веб-приложения – это отличный способ быстро перенести существующий веб-проект с отзывчивым дизайном на новые платформы. ManifoldJS – это инструмент, который использует метаданные с вашего сайта для генерации нативных хостящихся приложений для разных платформ, включая iOS, Android, Windows 10, Chrome OS и Firefox OS. Для платформ, которые не поддерживают хостящиеся веб-приложения напрямую, ManifoldJS использует Cordova.

Манифест, генерируемый ManifoldJS следует стандарту W3C для манифеста веб-приложения и включает такие метаданные, как стартовая страница сайта, белый список URL, имя сайта, цвет темы и изображения для приложения.

Примечание: Свежая версия и новости про ManifoldJS доступны на сайте <http://www.manifoldjs.com/>. Подробнее о манифесте W3C для веб-приложений можно узнать на сайте <https://w3c.github.io/manifest/>.

Задача 1 – Установка ManifoldJS и создание манифеста

1. Откройте командную строку из-под администратора. С помощью установленного npm используйте команду `npm install -g manifoldjs`, чтобы поставить ManifoldJS в глобальном режиме на вашей рабочей машине.

Command Prompt

```
> npm install -g manifoldjs
```

Примечание: Посетите сайт <https://nodejs.org/>, чтобы скачать и поставить пакетный менеджер node (npm).

2. Сгенерируйте манифест на сайте <http://www.manifoldjs.com/generator>. Вы также можете загрузить готовый манифест, чтобы генератор его поправил и сообщил вам о возможных пробелах.

Примечание: Если ваш сайт не имеет манифеста, ManifoldJS может сгенерировать его для вас. Однако вы, наверняка, захотите создать собственный, чтобы воспользоваться всеми возможностями брендирования.

3. Загрузите манифест в корень вашего сайта на сервере. Обычно манифест размещается там же, где и файл index.html.

Задача 2 – Генерация хостящегося веб-приложения

В рамках данной задачи мы создадим хостящиеся веб-приложения для вашего сайта под разные платформы.

4. Вернитесь к лоальной машине. Создайте папку для хранения хостящихся веб-приложений. Перейдите в командную строку. Передайте адрес сайта в manifoldjs для генерации манифеста. Мы используем Bing в качестве примеры. Вы можете также добавить параметр -l debug для словесного вывода.

Command Prompt

```
> manifoldjs http://www.bing.com/
```

5. Проверьте код, сгенерированный утилитой ManifoldJS в вашей папке.
6. Чтобы поставить и запустить сгенерированное Windows 10 приложение, запустите следующую команду из папки, созданной ManifoldJS:

Command Prompt

```
> manifoldjs run windows
```

7. Ваше приложение будет сгенерировано и запущено.

Резюме

Хостящиеся веб-приложения – это мощный способ интегрировать опыт существующего веб-проекта с возможностями Магазина Windows и API платформы. В рамках данной лабораторной работы мы создали хостящееся веб-приложение с собственными плитками, которое может отправлять уведомления, обновлять плитки и запускать камеру устройства. Мы также познакомились с тем, как генерировать веб-приложения для различных других платформ.



Лабораторный практикум

Использование облачного сервиса
Azure Mobile Apps в приложениях
на платформе Windows

Октябрь, 2015 г.



Описание

Azure App Service - это управляемый PaaS сервис для профессиональных разработчиков, предоставляющий широкие возможности для веб, мобильных и интеграционных сценариев.

Azure App Service предоставляет возможность развернуть любое веб-приложение на любом языке, получить готовый бэкенд для мобильных клиентов, инфраструктуру для API и готовый механизм рабочих процессов для управления потоками данных без написания кода.

Mobile Apps – мобильная служба, представляет собой функциональный модуль Azure App Service и является высоко масштабируемой, доступной из любой точки платформой для разработки мобильных приложений, которая предоставляет широкие возможности мобильным разработчикам и инструменты для быстрой и комфортной разработки корпоративных приложений.

Mobile Apps позволяет:

- **Создавать нативные и кроссплатформенные приложения.** Вы можете воспользоваться готовыми SDK службы не зависимо от того, какие приложения вы создаете: нативные iOS, Android и Windows приложения или с использованием кроссплатформенных технологий Xamarin и Apache Cordova.
- **Подключаться к системам и сервисам вашей компании.** С помощью Mobile Apps вы можете добавлять авторизацию с корпоративным аккаунтом за несколько минут и подключаться к ресурсам вашей компании, размещенными локально или в облаке.
- **Подключаться к API SaaS решений.** Более 40 коннекторов помогут вам легко интегрировать ваше приложение с SaaS решениями, которые использует ваша компания.
- **Реализовать офлайн-сценарии работы с приложениями.** Сделайте работу с вашими мобильными решениями более продуктивной, создавая приложения, которые могут работать в режиме офлайн и используют Mobile Apps для синхронизации данных в фоне при наличии соединения с любым из источников данных вашей организации или API SaaS решения.
- **Отправлять Push-уведомления миллионам пользователей за несколько секунд.** Взаимодействуйте с вашими клиентами, использующими любые устройства, с помощью технологии Push-уведомлений, персонализированными под нужды пользователей и отправленными в нужное время.

В этой лабораторной работе вы подключите универсальное приложение Windows (UWP) к мобильному бэкенду в Azure App Service. Сначала вы запустите службы мобильного бэкэнда локально на своём ПК и изучите код, для того чтобы понять, как клиентская часть мобильного приложения взаимодействует с серверной. Затем, вы добавите в клиентскую часть мобильного приложения аутентификацию через Azure Active Directory и подключите его к бэкэнду, размещённому на сервере в Microsoft Azure. Наконец, вы добавите в приложение работу с

оффлайн сценарием и синхронизацией - одну из самых мощных возможностей мобильной службы Azure App Service.

Цели

Настоящий курс научит вас:

- Запускать универсальное приложение Windows 10 (UWP), использующее бэкенд, расположенный локально и в облаке
- Подключать клиентское приложение к мобильному бэкэнду в Azure.
- Реализовать аутентификацию в клиентском приложении через Azure Active Directory.
- Реализовать оффлайн-сценарий работы приложения, с использованием локального хранилища данных SQLite и синхронизацией с мобильным бэкеном при восстановлении соединения.

Системные требования

Чтобы выполнить эту лабораторную работу, вам понадобится следующее программное обеспечение:

- Microsoft Windows 10
- Microsoft Visual Studio 2015

Дополнительно

Если вы хотите выполнить все шаги лабораторной работы, включая дополнительные, вам понадобится:

- Windows 10 Mobile Emulator или телефон под управлением Windows 10 Mobile

Настройка

Вам необходимо произвести следующие действия для подготовки своего компьютера к лабораторной работе:

1. Установите Microsoft Windows 10.
2. Установите Microsoft Visual Studio 2015. Выберите установку «Custom» и убедитесь, что в списке дополнительных функций выбран пункт «Universal Windows App Development Tools».
3. Дополнительно: Выберите пункт «Windows 10 Mobile Emulator».

Упражнения

Эта лабораторная работа включает в себя следующие упражнения:

1. Запуск мобильной службы Azure App Service
 2. Реализация аутентификации и подключение приложения к облаку
 3. Реализация оффлайн хранилища и синхронизации данных в приложении
-

Время для завершения этой лаборатории: **От 45 до 60 минут.**

Упражнение 1: Запуск мобильной службы Azure App Service

В этом упражнении, вы воспользуетесь универсальным приложением Windows 10 (UWP), которое является трекером для записи задач (вы можете загрузить аналогичное приложение с портала Azure, при создании мобильной службы). Вы подключите приложение-клиент к мобильной службе, которую запустите локально на своём компьютере.

Задание 1 – Знакомство с порталом Azure

Мы начнём с описания того, как создавать мобильный бэкенд для приложения на портале Azure.

1. Вам не нужно создавать экземпляр мобильной службы в Azure App Service для этой лабораторной работы, так как мы заранее подготовили его для вас.
Если вы хотите узнать, как самостоятельно создать мобильную службу Azure App Service, вам необходимо воспользоваться пошаговой инструкцией на портале:
<https://azure.microsoft.com/ru-ru/documentation/articles/app-service-mobile-dotnet-backend-windows-store-dotnet-get-started-preview/>.

Посмотрим, как создать новую мобильную службу в Azure:

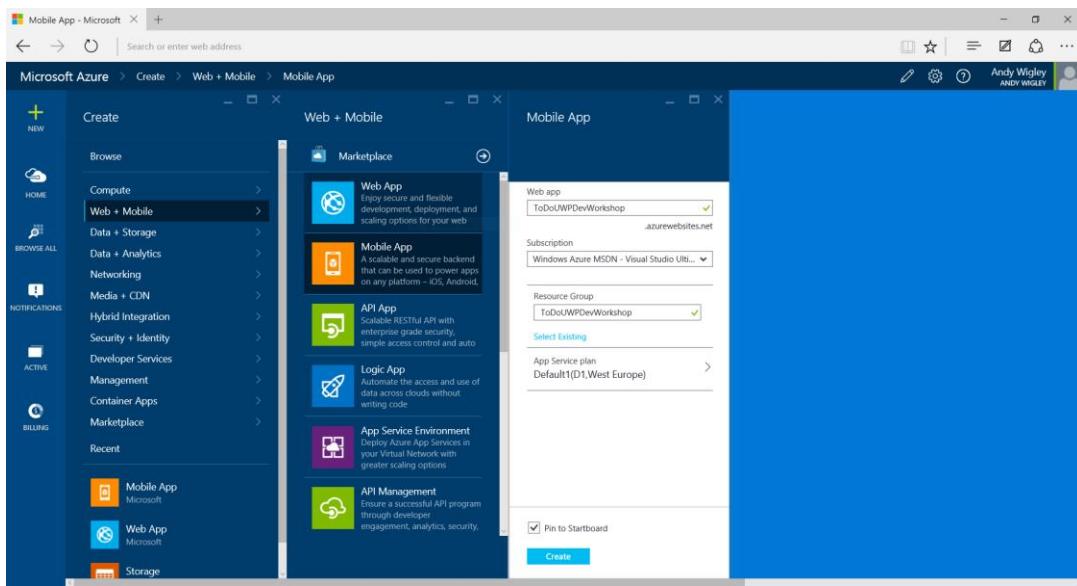


Рисунок 1

Начало создания мобильной службы на портале Azure.

2. После создания мобильной службы, вы увидите страницу с информацией:

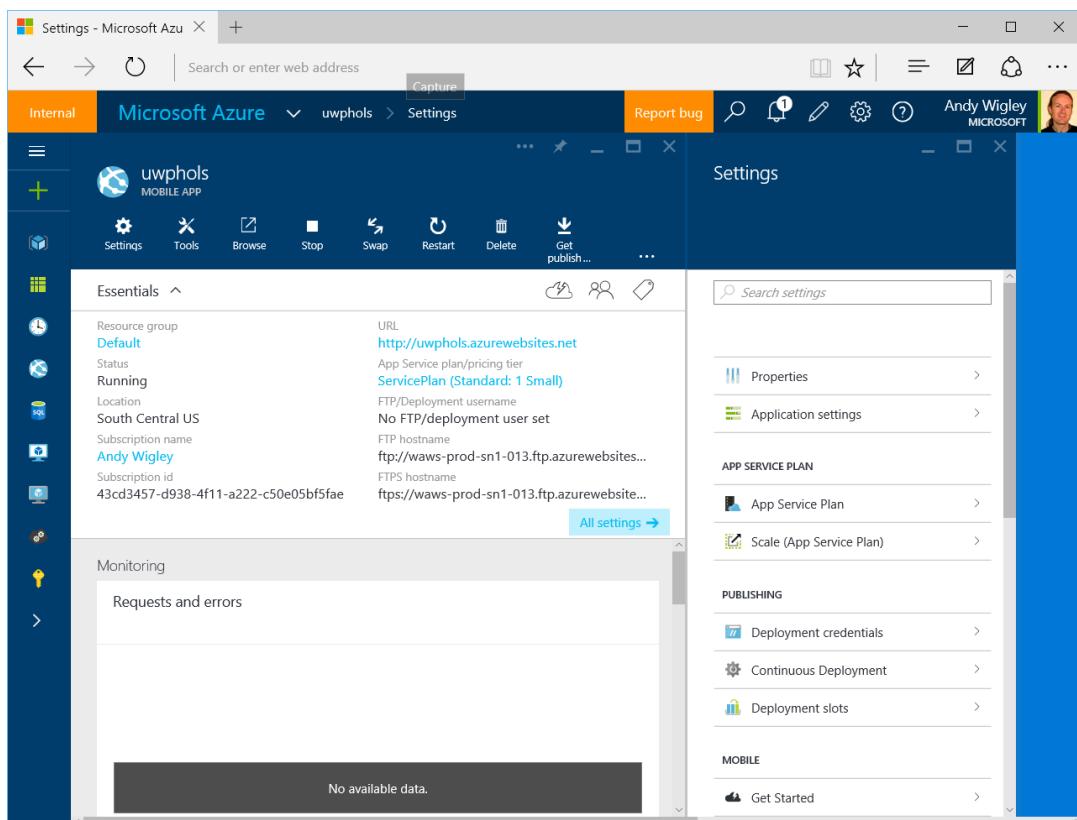


Рисунок 2

Блок информации о мобильной службе на портале Azure.

3. В блоке **Settings** вы найдете раздел **Mobile**, содержащий различные опции, включая **Get Started**. При нажатии на Get Started открывается панель Quickstart, где вы сможете получить инструкции для подключения вашего существующего приложения к мобильному бэкэнду в Azure или получите возможность загрузить примеры клиентской и серверной части мобильного приложения. Вы можете выбрать различные варианты мобильных приложений, включая Windows (C#), iOS (Objective-C), iOS (Swift), HTML/JavaScript, Xamarin.Android, Xamarin.iOS или Xamarin.Forms.

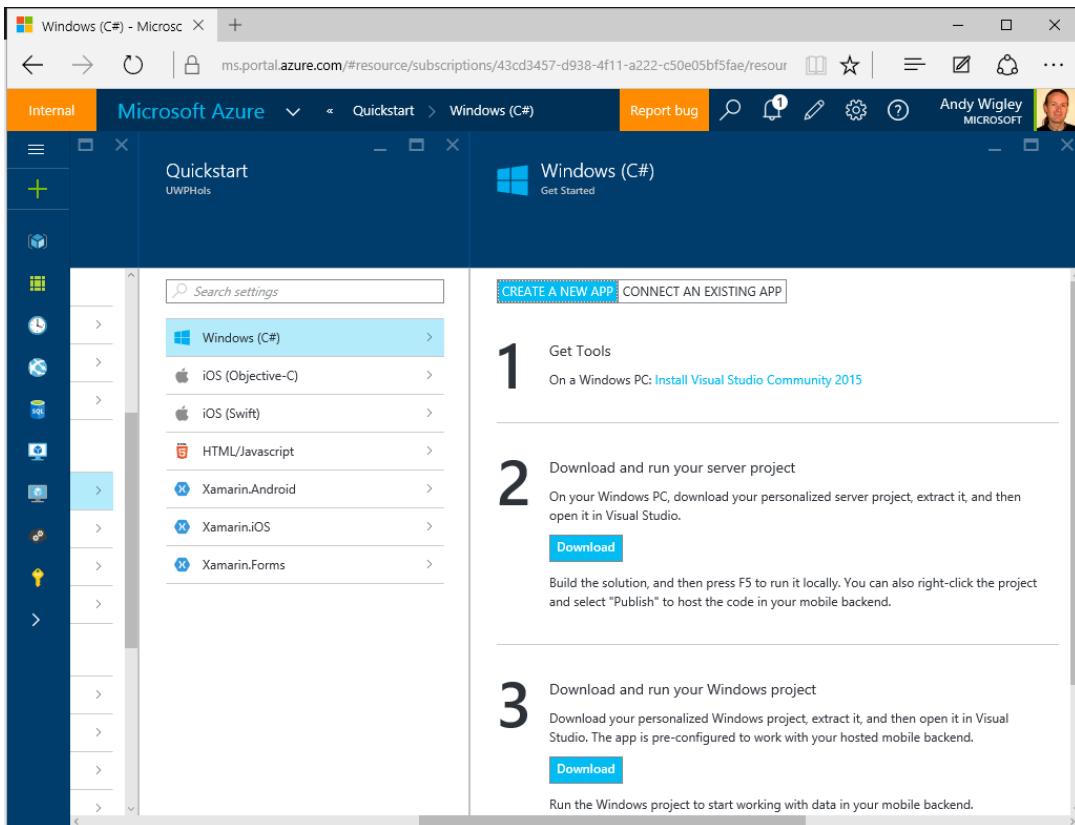


Рисунок 3

Quickstart панель на портале Azure.

4. На момент создания практикума (сентябрь 2015 года) для загрузки в качестве проекта Windows (C#) на Quickstart панели предлагается универсальное приложение Windows 8.1. Для этой лабораторной работы мы создали UWP версию этого клиента.

Задание 2 – Запуск серверной части приложения на своем ПК

Разберем код серверной части приложения, который можно загрузить при создании мобильного бэкенда в Azure. Вы сможете запустить его локально на своём ПК.

1. Откройте **Visual Studio 2015**. В меню **File** нажмите **Open Project/Solution**. Переместитесь в папку, в которой вы сохранили код для этой лабораторной работы, и откройте **Exercise 1\Begin-Cloud\UWPHolsService.sln**
2. Нажмите **Build Solution**, чтобы загрузить необходимые NuGet пакеты.

Примечание: Этот проект содержит C# код для .NET бэкенда в Azure. Вы так же можете выбрать реализацию логики бэкенда на NodeJS. Подробнее об этой возможности можно узнать на портале azure.com.

3. Этот проект содержит C# код для реализации логики бэкенда. В папке **DataObjects** откройте файл **ToDoltem.cs**. Этот класс определяет объекты данных, которые облачный сервис сохраняет в базе данных SQL Azure:

C#

```
public class TodoItem : EntityData
{
    public string Text { get; set; }

    public bool Complete { get; set; }
}
```

4. В папке **Controllers** откройте файл **TodoItemController.cs**. Этот класс содержит код, который определяет CRUD операции REST API облачного сервиса:

C#

```
public class TodoItemController : TableController<TodoItem>
{
    protected override void Initialize(
        HttpContext
controllerContext)
    {
        base.Initialize(controllerContext);
        UWPHolsContext context = new UWPHolsContext();
        DomainManager =
            new EntityDomainManager<TodoItem>(context, Request);
    }

    // GET tables/TodoItem
    public IQueryable<TodoItem> GetAllTodoItems()
    {
        return Query();
    }

    // GET tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
    public SingleResult<TodoItem> GetTodoItem(string id)
    {
        return Lookup(id);
    }

    // PATCH tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
    public Task<TodoItem> PatchTodoItem(string id, Delta<TodoItem>
patch)
    {
```

```

        return UpdateAsync(id, patch);
    }

    // POST tables/TodoItem
    public async Task<IHttpActionResult> PostTodoItem(TodoItem item)
    {
        TodoItem current = await InsertAsync(item);
        return CreatedAtRoute("Tables", new { id = current.Id },
current);
    }

    // DELETE tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
    public Task DeleteTodoItem(string id)
    {
        return DeleteAsync(id);
    }
}

```

5. В папке **App_Start** откройте файл **Startup.MobileApp.cs**. Этот файл содержит код для настройки службы, включая метод **Seed** класса **UWPHolInitialization**. Этот метод выполняется при первом запросе к REST сервису, и добавляет два элемента, созданных внутри этого метода, в базу данных (используется только на начальном этапе):

C#

```

protected override void Seed(UWPHolContext context)
{
    List<TodoItem> todoItems = new List<TodoItem>
    {
        new TodoItem { Id = Guid.NewGuid().ToString(),
                      Text = "First item", Complete = false },
        new TodoItem { Id = Guid.NewGuid().ToString(),
                      Text = "Second item", Complete = false },
    };

    foreach (TodoItem todoItem in todoItems)
    {
        context.Set<TodoItem>().Add(todoItem);
    }

    base.Seed(context);
}

```

6. Для Solution Configuration выберите Debug, а для Solution Platform выберите Any CPU. Выберите Microsoft Edge из списка Debug Target справа от кнопки Start Debugging.

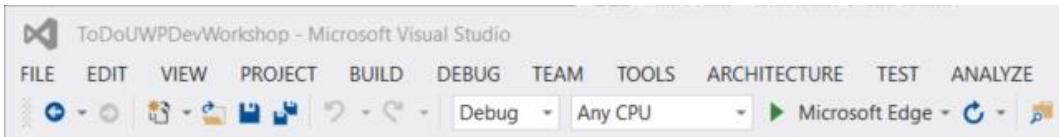


Рисунок 4

Настройка веб-сайта для запуска на Локальном компьютере.

Примечание: ▶ Start Debugging – кнопка запуска отладки.

7. Выполните сборку и запуск приложения. В браузере вы увидите стандартную страницу мобильной службы Azure.

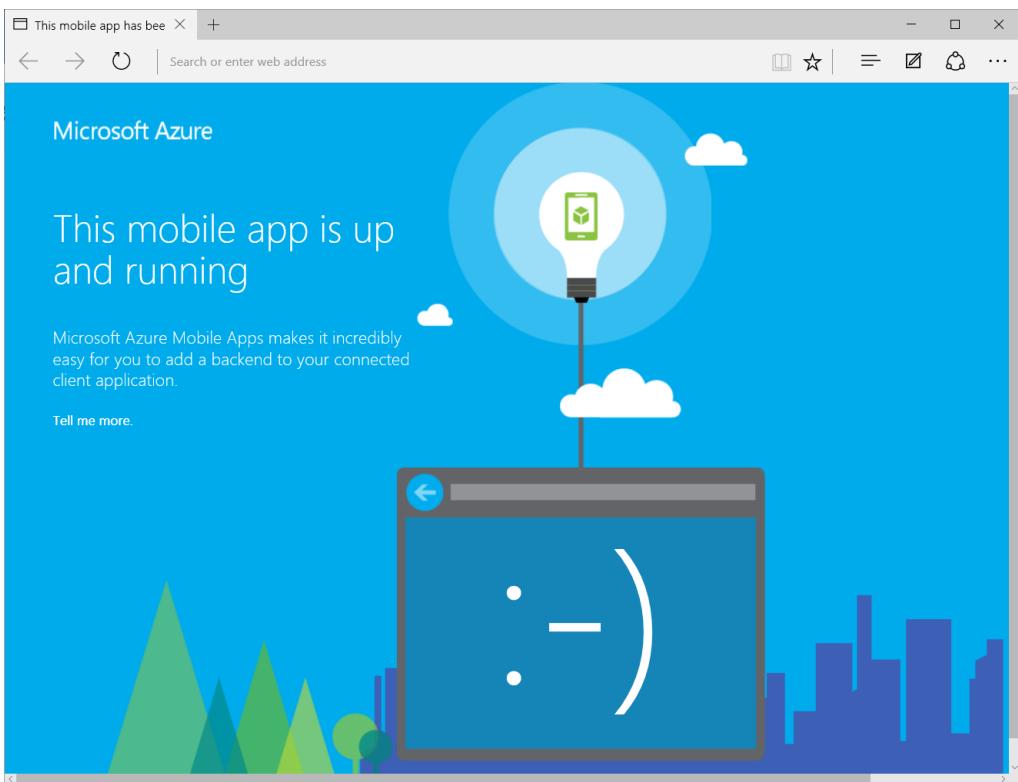


Рисунок 5

Мобильный бэкенд Azure, запущенный локально на вашем ПК.

Примечание: Возможность запуска бэкенда локально предоставляет вам еще один способ устранения неполадок в приложении.

Задание 3 – Запуск клиентской части приложения

Разберемся в коде UWP приложения и изменим его так, чтобы подключиться к мобильной службе, которая выполняется локально на вашем ПК.

1. Откройте второй экземпляр **Visual Studio 2015**. В меню **File** нажмите **Open Project/Solution**. Переместитесь в папку, где вы сохранили код для этой лабораторной работы, и откройте **Exercise 1\Begin-Client\ToDoUWPDevWorkshop.sln**.
2. Для Solution Configuration выберите Debug, а для Solution Platform выберите Any CPU x86. Выберите Local Machine из списка Debug Target справа от кнопки Start Debugging.

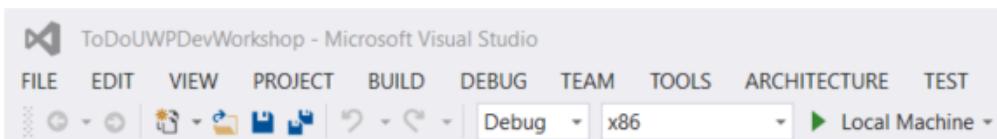


Рисунок 6

Настройка клиентского приложения для запуска на локальном компьютере.

3. Нажмите Build Solution, чтобы загрузить необходимые NuGet пакеты в проект.
4. Откройте файл **App.xaml.cs**. Обратите внимание, что в начале файла находится закомментированный код для создания экземпляра **MobileServiceClient**. Первые строчки необходимы, чтобы подключиться к облаку – оставьте этот код закомментированным. Следующий код подключает приложение к локальной версии облачного сервиса, который вы запустили в предыдущем задании.

```
C#
sealed partial class App : Application
{
    // Uncomment this code for configuring the MobileServiceClient to
    // communicate with the Azure Mobile Service and
    // Azure Gateway using the application key. You're all set to start
    // working with your Mobile Service!
    //public static MobileServiceClient MobileService =
    //    new MobileServiceClient(
    //        "https://uwpdevhols.azurewebsites.net",
    //        "",
    //        ""
    //    );

    // Use this code for configuring the MobileServiceClient to
    // communicate with your local
    // test project for debugging purposes.
    public static MobileServiceClient MobileService =
        new MobileServiceClient(
            "http://localhost:50781"
        );
}
```

5. Откройте файл **MainPage.xaml.cs**. В этом классе вы увидите, как в коде приложения реализуется обмен данными с облачным сервисом, загрузка данных из службы, добавление, обновление и удаление.

В начале файла объект **todoTable** имеет тип **IMobileServiceTable<TodoItem>** и инициализируется вызовом **App.MobileService.GetTable<TodoItem>()**. Этот объект используется по всему классу, чтобы выполнять операции для таблицы. Например, чтобы вставить новый элемент данных, вы используете метод **InsertAsync** объекта **todoTable**:

C#

```
private IMobileServiceTable<TodoItem> todoTable =
    App.MobileService.GetTable<TodoItem>();

...

private async Task InsertTodoItem(TodoItem todoItem)
{
    // This code inserts a new TodoItem into the database. When the
    // operation completes and Mobile Services has assigned an Id,
    // the item is added to the CollectionView
    await todoTable.InsertAsync(todoItem);
    items.Add(todoItem);

}
```

6. Выполните сборку и запуск приложения. Когда ваше приложение-клиент запущено, код в методе **OnNavigatedTo** в **MainPage** вызывает **RefreshTodosItems**, который в свою очередь вызывает службу бэкенда, чтобы извлечь все **ToDo** элементы, хранимые в базе данных бэкенда.

Примечание: Если ваше приложение прекращает работу с ошибкой **HttpRequestException**, то это может происходить, потому что вы остановили отладку приложения **UWPHelloService** из предыдущего задания. Запустите снова проект-сервер, оставьте его в запущенном состоянии и затем запустите приложение-клиент в отдельной копии **Visual Studio**.

7. Так как это был первый раз, когда был осуществлён доступ к службе, она автоматически настроит базу данных и запустит метод **Seed**, изученный нами ранее и добавляющий два сгенерированных внутри метода элемента. Через некоторое время вы увидите два элемента, отображаемые в интерфейсе приложения-клиента.

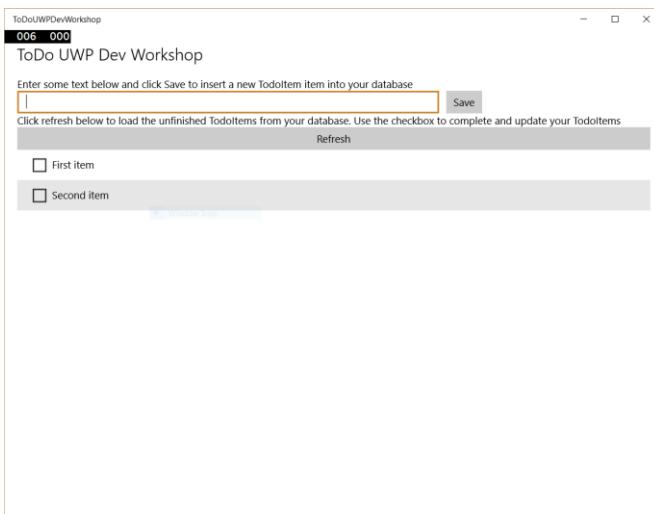


Рисунок 7

Приложение отображает два 'Seed' элемента из службы бэкенда.

8. Добавьте новые элементы к тем, которые уже есть.

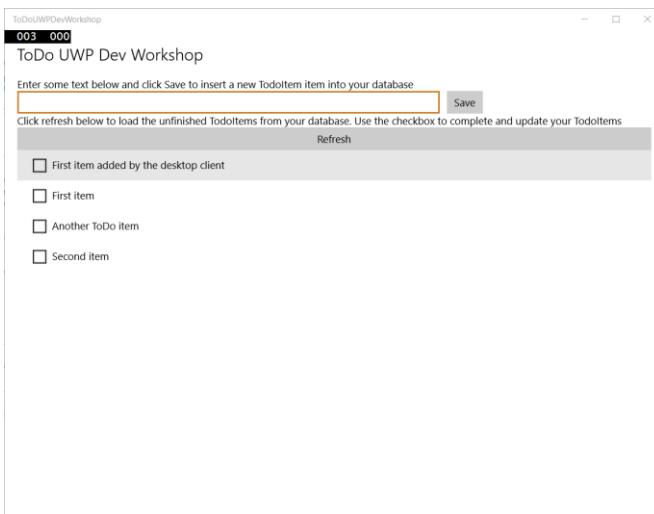


Рисунок 8

Добавление новых элементов.

9. Остановите отладку.

10. Если на вашем ПК установлен мобильный эмулятор Windows 10, выберите **Mobile Emulator** в качестве типа платформы для отладки. Если у вас подключено реальное мобильное устройство и включен режим разработчика, выберите **Device**. Если ни одна из этих опций недоступна, вам придется пропустить остальную часть этого упражнения.

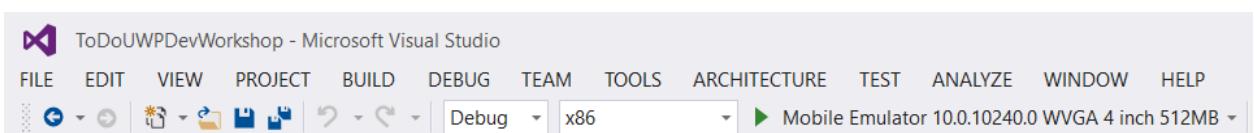


Рисунок 9

Установите Mobile Emulator или Device в Debug Target, если такие опции доступны.

11. Запустите отладку. Приложение запустится на вашем мобильном устройстве или в эмуляторе и отобразит те же элементы, которые вы видели в настольной версии к службе приложения-клиента и которые хранятся в базе данных мобильного бэкенда (в данный момент запущенного локально на вашем ПК).



Рисунок 10

Приложение на мобильном устройстве с Windows 10 использует тот же бэкенд.

12. Отключите отладку обоих проектов: клиента и сервера.

13. Вы завершили это упражнение и запустили мобильное приложение на двух различных устройствах, которые использовали данные из одного бэкенда. Вы реализовали онлайн сценарий работы с приложением.

Упражнение 2: Реализация аутентификации и подключение приложения к облачной службе

В этом упражнении, вы научитесь добавлять аутентификацию в UWP приложение, использующее мобильный бэкенд Azure. После успешной аутентификации и авторизации, вы увидите значение идентификатора пользователя. Вам предстоит изменить UWP приложение и подключить его к мобильной службе Azure App Service, выполняющейся в Microsoft Azure, вместо той, которая запущена локально. Мобильная служба в Azure заранее настроена и поддерживает аутентификацию через Azure Active Directory.

Задание 1 – Добавление аутентификации в приложение

Модифицируем UWP приложение, чтобы аутентифицировать пользователей до того, как приложение будет запрашивать данные из мобильной службы.

1. Откройте **Visual Studio 2015**. В меню **File** нажмите **Open Project/Solution**. Переместитесь в папку, в которой вы сохранили код для этой лабораторной работы, и откройте **Exercise 2\Begin-Client\ToDoUWPDevWorkshop.sln**.
2. Для Solution Configuration выберите Debug, а для Solution Platform выберите x86. Выберите Local Machine из списка Debug Target справа от кнопки Start Debugging.
3. Нажмите Build Solution, чтобы подключить необходимые NuGet пакеты.
4. Откройте файл **App.xaml.cs**. В этом упражнении код для создания экземпляра **MobileServiceClient** изменён - первые строки не закомментированы и вы используете их, чтобы подключиться к облачной службе. Следующий код подключает приложение к локальной версии облачного сервиса, которую вы использовали в предыдущем упражнении – сейчас эти строки закомментированы.

C#

```
sealed partial class App : Application
{
    // Uncomment this code for configuring the MobileServiceClient to
    // communicate with the Azure Mobile Service and
    // Azure Gateway using the application key. You're all set to start
    // working with your Mobile Service!
    public static MobileServiceClient MobileService =
        new MobileServiceClient(
            "https://uwpdevhols.azurewebsites.net",
            "");
```

```

);
// Use this code for configuring the MobileServiceClient to
// communicate with your local
// test project for debugging purposes.
//public static MobileServiceClient MobileService =
//    new MobileServiceClient(
//        "http://localhost:59989"
//);
...

```

5. В браузере пройдите по ссылке для мобильной службы Azure App Service:
<https://uwpdevhols.azurewebsites.net> Вы увидите стандартную информационную страницу мобильной службы Azure.

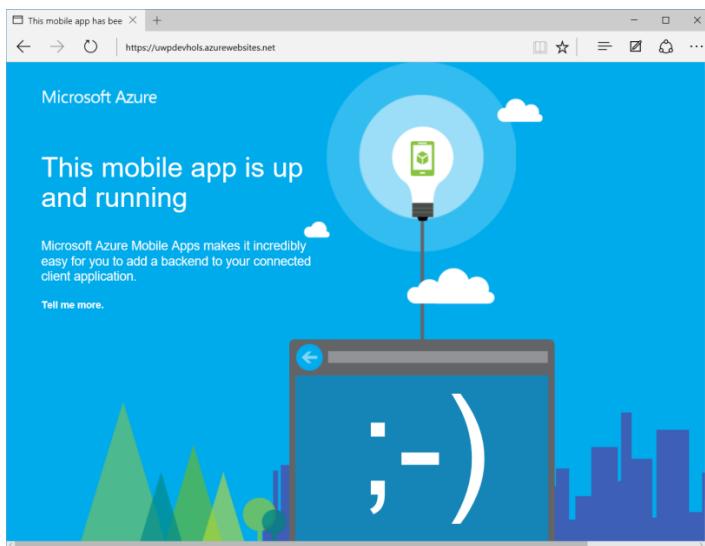


Рисунок 11

Мобильная служба, запущенная в Microsoft Azure.

6. Этот облачный сервис немного отличается от того, который был запущен локально в предыдущем упражнении. Он настроен так, чтобы доступ предоставлялся только аутентифицированным пользователям. Чтобы продемонстрировать это, запустите клиент **TodoUWPDevWorkshop**. Приложение не сможет загрузить элементы из службы, потому что пользователь приложения не был аутентифицирован через настроенного провайдера.

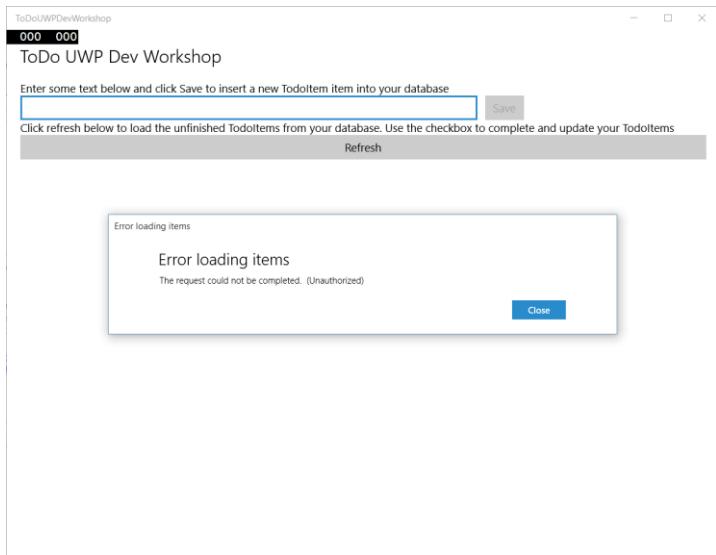


Рисунок 12

Мобильная служба Azure требует пройти аутентификацию для доступа в приложение.

Примечание: Мобильная служба Azure App Service поддерживает аутентификацию через:

- Azure Active Directory
- Facebook
- Google
- Microsoft
- Twitter

Служба, к которой вы подключаетесь, настроена на поддержку аутентификации через Azure Active Directory.

Подробные инструкции о том, как настроить один из таких провайдеров в мобильной службе Azure App Service, можно посмотреть в документации, в разделе **Authenticate Users:** <https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-dotnet-backend-windows-store-dotnet-get-started-users-preview/>

7. Теперь вы можете добавить аутентификацию в приложение. Откройте файл **MainPage.xaml.cs**. Определите переменную для хранения зарегистрированного пользователя и способ аутентификации.

C#

```
// Define a member variable for storing the signed-in user.  
private MobileServiceUser user;  
  
// Define a method that performs the authentication process  
// using an Azure Active Directory sign-in.  
private async System.Threading.Tasks.Task AuthenticateAsync()  
{  
    while (user == null)  
    {  
        string message;
```

```

    // This sample uses the Azure Active Directory provider.
    var provider = "AAD";

    try
    {
        // Sign-in using AAD authentication.
        user = await App.MobileService.LoginAsync(provider);
        message =
            string.Format("You are now signed in - {0}", user.UserId);
    }
    catch (InvalidOperationException)
    {
        message = "You must log in. Login Required";
    }

    var dialog = new MessageDialog(message);
    dialog.Commands.Add(new UICommand("OK"));
    await dialog.ShowAsync();
}
}

```

- Закомментируйте или удалите вызов метода **RefreshTodoItems** в существующем переопределении метода **OnNavigatedTo**. Это предотвратит загрузку данных до аутентификации пользователя. Затем добавьте кнопку Sign in, которая будет запускать процесс аутентификации.
- Добавьте следующий метод в класс **MainPage**:

C#

```

private async void ButtonLogin_Click(object sender, RoutedEventArgs e)
{
    // Login the user and then load data from the mobile app.
    await AuthenticateAsync();

    // Hide the login button and load items from the mobile app.
    this.ButtonLogin.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
    await RefreshTodoItems();
}

```

- Откройте файл **MainPage.xaml** и добавьте **Button** в элемент **StackPanel** в строчке Row 0 таблицы Grid, перед элементом **TextBlock**:

XAML

```

<StackPanel Grid.Row="0" Grid.ColumnSpan="2" >

    <Button Name="ButtonLogin" Click="ButtonLogin_Click"
        Visibility="Visible">Sign in</Button>
    <TextBlock Style="{StaticResource BodyTextBlockStyle}" Text="Enter some
        text below and click Save to insert a new TodoItem item into your database"
        TextWrapping="Wrap"/>
</StackPanel>

```

11. Прежде чем запустить приложение, вам необходимо получить имя пользователя и пароль, чтобы использовать их во время аутентификации через Azure Active Directory. Мы заранее подготовили 500 пользователей AAD специально для этих лабораторных работ. Чтобы узнать, какое имя пользователя и пароль вы можете использовать, необходимо зайти на сайт: <http://uwpholsusers.azurewebsites.net/>

Вы увидите страницу, на которой будут указаны имя пользователя и пароль:

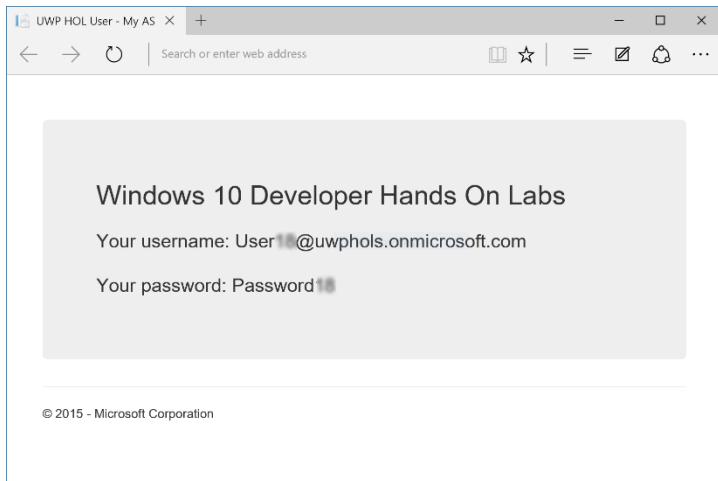


Рисунок 13

Данные учетной записи в AAD.

Примечание: каждый запрос к этой странице возвращает разные имя пользователя и пароль из 500 настроенных для этой лабораторной работы. Вы должны оказаться единственным, кто использует полученные вами данные во время лабораторной работы, однако это условие не гарантируется. Поскольку идентификатор пользователя используется для разделения данных каждого пользователя в базе данных SQL Azure, то если кто-то еще будет использовать данные вашей учетной записи, вы увидите это в качестве элементов ToDo, созданных без вашего участия. Естественно, для реальных приложений, данные учетной записи не распространяются подобным образом.

12. Нажмите **F5**, чтобы запустить приложение, нажмите Sign In, и зайдите в приложение, используя полученные логин и пароль.

13. Когда вы войдете, приложение запуститься без ошибок, и вы должны будете получить доступ к вашему мобильному приложению и к возможностям добавлять ToDo элементы.

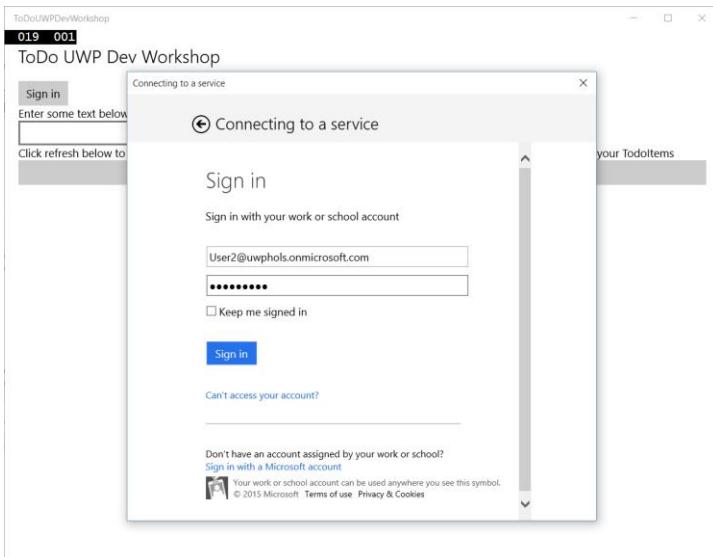


Рисунок 4

Аутентификация через Azure Active Directory.

Примечание: Мобильная служба Azure App Service, к которой вы подключены, используется и другими пользователями сервиса. Все данные для пользователей хранятся в одной базе данных SQL Azure, и все пользователи имеют доступ к REST сервису в Azure.

Для поддержки многопользовательского режима, необходимо было произвести некоторые изменения в коде службы запущенной в облаке, что отличает ее от службы, запущенной локально в Упражнении 1.

Вы можете изучить код, выполняющийся в службе, к которой вы подключены, открыв **Exercise 2\Cloud\UwpDevHols.sln**. Ниже перечислены изменения:

- В файле **DataObjects\TodoItem.cs** добавлено дополнительное поле **UserId** для хранения идентификатора пользователя для каждого элемента ToDo
- В файле **Controllers\TodoItemController.cs** в методе **GetAllTodos** добавлен код, который получает идентификатор пользователя и использует его, чтобы выполнить запрос только тех ToDo элементов, которые связаны с пользователем; в методе **PostTodoItem** добавлена логика получения идентификатора пользователя и сохранения его в поле **UserId** каждого ToDo элемента.
- Добавлен дополнительный атрибут **[Authorize]** в класс **TodoItemController**, который ограничивает доступ не аутентифицированным пользователям

Задание 2 – Сохранение токен аутентификатора на клиенте

В предыдущем задании вы реализовали стандартный вход в приложение, который требует от клиента подключаться и к провайдеру идентификатора, и к мобильному сервису каждый раз, когда запускается приложение. Этот метод не всегда эффективен. Лучше реализовать подход, когда производится кэширование токена аутентификатора, возвращенного службой, и постараться использовать его прежде чем вход через провайдера.

1. Продолжите работу с **ToDoUWPDevWorkshop.sln** из предыдущего задания.
2. В файл **MainPage.xaml.cs** подключите следующие библиотеки:

C#

```
using System.Linq;
using Windows.Security.Credentials;
```

3. Замените метод **AuthenticateAsync** следующим кодом:

C#

```
// Define a method that performs the authentication process
// using an Azure Active Directory sign-in.
private async System.Threading.Tasks.Task AuthenticateAsync()
{
    string message = string.Empty;
    // This sample uses the Azure Active Directory provider.
    var provider = "AAD";

    // Use the PasswordVault to securely store and access credentials.
    PasswordVault vault = new PasswordVault();
    PasswordCredential credential = null;

    while (credential == null)
    {
        try
        {
            // Try to get an existing credential from the vault.
            credential =
        vault.FindAllByResource(provider).FirstOrDefault();
        }
        catch (Exception)
        {
            // When no matching resource an error occurs, which we ignore.
        }

        if (credential != null)
        {
            // Create a user from the stored credentials.
            user = new MobileServiceUser(credential.UserName);
            credential.RetrievePassword();
            user.MobileServiceAuthenticationToken= credential.Password;

            // Set the user from the stored credentials.
            App.MobileService.CurrentUser = user;
        }
    }
}
```

```

        try
    {
        // Try to return an item now to determine if the
        // cached credential has expired.
        await App.MobileService.GetTable<TodoItem>().Take(1)
            .ToListAsync();
    }
    catch (MobileServiceInvalidOperationException ex)
    {
        if (ex.Response.StatusCode ==
            System.Net.HttpStatusCode.Unauthorized)
        {
            // Remove the credential with the expired token.
            vault.Remove(credential);
            credential = null;
            continue;
        }
    }
}
else
{
    try
    {
        // Login with the identity provider.
        user = await App.MobileService
            .LoginAsync(provider);

        // Create and store the user credentials.
        credential = new PasswordCredential(provider,
            user.UserId, user.MobileServiceAuthenticationToken);
        vault.Add(credential);
    }
    catch (InvalidOperationException)
    {
        message = "You must log in. Login Required";
    }

    var dialog = new MessageDialog(message);
    dialog.Commands.Add(new UICommand("OK"));
    await dialog.ShowAsync();
}
}

```

В этой версии метода `AuthenticateAsync` для доступа к мобильной службе приложение пытается использовать данные, которые хранятся в `PasswordVault`. Отправляется простой запрос, чтобы убедиться, что срок действия сохраненного токена не закончился. Когда возвращается код 401, осуществляется вход в систему через провайдера. Также обычный вход в систему осуществляется, когда отсутствуют сохраненные идентификационные данные.

Примечание: Срок действия токен может истечь и после аутентификации, когда приложение уже используется. Чтобы отслеживать такие ситуации, воспользуйтесь

информацией из материалов [Caching and handling expired tokens in Azure Mobile Services managed SDK](#):

<http://blogs.msdn.com/b/carlosfigueira/archive/2014/03/13/caching-and-handling-expired-tokens-in-azure-mobile-services-managed-sdk.aspx>

4. Перезапустите приложение дважды.

Во время первого запуска осуществите вход в систему через форму провайдера. Во время второго запуска будут использоваться сохраненные идентификационные данные и произойдет вход в систему.

5. **[Дополнительно]** Одной из возможностей PasswordVault является следующее - идентификационные данные, которые вы храните, попадают через облако во все ваши устройства, где установлено это приложение. Если у вас есть второе устройство с операционной системой Windows 10 или Windows 10 Mobile, например, телефон, планшет или ПК, и это устройство связано с тем же аккаунтом Microsoft, что и ваш ПК, вы можете протестировать эту возможность, установив то же UWP приложение на это устройство.

Если вы подождете, пока завершится перемещение идентификационных данных, то когда вы войдете в приложение на одном из устройств, оно обнаружит хранилище идентификационных данных в **PasswordVault** и вам не придется заново заходить в систему, если только срок токена не истек.

Упражнение 3: Реализация онлайн хранилища и синхронизации данных в приложении

В этом упражнении, вы реализуете поддержку онлайн режима в UWP приложении при помощи мобильного бэкенда в Azure. Синхронизация в онлайн сценарии позволяет использовать функции просмотра, добавления или изменения данных в приложении, даже при отсутствии соединения с сетью. Изменения сохраняются в локальной базе данных. Как только устройство вновь получает доступ к сети, изменения синхронизируются с удаленным хранилищем на бэкенде.

В этом упражнении вы модифицируете приложение Windows, добавив поддержку онлайн возможностей с мобильной службой Azure.

Задание 1 – Реализация онлайн сценария работы с мобильным приложением

Онлайн сценарий с мобильной службой Azure позволяет сохранять данные и взаимодействовать с локальной базой данных. Для того чтобы воспользоваться этими возможностями в вашем приложении, необходимо инициализировать метод **MobileServiceClient.SyncContext** для локального хранилища. Затем связать таблицу через интерфейс **IMobileServiceSyncTable**. В этой лабораторной работе мы будем использовать в качестве локального хранилища SQLite.

Откройте проект ToDoUWPDevWorkshop, который вы использовали в предыдущем упражнении.

1. Переместитесь в папку, где вы сохранили **ToDoUWPDevWorkshop** приложение-клиент из Упражнения 2. Откройте **ToDoUWPDevWorkshop.sln** в Visual Studio 2015.
2. Установите SQLite для UWP приложения.
 - a. В Visual Studio, в меню **Tools** нажмите на **Extensions and Updates**
 - b. В левой панели **Extensions and Updates**, нажмите **Online**
 - c. В поле поиска, введите **SQLite**
 - d. Когда отобразятся результаты поиска, прокручивайте список, пока не увидите **SQLite for Universal App Platform**. Если этот SDK еще не установлен, выберите этот элемент и нажмите на кнопку **Download**.

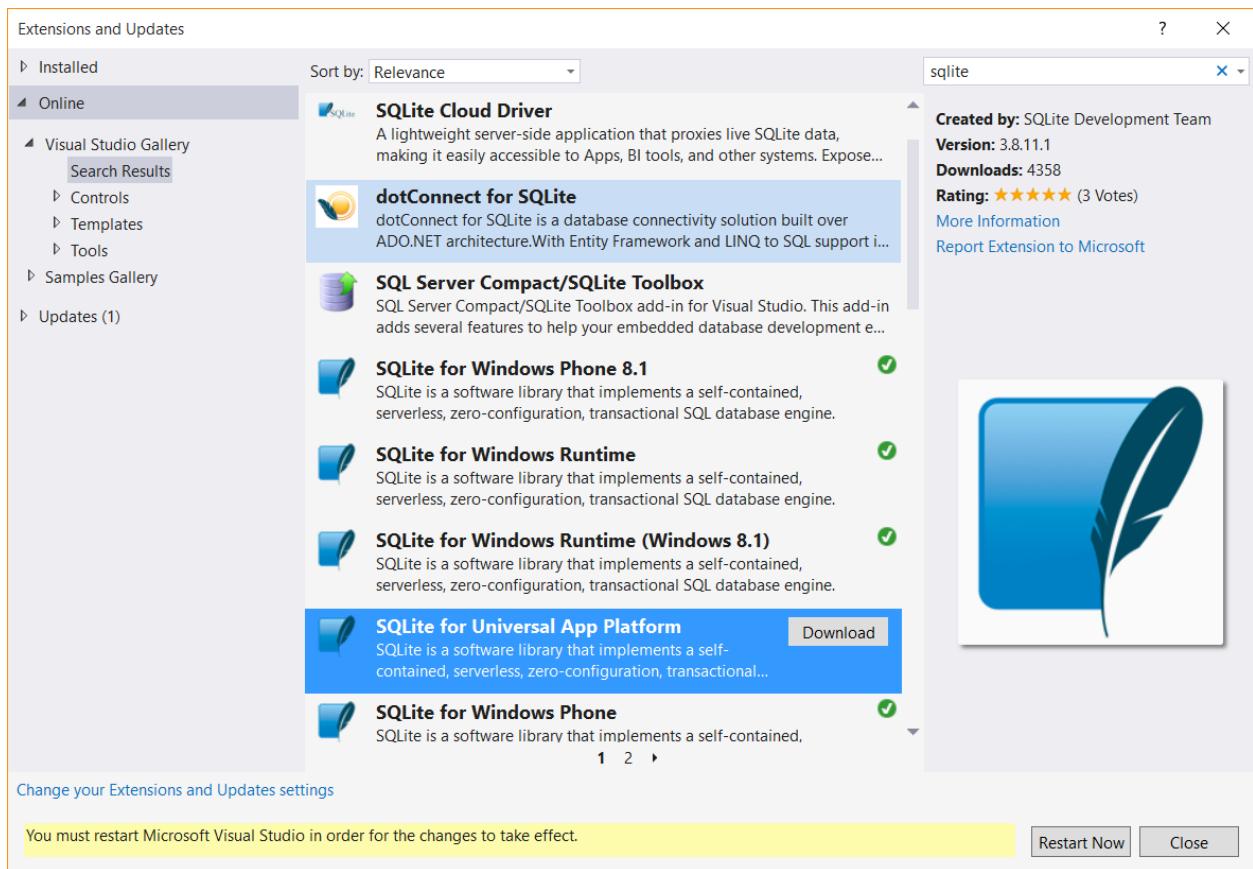


Рисунок 15

Скачайте и установите SQLite для Universal App Platform SDK.

- e. Когда на экране появится UAC окно, нажмите **OK**.
 - f. В окне VSIX Installer нажмите **Install**. После того, как расширение установится, нажмите **Close**.
 - g. Нажмите на кнопку **Restart Now** в окне Extensions and Updates и подождите пока перезапустится Visual Studio 2015.
3. Добавьте ссылку на библиотеку SQLite в ваш проект.
- a. В Solution Explorer нажмите правой кнопкой на References и затем выберете Add Reference, чтобы запустить Reference Manager.
 - b. В категории Universal Windows выберите пункт Extensions в панели слева.

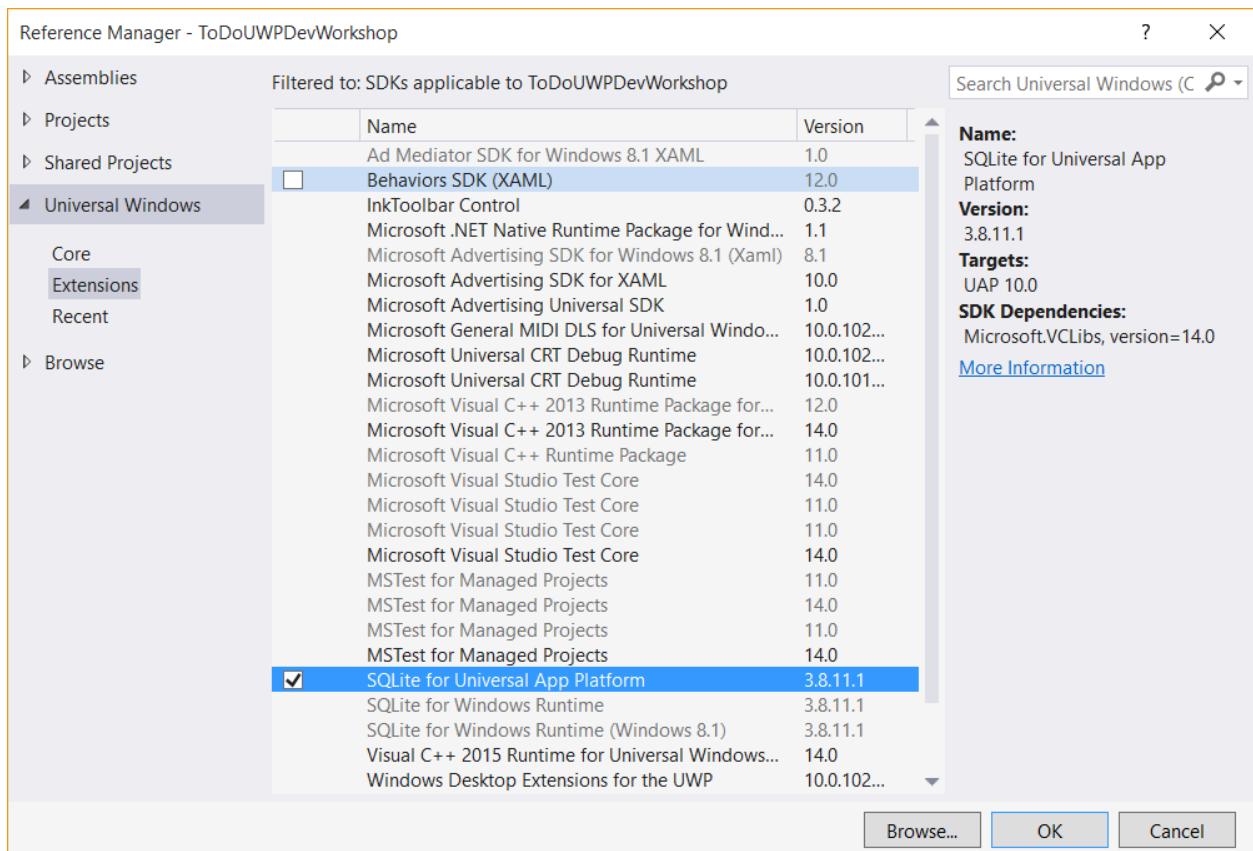


Рисунок 14

Добавление ссылки на библиотеку SQLite к проекту.

- c. Выберите **SQLite для Universal App Platform** и затем **нажмите OK**.
4. Установите NuGet пакет WindowsAzure.MobileServices.SQLiteStore.
 - a. В Solution Explorer правой кнопкой нажмите на проект и выберите **Manage Nuget Packages**, чтобы запустить NuGet Package Manager.
 - b. На вкладке Online выберите опцию **Include Prerelease**. Выполните поиск **SQLiteStore**, чтобы затем выбрать **WindowsAzure.MobileServices.SQLiteStore**.
 - c. Затем нажмите **Install**, чтобы добавить ссылку на NuGet пакет в проект.
 - d. Нажмите **I Accept** в окне License Acceptance.
5. В Solution Explorer откройте **файл MainPage.cs**. Раскомментируйте следующие строки:

```
C#
using Microsoft.WindowsAzure.MobileServices.SQLiteStore; // offline sync
using Microsoft.WindowsAzure.MobileServices.Sync;           // offline sync
```

6. В файле MainPage.cs закомментируйте код, который инициализирует todoTable через IMobileServiceTable. И раскомментируйте код, который инициализирует todoTable через IMobileServiceSyncTable:

C#

```
//private IMobileServiceTable<TodoItem> todoTable =  
App.MobileService.GetTable<TodoItem>();  
private IMobileServiceSyncTable<TodoItem> todoTable =  
    App.MobileService.GetSyncTable<TodoItem>(); // offline sync
```

7. В файле **MainPage.cs** в разделе, отмеченном **Offline sync**, раскомментируйте методы **InitLocalStoreAsync** и **SyncAsync**. **InitLocalStoreAsync** инициализирует синхронизацию данных на клиенте и в SQLite. В Visual Studio вы можете выделить все закомментированные строки и использовать Ctrl+K+U, чтобы раскомментировать их.

Обратите внимание, что в **SyncAsync** push операции выполняет метод **MobileServiceClient.SyncContext** вместо **IMobileServicesSyncTable**. Это необходимо для того, чтобы отслеживать изменения, произведенные на клиенте для всех таблиц. Это подходит в случаях, где таблицы связаны между собой.

Дополнительная информация доступна в документации, в разделе [Offline Data Sync в Azure Mobile Apps](#): <https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-offline-data-sync-preview/>

C#

```
private async Task InitLocalStoreAsync()  
{  
    if (!App.MobileService.SyncContext.IsInitialized)  
    {  
        var store = new MobileServiceSQLiteStore("localstore.db");  
        store.DefineTable<TodoItem>();  
        await App.MobileService.SyncContext.InitializeAsync(store);  
    }  
  
    await SyncAsync();  
}  
  
private async Task SyncAsync()  
{  
    await App.MobileService.SyncContext.PushAsync();  
    await todoTable.PullAsync("todoItems", todoTable.CreateQuery());  
}
```

8. В обработчике события **OnNavigatedTo** раскомментируйте вызов **InitLocalStoreAsync**:

C#

```
protected override async void OnNavigatedTo(NavigationEventArgs e)  
{  
    await InitLocalStoreAsync(); // offline sync  
    // await RefreshTodoItems();  
}
```

9. Раскомментируйте 3 вызова SyncAsync в методах InsertTodoItem, UpdateCheckedTodoItem и ButtonRefresh_Click

C#

```
private async Task InsertTodoItem(TodoItem todoItem)
{
    await todoTable.InsertAsync(todoItem);
    items.Add(todoItem);

    await SyncAsync(); // offline sync
}

...

private async Task UpdateCheckedTodoItem(TodoItem item)
{
    await todoTable.UpdateAsync(item);
    items.Remove(item);
    ListItems.Focus(Windows.UI.Xaml.FocusState.Unfocused);

    await SyncAsync(); // offline sync
}

private async void ButtonRefresh_Click(object sender, RoutedEventArgs e)
{
    ButtonRefresh.IsEnabled = false;

    await SyncAsync(); // offline sync
    await RefreshTodoItems();

    ButtonRefresh.IsEnabled = true;
}
```

10. Измените код в SyncAsync, чтобы добавить дополнительную обработку исключений. В режиме оффлайн MobileServicePushFailedException будет перехвачен с PushResult.Status == CancelledByNetworkError.

C#

```
private async Task SyncAsync()
{
    String errorString = null;

    try
    {
        await App.MobileService.SyncContext.PushAsync();
        // first param is query ID, used for incremental sync
        await todoTable.PullAsync("todoItems", todoTable.CreateQuery());
    }
}
```

```

    catch (MobileServicePushFailedException ex)
    {
        errorString = "Push failed because of sync errors. " +
                      "You may be offline.\nMessage: " +
                      ex.Message + "\nPushResult.Status: " +
                      ex.PushResult.Status.ToString();
    }
    catch (Exception ex)
    {
        errorString = "Pull failed: " + ex.Message +
                      "\n\nIf you are still in an offline scenario, " +
                      "you can try your Pull again when connected with " +
                      "your Mobile Service.";
    }

    if (errorString != null)
    {
        MessageDialog d = new MessageDialog(errorString);
        await d.ShowAsync();
    }
}

```

Примечание: MobileServicePushFailedException может произойти в обоих случаях, с pull и push методами. В операции pull это может произойти, потому что pull метод внутри себя вызывает push, для того чтобы убедиться, что таблицы согласованы между собой.

11. В Visual Studio нажмите F5, чтобы выполнить сборку и запустить приложение. Приложение-клиент будет вести себя так же, как и до добавления оффлайн возможностей. Тем не менее, эти изменения позволяют создать локальную базу данных, которая будет использоваться в режиме оффлайн, и заполнить ее данными. Мы протестируем режим оффлайн в следующем задании.

Задание 2 – Проверка работы приложения в оффлайн режиме

В этом упражнении вы запустите приложение-клиент и проверите его работу в режиме оффлайн. Когда вы добавите данные, обработчик исключений сообщит вам, что приложение работает в оффлайн режиме с **PushResult.Status == CancelledByNetworkError**. Добавленные элементы будут храниться в локальном хранилище, но не будут синхронизированы с мобильным бэкендом, пока вы не восстановите соединение.

1. Добавьте ToDo элементы, чтобы убедиться, что оффлайн режим приложения работает.
 - Вариант 1:** Для того, чтобы протестировать оффлайн режим приложения для вашего ПК, нажмите в трее на иконку Notifications, для того чтобы открыть Notification Center. В панели быстрых задач включите Flight Mode.
 - Вариант 2:** Поменяйте тип устройства для отладки на эмулятор Windows 10 Mobile. Установите на эмулятор приложение. Когда эмулятор запустится, нажмите **кнопку >>** на панели команд, чтобы увидеть дополнительные возможности эмулятора.

В Network выберите **Enable Network simulation** и установите для **Network speed** значение **No network**.

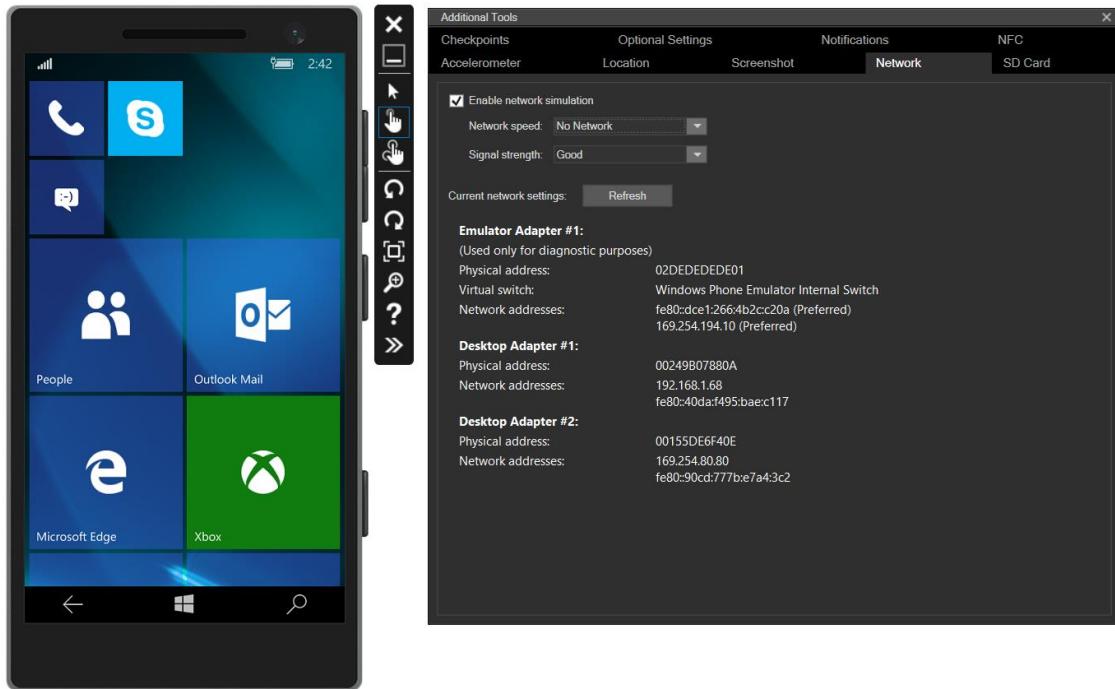


Рисунок 17

Настройка эмулятора Windows 10 Mobile для работы в режиме оффлайн.

2. Нажмите F5 для создания и запуска приложения. Обратите внимание, что при запуске приложения, синхронизация не произошла.
3. Добавьте новые ToDo элементы и нажмите Save. Для каждого элемента Push метод вызовет исключение PushResult.Status=CancelledByNetworkError. Новые ToDo элементы существуют только в локальном хранилище до того момента, пока они не будут отправлены в бэкенд мобильного приложения.
Вы можете обработать возникающие исключения **PushResult.Status=CancelledByNetworkError** таким образом, чтобы приложение – клиент вело себя так, как если бы оно было соединено с мобильной службой.
4. Закройте приложение и запустите его снова, чтобы убедиться, что элементы, которые вы создали, до сих пор отображаются в приложении и хранятся в локальном хранилище.

Задание 3 – Повторное подключение приложения к облачной службе

В этом упражнении вы снова подключите приложение к мобильному бэкенду. Приложение перестроится из работы в режиме оффлайн в режим онлайн работы. Когда вы первый раз запустите приложение, обработчик событий **OnNavigatedTo** вызовет **InitLocalStoreAsync**. Это приведет к тому, что вызовется метод **SyncAsync** для синхронизации локального хранилища с удаленным. Таким образом, приложение попытается синхронизировать данные после запуска.

1. Переведите приложение в режим онлайн.
 - a. **Вариант 1:** Если вы используете десктоп версию приложения, то отключите Flight Mode и верните ваше устройство в режим онлайн.
 - b. **Вариант 2:** Если вы тестируйте оффлайн режим при помощи мобильного эмулятора, нажмите >> на панели команд, в разделе **Network** отключите **Enable network simulation**.
2. Нажмите **F5** для сборки и запуска приложения. Приложение синхронизирует локальные изменения с мобильным бэкендом в Azure с помощью операций pull и push в тот момент, когда вызывается обработчик события **OnNavigatedTo**.
3. **[Дополнительно]** Если у вас есть другое устройство с установленным приложением, запустите приложение на этом устройстве с теми же данными учетной записи, что и прежде. Проверьте, что ToDo элементы, созданные в оффлайн режиме синхронизированы с мобильным бэкендом в Azure и доступны на втором устройстве.
4. В приложении поставьте галочку напротив некоторых ToDo элементов, чтобы отметить их как завершенные и сохранить эти данные в локальном хранилище.

UpdateCheckedTodoItem вызовет SyncAsync для синхронизации информации о завершенных ToDo элементах с бэкендом. SyncAsync вызовет pull и push операции. Имейте ввиду, что при выполнении операции pull для таблицы, в которой на клиенте произошли изменения, операция push будет производиться автоматически. Для дополнительной информации обратитесь к документации на портале в разделе [Offline Data Sync в Azure Mobile Apps: https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-offline-data-sync-preview/](https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-offline-data-sync-preview/)

Заключение

В этой лабораторной работе вы изучили, как подключить универсальное приложение Windows (UWP) к мобильному бэкенду в Azure App Service и запустили службы мобильного бэкэнда локально на своём ПК. Вы так же добавили в клиент мобильного приложения аутентификацию через Azure Active Directory и подключили приложение к мобильному бэкэнду, размещённому на сервере в облаке Microsoft Azure. Наконец, вы реализовали сценарий работы приложения в режиме оффлайн с последующей синхронизацией локального и удаленного хранилищ.



Лабораторный практикум

Запуск приложений при помощи
голосовых команд

Октябрь 2015 года



Обзор

Приложения Windows 10 могут использовать интерфейс Cortana, чтобы взаимодействовать с пользователями через удобные и настраиваемые голосовые команды. Cortana может запустить ваше приложение на переднем плане, или взаимодействовать с данными приложения в фоновом режиме.

Специфические для приложения голосовые команды начинаются с префикса, как правило, с имени приложения или ключевого слова, что позволяет устраниć неоднозначность в отношении других приложений, которые могут иметь аналогичные команды. Вы можете определить опции произнесения имени приложения перед или после команды и выбрать схему поведения после запуска приложения.

В этой работе вы создадите файл определения голосовых команд и будете добавлять команды, чтобы запустить своё приложение и работать с ним.

Цели

Эта лабораторная работа покажет вам как:

- Создать файл определения голосовых команд
- Запустить своё приложение, используя голосовую команду
- Использовать переключение для выбора входящих голосовых команд
- Передавать информацию из голосовой команды в приложение
- Переключать вид приложения по входящей голосовой команде
- Запускать фоновую задачу из голосовой команды
- Возвращать письменные и голосовые ответы Cortana из своего приложения

Системные требования

Чтобы выполнить этот практикум, необходимо обладать следующим набором программных инструментов:

- Microsoft Windows 10 настраивается на один из языков и регионов, в которых поддерживается Cortana
 - В сентябре 2015 года Cortana стала доступной в следующих странах/регионах: Китай, Франция, Германия, Италия, Испания, Соединённое Королевство и Соединённые Штаты. Cortana доступна на следующих языках: Китайский (упрощённый), английский (Великобритания), английский (США), французский, итальянский, немецкий и испанский языки
 - Для того, чтобы пользоваться Cortana, необходимо произвести все настройки на один и тот же язык:
 - ◆ Языки (язык вашего устройства)
 - ◆ Разговорный язык (должен быть установлен языковой пакет)
 - ◆ Страна или регион
 - Microsoft Visual Studio 2015
-

Настройка

Вы должны осуществить следующие шаги для подготовки своего компьютера для этого курса:

1. Установите Microsoft Windows 10.
 2. Установите Microsoft Visual Studio 2015.
-

Упражнения

Этот лабораторный практикум включает следующие упражнения:

1. Запуск при помощи голосовых команд
 2. Передача голосового параметра в приложение
 3. Ответ на голосовую команду при помощи фоновой задачи
-

Расчётное время для завершения работы: **От 30 до 45 минут.**

Упражнение 1: Запуск при помощи ГОЛОСОВЫХ КОМАНД

Голосовые команды представляют удобную альтернативу ручному запуску вашего приложения. Для использования голосовых команд вам нужно создать файл голосового описания, указывающий команду запуска вашего приложения и соответствующий ответ Cortana. Зарегистрируйте команды с Cortana и используйте точку старта OnActivated для старта приложения.

Задача 1 – Создать шаблон приложения Universal Windows

Начнём с создания проекта из шаблона пустого приложения.

1. В новой версии Visual Studio 2015 выберите **File (Файл) -> New (Новый) -> Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed (Установленное) > Templates (Шаблоны) > Visual C# > Windows > Universal**, а затем выберите шаблон **Blank App** приложения (Universal Windows).
2. Назовите свой проект **SpeechRecognition** и выберите местоположение файловой системы, в которой вы сохраните результаты прохождения практикума. На диске **C** создана папка под именем "**HOL**", информация о которой будет представлена в скриншотах во время прохождения всех практикумов.

Не изменяйте настройки, установленные для **Create new solution (Создания нового решения)** и **Create directory for solution (Создания папки для решения)**. Вы можете снять галочки как с **Add to source control (Добавить в систему контроля версий)**, так и с **Show telemetry in the Windows Dev Center (Отобразить телеметрию в Windows Dev Center)**, если не хотите использовать эти возможности. Нажмите **OK** для создания проекта.

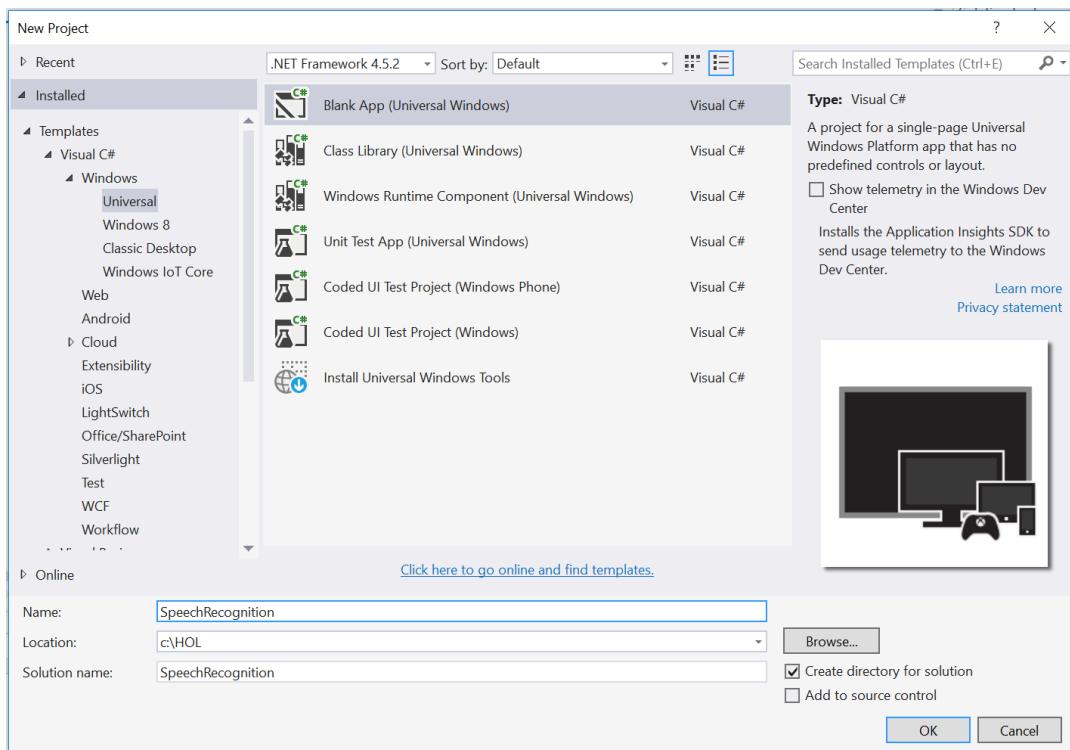


Рисунок 1

Создание нового проекта приложения в Visual Studio 2015.

3. Настройте Текущую конфигурацию решения на **Отладку** и Платформу решений в **x86**. Выберите **Локальный компьютер** из выпадающего меню Цели отладки.

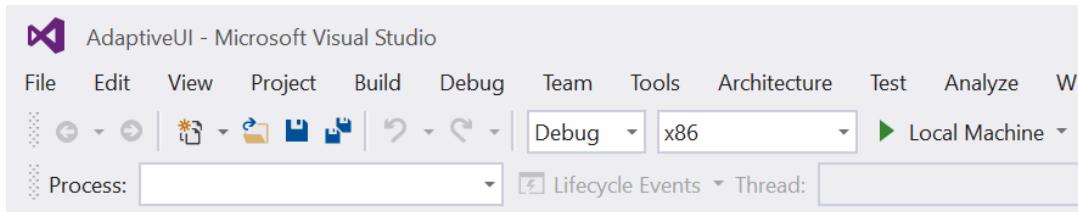


Рисунок 2

Сконфигурируйте свое приложение таким образом, чтобы оно запускалось на Локальном компьютере.

4. Скомпилируйте и запустите своё приложение. Вы увидите окно шаблона приложения со счетчиком частоты кадров, активированном по умолчанию для отладки.

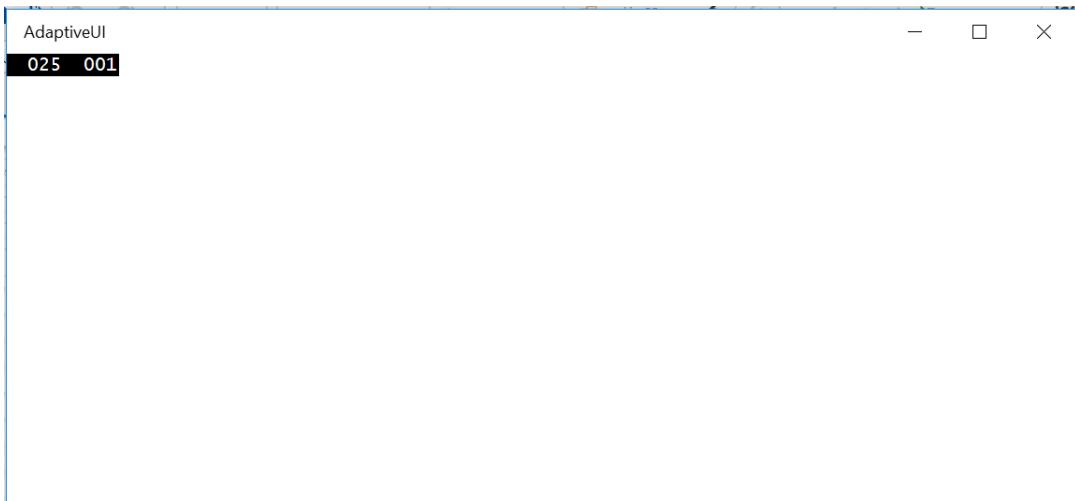


Рисунок 3

Пустое универсальное приложение, запущенное в режиме настольного компьютера.

Примечание: Счетчик частоты кадров является инструментом, используемым в процессе отладки, который помогает следить за производительностью вашего приложения. Он полезен для тех приложений, которые требуют интенсивной графической обработки, однако не удобен для простых приложений, которые будут создаваться вами в данном практическом курсе.

В шаблоне пустого приложения директива препроцессора для активации или отключения счетчика частоты кадров находится на **App.xaml.cs**. Счетчик частоты кадров может перекрывать или скрывать контент вашего приложения, если не убрать его. При выполнении данных работ вы можете отключить его, установив значение **DebugSettings.EnableFrameRateCounter** в **false**.

5. Вернитесь к Visual Studio и остановите отладку.

Задача 2 – Создайте файл определения голосовых команд

Схема голосовой команды определяется в XML-файле. В этой задаче вы создадите простую схему для голосового запуска приложения.

1. Щёлкните правой кнопкой мыши по наименованию проекта и выберите **Add (Добавить) > New Item (Новый элемент)**.
2. Выберите тип XML-файлов и присвойте ему имя **VoiceCommands.xml**.

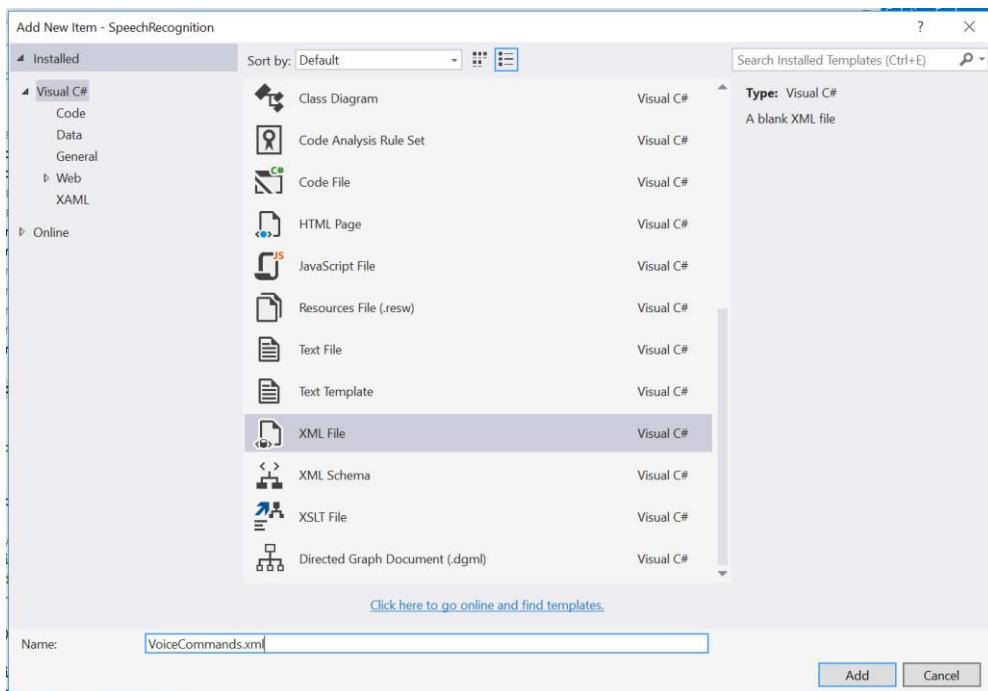


Рисунок 4

Создайте XML файл голосовых команд.

3. Откройте VoiceCommands.xml. После заголовка версии XML-файла добавьте элемент VoiceCommands и атрибут xmlns, ссылающийся на формат схемы голосовых команд . Добавьте элемент, описывающий набор команд, для языка EN-US (или другого используемого вами в работе поддерживаемого языка).

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.2">
    <CommandSet xml:lang="en-us" Name="HoLCommandSet_en-us">
    </CommandSet>
    <!--Опционально: второй набор команд для другого языка -->
    <!-- <CommandSet xml:lang="de-de" Name="HoLCommandSet_de-de"/> -->
</VoiceCommands>
```

Примечание: Мы добавили язык **EN-US** для этого примера, но вы можете добавить свой собственный набор команд на другом языке. Например, языковый тег для Германии был бы `xml:lang="de-de"`. Список регионов и языков, которые поддерживает Cortana, находится на <http://windows.microsoft.com/en-us/windows-10/cortanas-regions-and-languages>

Если вы решаете добавить другой набор команд на поддерживаемом языке, убедитесь в том, что каждый раз добавляя одну из EN-US команд, вы одновременно добавляете эквивалентную команду на этом языке.

4. Добавьте голосовой префикс для вашего приложения. Префикс – это слово или фраза, которую ваши пользователи должны сказать, чтобы указать ваше приложение в качестве получателя команд.

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.2">
    <CommandSet xml:lang="en-us" Name="HoLCommandSet_en-us">
        <CommandPrefix> Hands on Labs, </CommandPrefix>
    </CommandSet>
```

Примечание: Запятая после префикса команды необязательна. Если вы решите добавить её, она будет означать небольшую паузу между префиксом команды и самой командой. При распознавании команды Cortana также кратко отобразит префикс и команду именно так, как они записаны здесь.

Если вы решите добавить дополнительный набор голосовых команд на другом языке, добавьте префикс команды на этом языке в соответствующий набор команд.

5. Наша цель - запуск приложения с помощью голосовой команды. Создайте команду запуска с помощью элемента **ListenFor (Прослушать)**, который определяет слово или слова, которые будут распознаны для этой команды, с помощью элемента **Feedback (Обратная связь)**, который определяет текст подтверждения, проговариваемый Cortanой после пользователя при распознавании команды, и элемента **Navigate (Переход)**. Добавьте элемент **Example (Пример)** для новой команды к содержащему её **CommandSet (Набору команд)**.

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.2">
    <CommandSet xml:lang="en-us" Name="HoLCommandSet_en-us">
        <CommandPrefix> Hands on Labs, </CommandPrefix>
        <Example> Launch </Example>

        <Command Name="LaunchApp">
            <Example>launch</Example>
            <ListenFor>launch</ListenFor>
            <Feedback>Opening your speech recognition app</Feedback>
            <Navigate />
        </Command>
    </CommandSet>
</VoiceCommands>
```

Примечание: Элемент <Navigate/> означает, что в ответ на команду произойдет запуск приложения на переднем плане. Альтернативой запуска на переднем плане является определение компонента WinRT, который обрабатывает скрытое взаимодействие с данными приложения. Вы можете узнать больше об определениях голосовых команд по ссылке:

<https://msdn.microsoft.com/en-us/library/windows/apps/dn722331.aspx>

Задача 3 – Зарегистрируйте определения голосовых команд

Необходимо зарегистрировать определения голосовых команд (VCD) в переопределении функции **OnLaunched**. Не существует простого способа проверить, был ли ранее импортирован файл VCD, поэтому удобно каждый раз при запуске приложения регистрировать последнюю версию.

1. Откройте **App.xaml.cs**, добавьте ключевое слово **async** в переопределение **OnLaunched** и добавьте следующие строки:

C#

```
protected override async void OnLaunched(LaunchActivatedEventArgs e)
{
    //##if DEBUG
    //        if (System.Diagnostics.Debugger.IsAttached)
    //        {
    //            this.DebugSettings.EnableFrameRateCounter = true;
    //        }
    //##endif

    var storageFile = await
Windows.Storage.StorageFile.GetFileFromApplicationUriAsync(new Uri("ms-
appx:///VoiceCommands.xml"));

    await
Windows.ApplicationModel.VoiceCommands.VoiceCommandDefinitionManager.Instal
lCommandDefinitionsFromStorageFileAsync(storageFile);

    Frame rootFrame = Window.Current.Content as Frame;
```

Примечание: Определение голосовых команд, (VCD) будет установлено при первом запуске приложения. Необходимо первый раз открыть своё приложение из меню запуска для того, чтобы убедиться, что голосовые команды зарегистрированы.

Задача 4 – Активация по голосовой команде

Если ваше приложение запускается через голосовую команду, то его тип запуска будет **Activated (активировано)**. Соответственно, мы будем обрабатывать входящие голосовые команды в переопределении функции **OnActivated**. Необходимо при запуске проверить, было ли приложение запущено через голосовую команду или нет, и соответствующим образом скорректировать поведение приложения.

1. Создайте переопределение **OnActivated** в **App.xaml.cs**. Запуск голосовой команды соответствует вызову **OnActivated**, так что вы уже здесь сможете обработать входящую голосовую команду.

C#

```
protected override void OnActivated(IActivatedEventArgs args)
{
    base.OnActivated(args);
}
```

- Добавьте команду для обработки случая **ActivationKind.VoiceCommand**, и вызовите метод **HandleVoiceCommand**. Метод **HandleVoiceCommand** вы создадите на следующем этапе.

C#

```
protected override void OnActivated(IActivatedEventArgs args)
{
    switch (args.Kind)
    {
        case ActivationKind.VoiceCommand:
            HandleVoiceCommand(args);
            break;

        default:
            break;
    }
    base.OnActivated(args);
}
```

- Добавьте пространство имен **System.Diagnostics** к **App.xaml.cs**.

C#

```
using System.Diagnostics;
```

- Создайте метод **HandleVoiceCommand**. Этот метод обрабатывает входящую голосовую команду. Вы задали команду **LaunchApp** (**запуск приложения**) в своём файле определений голосовых команд в разделе 2.

C#

```
private void HandleVoiceCommand(IActivatedEventArgs args)
{
    var commandArgs = args as VoiceCommandActivatedEventArgs;
    var speechRecognitionResult = commandArgs.Result;
    var command = speechRecognitionResult.Text;

    var voiceCommandName = speechRecognitionResult.RulePath[0];
    var textSpoken = speechRecognitionResult.Text;

    Debug.WriteLine("Command: " + command);
    Debug.WriteLine("Text spoken: " + textSpoken);

    switch (voiceCommandName)
    {
        case "LaunchApp":
            break;
    }
}
```

```

        default:
            break;
    }
}

```

Примечание: Промежуточная печать команды через Debug.WriteLine будет полезна позднее при отладке кода. Вы можете добавить дополнительную печать, если хотите увидеть значения commandArgs и speechRecognitionResult. Далее в этой задаче мы рассмотрим некоторые приемы отладки приложений, запускаемых через голосовые команды.

5. Для успешного запуска приложения и перехода на страницу, нам нужно будет воспроизвести процедуру запуска приложения в функции OnLaunched. Шаблон приложения Visual Studio создаёт root frame и активирует окно, но в функции OnActivated эти действия не производятся. Добавьте соответствующий код для отображения окна в функцию OnActivated.

C#

```

protected override void OnActivated(IActivatedEventArgs args)
{
    Frame rootFrame = Window.Current.Content as Frame;

    if (rootFrame == null)
    {
        rootFrame = new Frame();
        rootFrame.NavigationFailed += OnNavigationFailed;
        Window.Current.Content = rootFrame;
    }

    switch (args.Kind)
    {
        case ActivationKind.VoiceCommand:
            HandleVoiceCommand(args);
            break;

        default:
            break;
    }

    Window.Current.Activate();

    base.OnActivated(args);
}

```

Примечание: Чтобы избежать дублирования в реальном приложении, обычно имеет смысл создать общий код запуска в отдельной функции, который выполнится для запущенных и активированных приложений. Template10 демонстрирует более унифицированный способ обработки этих важных действий при запуске приложения. Для ознакомления с более подробной информацией о Template10, посетите страницу <https://github.com/Windows-XAML/Template10>

6. Передайте ссылку на rootFrame методу **HandleVoiceCommand** в дополнение к **args**. Вам понадобится контекст корневого кадра для перехода на страницу.

C#

```
switch (args.Kind)
{
    регистр ActivationKind.VoiceCommand (тип активации. Голосовая команда):
        HandleVoiceCommand (аргументы, rootFrame);
        разрыв;

    по умолчанию:
        разрыв;
}
```

7. В методе **HandleVoiceCommand** добавьте параметр **frame** и используйте его для перехода на страницу **MainPage**:

C#

```
private void HandleVoiceCommand(IActivatedEventArgs args, Frame frame)
{
    var commandArgs = args as VoiceCommandActivatedEventArgs;
    var speechRecognitionResult = commandArgs.Result;
    var command = speechRecognitionResult.Text;
    var voiceCommandName = speechRecognitionResult.RulePath[0];
    var textSpoken = speechRecognitionResult.Text;
    switch (voiceCommandName)
    {
        case "LaunchApp":
            frame.Navigate(typeof(MainPage));
            break;
        default: break;
    }
}
```

8. Откройте **MainPage.xaml** и добавьте заголовок страницы. Текст поможет определить, что переход был совершен в момент запуска приложения.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <TextBlock Text="Speech Recognition and Voice Commands"
        FontWeight="Light" FontSize="20" Margin="12" />
</Grid>
```

9. Создайте и запустите своё приложение на локальном компьютере.

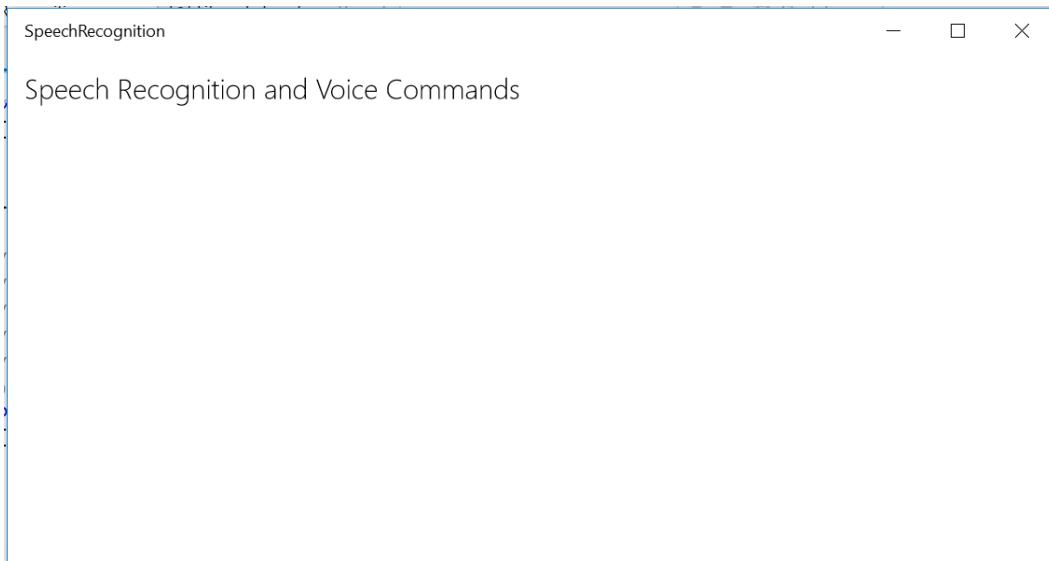


Рисунок 5

Приложение, выполняющееся на Локальном компьютере.

10. Закройте приложение. Откройте свойства проекта из обозревателя решений и перейдите во вкладку **Отладка**. Выберите действие при запуске, функцию **Не запускать, но произвести отладку моего кода при запуске**, и сохраните файл свойств. Вы можете использовать эту опцию, чтобы отладить приложение, которое вы не запускаете прямо из Visual Studio.
11. Для активации отладки без запуска приложения используйте кнопку **Начало отладки**.
12. Установите точку останова на **App.xaml.cs** после печати отладочных сообщений `Debug.WriteLine`.
13. Щёлкните по кнопке микрофона в своей панели задач для подготовки к запуску через голосовую команду.
14. Произнесите слова: " Hands-on Labs, launch". Cortana словами подтвердит, что открывает ваше приложение.

Примечание: Если ваши настройки региона, языка и распознавания речи будут установлены на другой поддерживаемый язык, для которого вы создали набор голосовых команд, вы можете использовать команды на другом языке. Если вы измените свои настройки региона, языка и речи, вам может понадобиться выйти из системы и снова зайти, чтобы изменение полностью вступило в силу.

15. Когда вы достигнете точки останова, откройте **окно вывода**, чтобы просмотреть сообщения об отладке. Вы увидите команду, которую распознала Cortana, а также фактически произнесенный текст. Они могут различаться в зависимости от того, насколько хорошо Cortana истолковала произнесенный вами текст. Используйте кнопку **Continue (Продолжить)** для возобновления запуска приложения из точки останова.

16. Ваше приложение запустится и перейдет к главной странице.

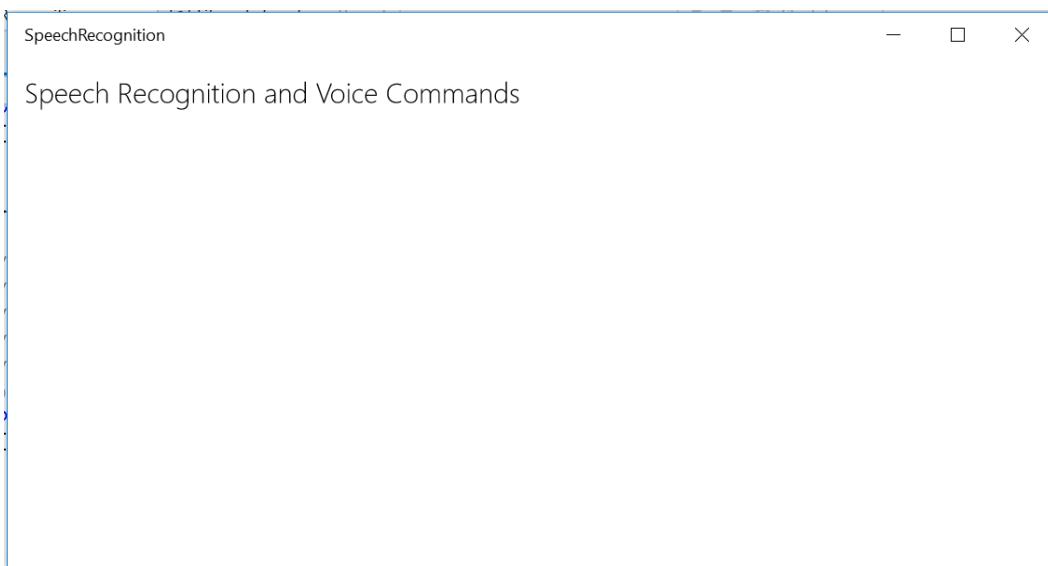


Рисунок 6

Приложение запускается через голосовую команду.

17. Закройте приложение и вернитесь в Visual Studio. Удалите точку останова из **App.xaml.cs**, и снимите галочку с опции **Do not launch, but debug my code when it starts** (**Не запускать, произвести отладку моего кода при активации**) в свойствах проекта. Сохраните файл.

Упражнение 2: Используйте голосовую команду, чтобы изменить вид приложения

В дополнение к запуску вашего приложения, голосовые команды могут взаимодействовать с содержимым приложения. В этом упражнении вы будете использовать голосовую команду для изменения фонового цвета приложения при его запуске.

Задача 1 – Установите цвет фона

В этой задаче вы зададите исходный цвет фона для приложения и атрибут **x:Name**, чтобы можно было обращаться к свойствам Grid.

1. Укажите **AliceBlue** как цвет фона для объекта **Grid** в **MainPage.xaml** и присвойте ему имя **Container**.

XAML

```
<Grid Background="AliceBlue" x:Name="Container" >
    <TextBlock Text="Speech Recognition and Voice Commands"
    FontWeight="Light" FontSize="20" Margin="12" />
</Grid>
```

2. Создайте и запустите своё приложение на Локальном компьютере. Вы увидите заголовок страницы на светло-синем фоне.



Рисунок 7

Работа приложения с цветным фоном.

3. Завершите отладку и вернитесь в Visual Studio.

Задача 2 – Создайте голосовую команду, чтобы вызвать изменение цвета

В этой задаче вы создадите голосовую команду для изменения фонового цвета сетки на красный.

1. Откройте свой файл определения VoiceCommands.xml и добавьте новую команду. Присвойте команде имя **TurnRed**.

XML

```
<Command Name="TurnRed">
    <Example>turn red</Example>
    <ListenFor>turn red</ListenFor>
    <Feedback>My favorite color is red</Feedback>
    <Navigate />
</Command>
```

Примечание: Если вы поддерживаете дополнительные языки, добавьте эквивалентную команду в соответствующий набор команд.

2. Добавьте случай для команды **TurnRed** в оператор switch, обрабатывающий **voiceCommandName** в файле **App.xaml.cs**. В этот раз при переходе на главную страницу MainPage передайте параметр.

C#

```
switch (voiceCommandName)
{
    case "LaunchApp":
        frame.Navigate(typeof(MainPage));
        break;
    case "TurnRed":
        frame.Navigate(typeof(MainPage), "Red");
        break;
    default:
        break;
}
```

3. Откройте код MainPage и создайте переопределение **OnNavigatedTo** для обработки входящего параметра. Задайте фон Grid в соответствии с входящем параметром.

C#

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    if (e.Parameter.ToString() == "Red")
        Container.Background = new
SolidColorBrush(Windows.UI.Colors.DarkRed);

    base.OnNavigatedTo(e);
}
```

4. Щёлкнитесь правой кнопкой мыши на проекте в обозревателе решений и выберите **Развёртывание**.

5. Создайте и запустите своё приложение на Локальном компьютере для регистрации новой голосовой команды. Вы должны всё еще видеть синий фон. Закройте своё приложение.
6. Щёлкните по кнопке микрофона Cortana и проговорите команду: " **Hand-on Labs, turn red** ". Cortana ответит, что красный – её любимый цвет. При следующем запуске приложения вы увидите, что цвет фона заменен на красный.



Рисунок 8

Приложение запускается через голосовую команду с красным фоном.

Примечание: Если вам нужно отладить код, который вы написали в этой задаче, используйте рассмотренный ранее метод с установкой опции **Do not launch, but debug my code when it starts** (**Не запускать, произвести отладку моего кода при активации**) из Упражнения 1: Задача 4.

7. Завершите отладку и вернитесь в Visual Studio.

Упражнение 3: Ответ на голосовую команду при помощи фоновой задачи

Голосовые команды могут активировать работу фоновых задач без запуска вашего приложения. Такая схема может быть полезной, когда вы хотите разрешить своим пользователям выполнять простые задачи, связанные с вашим приложением, непосредственно через Cortana без запуска приложения. В этом упражнении вы создадите компонент для ответа на вопрос пользователя через фоновую задачу.

Задача 1 – Создайте компонент времени выполнения Windows

Мы начнём с создания компонента времени выполнения Windows для фоновой задачи:

1. Щёлкните правой кнопкой мыши на названии решения в Обозревателе решений. Выберите **Добавить -> Новый Проект** и выберите тип проекта **компонент времени выполнения Windows** (Windows Универсальная).
2. Назовите проект **VoiceCommandService**.

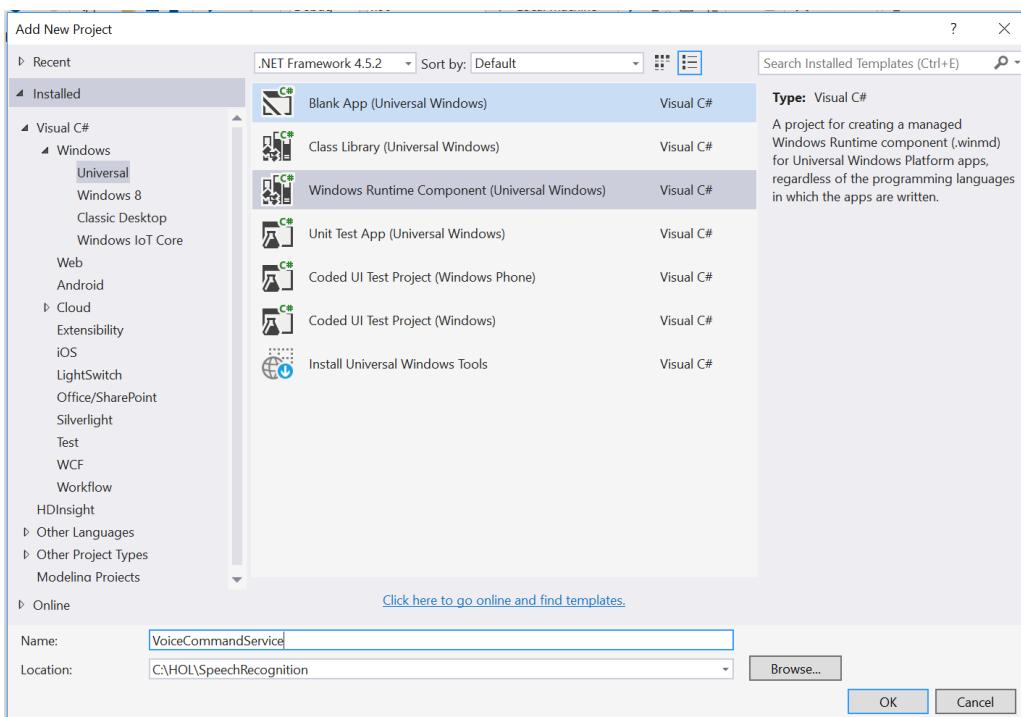


Рисунок 9

Добавьте Компонент Времени выполнения Windows.

3. Щёлкните правой кнопкой мыши по Class1.cs в Обозревателе решений и переименуйте его в **HolVoiceCommandService**. Если появится опция сменить имя проекта по всем ссылкам, выберите "Да".

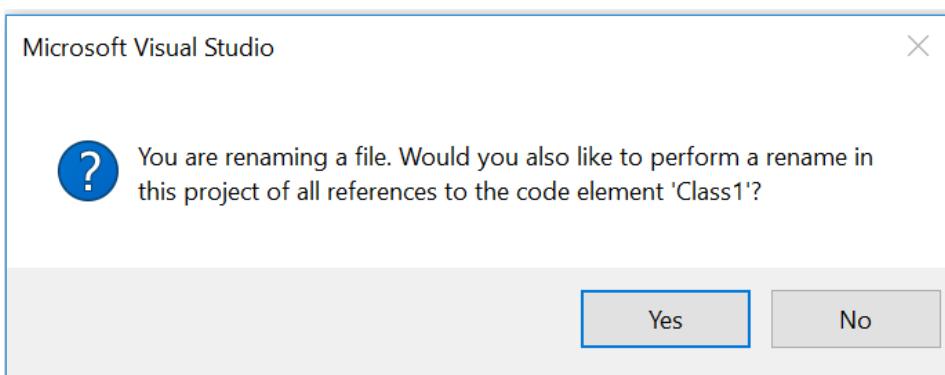


Рисунок 10

Переименуйте Class1.cs в HolVoiceCommandService.cs.

4. Вернитесь к проекту Распознавание речи. Щёлкните правой кнопкой мыши по разделу References (ссылки) и добавьте VoiceCommandService к списку ссылок.

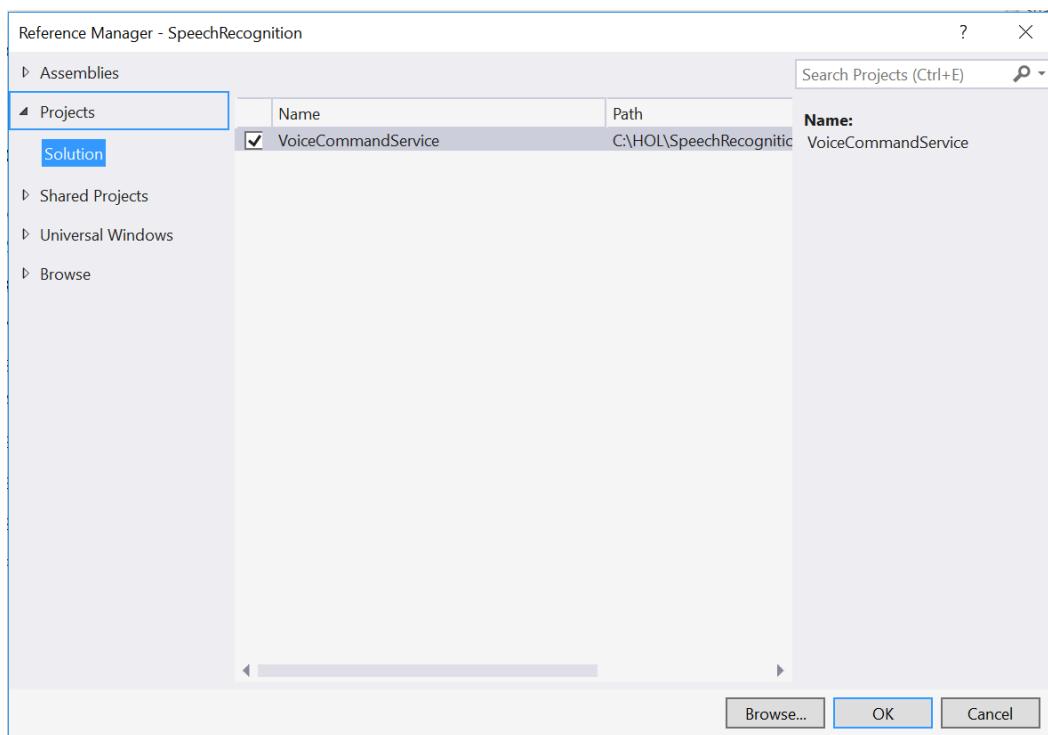


Рисунок 11

Ссылка на *VoiceCommandService* из проекта *SpeechRecognition*.

Задача 2 – Добавьте голосовую команду, ссылающуюся на VoiceCommandService

Голосовые команды, которые запускают фоновые задачи, отличаются от голосовых команд, которые вы создавали раньше. Вместо того, чтобы использовать элемент Navigate, команда задает пользовательский элемент VoiceCommandService , имеющий целевой атрибут с указанием на класс HolVoiceCommandService . В этой задаче вы создадите голосовую команду, а также предоставите ей дополнительные речевые опции. Вы можете добавить речевые опции в любые голосовые команды, независимо от того, используются ли они для запуска приложения на переднем плане или фоновой задачи.

1. В своём проекте SpeechRecognition добавьте к **VoiceCommands.xml** команду **SayHello**. Вы можете определять в команде многочисленные **<ListenFor>** элементы. В этом случае определите для Cortana два ключевых слова, оба из которых будут служить для запуска команды.

Примечание: Если вы поддерживаете дополнительные языки, добавьте эквивалентную команду в соответствующий набор команд.

XML

```
<Command Name="SayHello">
    <Example>say hello</Example>
    <ListenFor RequireAppName="BeforeOrAfterPhrase">How's it
going</ListenFor>
    <ListenFor RequireAppName="BeforeOrAfterPhrase">Say hello</ListenFor>
    <Feedback>Hold on, let me ask</Feedback>
    <VoiceCommandService Target="HolVoiceCommandService" />
</Command>
```

Примечание: Атрибут **RequireAppName="BeforeOrAfterPhrase"** придает гибкость и естественность словесному выражению ваших голосовых команд. Обе фразы "Лабораторный практикум, как дела?" И "Как дела, лабораторный практикум?" допустимы, если этот атрибут установлен в BeforeOrAfterPhrase. Для получения более подробной информации об опциях словесного выражения зайдите в документацию на ListenFor по ссылке <https://msdn.microsoft.com/en-us/library/windows/apps/dn706593.aspx>

Задача 3 – Зарегистрируйте сервис в манифесте приложения

Для фонового запуска HolVoiceCommandService, он должен быть зарегистрирован в манифесте приложения SpeechRecognition.

1. Откройте Package.appxmanifest в редакторе манифестов и перейдите на вкладку Declarations.
2. Используя выпадающее меню **Available Declarations**, выберите **App Service (Служба приложений)** и кликните **Добавить**, чтобы добавить службу в список поддерживаемых описаний.

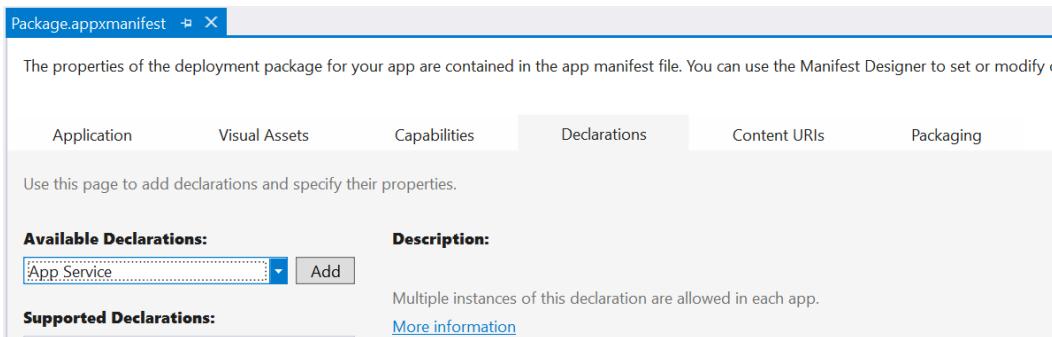


Рисунок 12

Добавьте Службу приложений в список поддерживаемых описаний.

3. В Свойствах манифеста сервисов приложений, установите **Name = HolVoiceCommandService**.
4. Установите свойство точки входа (Entry Point) в **VoiceCommandService.HolVoiceCommandService**.

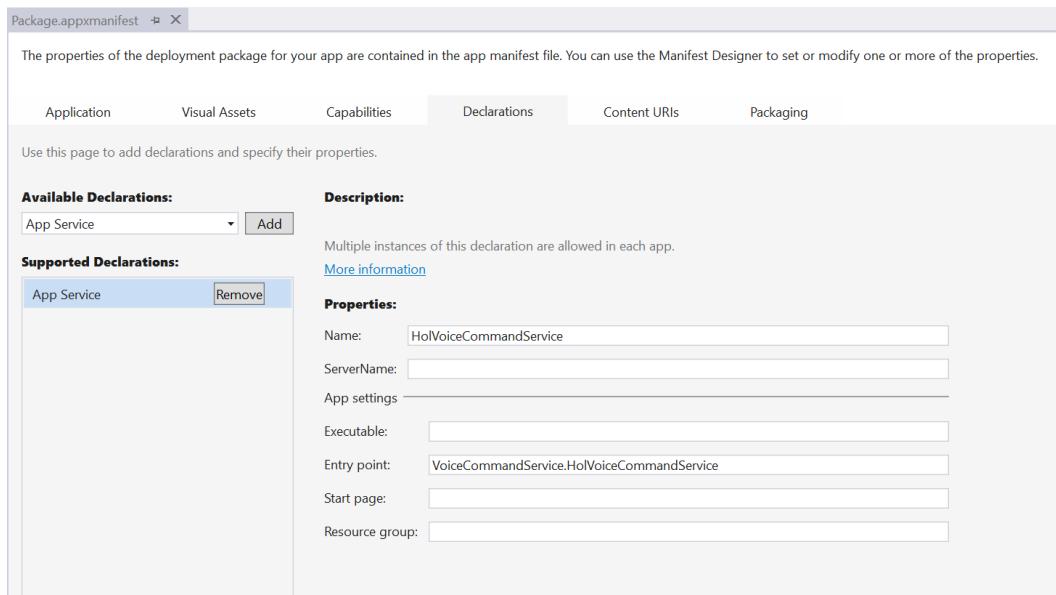


Рисунок 13

Зарегистрируйте Службу приложений в манифесте приложений.

Примечание: При регистрации службы приложений, свойство Name должно соответствовать имени класса в рамках вашего компонента, а не имени самого компонента. Мы дали классу HolVoiceCommandService имя, отличное от компонента WinRT, чтобы их было легче различать.

Это различие также важно при установке атрибута Target в определении голосовых команд, которое вы делали в задаче 2.

5. Закройте манифест, сохранив изменения.

Задача 4 – Обработайте входящую команду в сервисе

В этой задаче необходимо реализовать точку входа для всех скрытых (не приводящих к запуску приложения) голосовых команд, запущенных через Cortana. Фоновая задача, которую вы создадите, должна в течение 0.5 секунды предоставить ответ Cortana, и каждые 5 секунд должна сообщать о прогрессе – в противном случае она будет принудительно закрыта.

1. Откройте **HolVoiceCommandService.cs** и добавьте пространство имен Windows.ApplicationModel.Background.

C#

```
using Windows.ApplicationModel.Background;
```

2. Модифицируйте описание класса так, чтобы он реализовывал интерфейс **IBackgroundTask**. Добавьте определенный в этом интерфейсе метод **Run**

C#

```
public sealed class HolVoiceCommandService : IBackgroundTask
{
    public void Run(IBackgroundTaskInstance taskInstance)
    {
        throw new NotImplementedException();
    }
}
```

3. Объявите элемент класса **serviceDeferral** типа **BackgroundTaskDeferral**. В своём методе **Run** получите deferral для своего экземпляра задачи и сохраните его в **serviceDeferral**. Добавьте методы **OnVoiceCommandCompleted()** и **OnTaskCanceled()** для обработки завершения. На более позднем этапе вы оформите подписку на события **VoiceCommandCompleted**.

C#

```
public sealed class HolVoiceCommandService : IBackgroundTask
{
    BackgroundTaskDeferral serviceDeferral;

    public void Run(IBackgroundTaskInstance taskInstance)
    {
        serviceDeferral = taskInstance.GetDeferral();

        taskInstance.Canceled += OnTaskCanceled;
    }
}

private void OnVoiceCommandCompleted(VoiceCommandServiceConnection
sender, VoiceCommandCompletedEventArgs args)
{
    if (this.serviceDeferral != null)
    {
        this.serviceDeferral.Complete();
    }
}
```

```

        }

        private void OnTaskCanceled(IBackgroundTaskInstance sender,
BackgroundTaskCancellationReason reason)
{
    System.Diagnostics.Debug.WriteLine("Задача отменена, очистить");
    if (this.serviceDeferral != null)
    {
        //Завершить отсрочку службы
        this.serviceDeferral.Complete();
    }
}

```

- Добавить пространства имен **Windows.ApplicationModel.AppService** и **Windows.ApplicationModel.VoiceCommands**.

C#

```

using Windows.ApplicationModel.AppService
using Windows.ApplicationModel.VoiceCommands;

```

- Добавьте ссылку на подключение к службе голосовых команд.

C#

```

public sealed class HolVoiceCommandService : IBackgroundTask
{
    VoiceCommandServiceConnection voiceServiceConnection;

    BackgroundTaskDeferral serviceDeferral;
}

```

Примечание: Подключение к службе сохраняется в течение всего периода работы с Cortana.

- Проверьте, соответствует ли имя указанному при регистрации службы приложений из манифеста приложений. Если да, то установите соединение со службой Cortana, используя блок try-catch для обнаружения проблем.

C#

```

taskInstance.Canceled += OnTaskCanceled;

var triggerDetails = taskInstance.TriggerDetails as
AppServiceTriggerDetails;

if (triggerDetails != null && triggerDetails.Name ==
"HolVoiceCommandService")
{
    try
    {
        voiceServiceConnection =
VoiceCommandServiceConnection.FromAppServiceTriggerDetails(
            triggerDetails);
    }
}

```

```

        voiceServiceConnection.VoiceCommandCompleted +=  

        OnVoiceCommandCompleted;  

    }  

    catch (Exception ex) {  

        System.Diagnostics.Debug.WriteLine ("Обработка голосовой команды  

потерпела неудачу" + ex.ToString());  

    }  

}

```

Примечание: Подписка на событие VoiceCommandCompleted находится именно в этом фрагменте кода, поскольку оно должно произойти после установки voiceServiceConnection.

Если вам нужно отладить этот код, используйте метод, описанный выше, и установите в вашем методе run точку останова.

7. Используйте await для ожидания входящей голосовой команды и создайте переключатель для обработки команды **SayHello**. Добавьте ключевое слово **async** к заголовку метода Run.

C#

```

try
{
    voiceServiceConnection =
        VoiceCommandServiceConnection.FromAppServiceTriggerDetails(
            triggerDetails);

    voiceServiceConnection.VoiceCommandCompleted +=  

    OnVoiceCommandCompleted;

    VoiceCommand voiceCommand = await
        voiceServiceConnection.GetVoiceCommandAsync();

    switch (voiceCommand.CommandName)
    {
        case "SayHello":
            break;
        default:
            break;
    }
}

```

8. Если входящая голосовая команда будет **SayHello**, создайте ответное сообщение и задайте дисплейное и речевое сообщения, которые Cortana передаст обратно пользователю.

C#

```

switch (voiceCommand.CommandName)
{
    case "SayHello":
        var userMessage = new VoiceCommandUserMessage();

```

```
        userMessage.DisplayMessage = "Hello!";
        userMessage.SpokenMessage = "Your app says hi. It is having a great
time.";
        var response = VoiceCommandResponse.CreateResponse(userMessage);
        await voiceServiceConnection.ReportSuccessAsync(response);
        break;
    default:
        break;
}
```

9. Сохраните HolVoiceCommandService.
-

Задача 5 – Начните работу со своим приложением через фоновую задачу

Теперь, когда вы настроили голосовую команду, фоновую задачу и зарегистрировали службу приложений, самое время попробовать, как это работает.

1. Запустите проект SpeechRecognition на Локальном компьютере, чтобы зарегистрировать самую последнюю версию файла определения голосовых команд.
2. Закройте приложение.
3. Используйте интерфейс Cortana для того, чтобы проговорить один из вариантов голосовой команды SayHello:
 - “Hands-on labs, how’s it going?”
 - “How’s it going, Hands-on labs?”
 - “Hands-on labs, say hello.”
 - “Say hello, Hands-on labs.”
4. Cortana покажет изображение вашего приложения и дисплейное сообщение. Одновременно она проговорит речевое сообщение, которое вы выбрали в фоновой задаче.

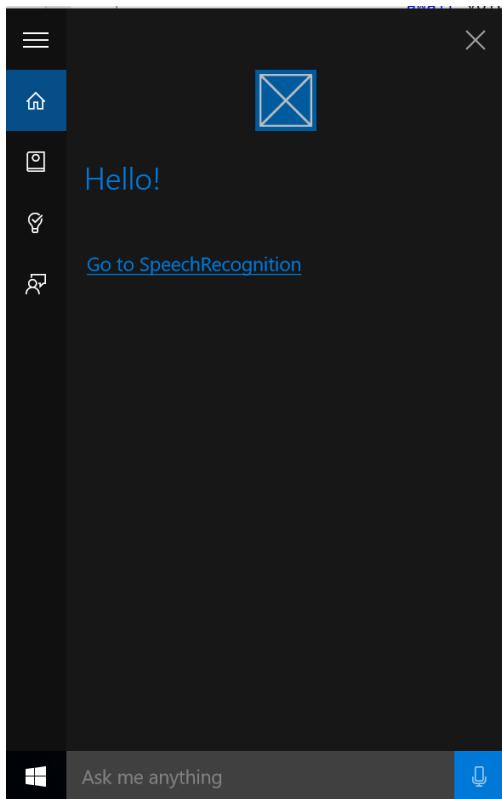


Рисунок 14

Cortana возвращает сообщение из фоновой задачи приложения.

Примечание: Чтобы отладить свою фоновую задачу, используйте метод аналогичный тому, который вы использовали для отладки своих голосовых команд. Щёлкните правой кнопкой мыши на проекте SpeechRecognition в Обозревателе решений и откройте редактор **Свойств**. На вкладке **Отладка** отметьте опцию **Не запускать, произвести отладку моего кода при активации**. Сохраните файл свойств и используйте кнопку пуска отладки для активации отладки. Установите точку останова в HolVoiceCommandService и запустите приложение, используя команду SayHello.

5. Остановите отладку и вернитесь в Visual Studio.

Краткий обзор

Голосовые команды – это важная часть взаимодействия Windows 10 с пользователем. В этой работе вы создали файл определения голосовых команд и исследовали структуру VCD. Вы добавили команды запуска приложения, взаимодействия с ним и получения ответа из фоновой задачи. Вы также научились регистрировать и различать голосовые команды.



Лабораторный практикум

Использование рукописного ввода

Октябрь 2015 года



Обзор

Рукописный ввод является мощным средством преобразования естественного движения в пиксели. Поскольку все больше устройств поддерживает сенсорные экраны, то рукописный ввод играет всё большую роль во взаимодействии с пользователями. Традиционно стилус является основным устройством ввода чернил. В UWP, использование стилуса, касаний и мыши может быть добавлено всего несколькими строчками кода. Элемент InkCanvas упростит работу с чернилами в вашем приложении.

В этом курсе вы создадите InkCanvas, принимающий ввод данных через мышку, прикасновения и стилус. Вы научитесь моментально настраивать цвет чернил и стирать, очищать, сохранять и загружать штрихи. В Упражнении 2 вы осуществите распознавание рукописных текстов на InkCanvas.

Цели

Пройдя эту работу, вы сможете:

- Создавать InkCanvas в приложении UWP
 - Применять для ввода данных стилус, прикасновение и мышь
 - Изменять цвет чернил по выбору пользователя
 - Стирайте одиночные штрихи
 - Очищать холст
 - Сохранять и загружать штрихи
 - Распознавать рукописный текст
-

Системные требования

Чтобы выполнить этот курс, необходимо обладать следующим набором программных инструментов:

- Microsoft Windows 10
 - Microsoft Visual Studio 2015
-

Настройка

Вам следует выполнить следующие действия для подготовки компьютера:

1. Установить Microsoft Windows 10.
 2. Установить Microsoft Visual Studio 2015.
-

Упражнения

Эта лабораторная работа включает следующие упражнения:

1. Рисование с помощью InkCanvas
 2. Распознавание рукописных текстов
-

Расчетное время для завершения практикума: **45-60 минут.**

Упражнение 1: Рисование с помощью InkCanvas

InkCanvas является отличным способом быстро применения чернил в вашем UWP приложении. В этом упражнении вы создадите InkCanvas и модифицируете его графические атрибуты для создания штрихов разных цветов. Вы также сможете сохранить, загрузить и стереть штрихи и очистить холст.

Задача 1 – Создайте шаблон приложения Universal Windows

Начнём с создания проекта из шаблона пустого приложения.

1. В новой версии Visual Studio 2015 выберите **File (Файл) -> New (Новый) -> Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed (Установленное) > Templates (Шаблоны) > Visual C# > Windows > Universal**, а затем выберите шаблон **Blank App** приложения (Universal Windows).
2. Назовите свой проект **Inking** и выберите местоположение файловой системы, в котором будет осуществлено хранение результатов прохождения лабораторного курса. Мы создали папку на диске **C** и назвали её **HOL**, именно папку с этим названием вы будете видеть на скриншотах в ходе всего практического курса.

Не изменяйте настройки, установленные для **Create new solution (Создания нового решения)** и **Create directory for solution (Создания папки для решения)**. Вы можете

снять галочки как с **Add to source control (Добавить в систему контроля версий)**, так и с **Show telemetry in the Windows Dev Center (Отобразить телеметрию в Windows Dev Center)**, если не хотите использовать эти возможности. Нажмите **OK** для создания проекта.

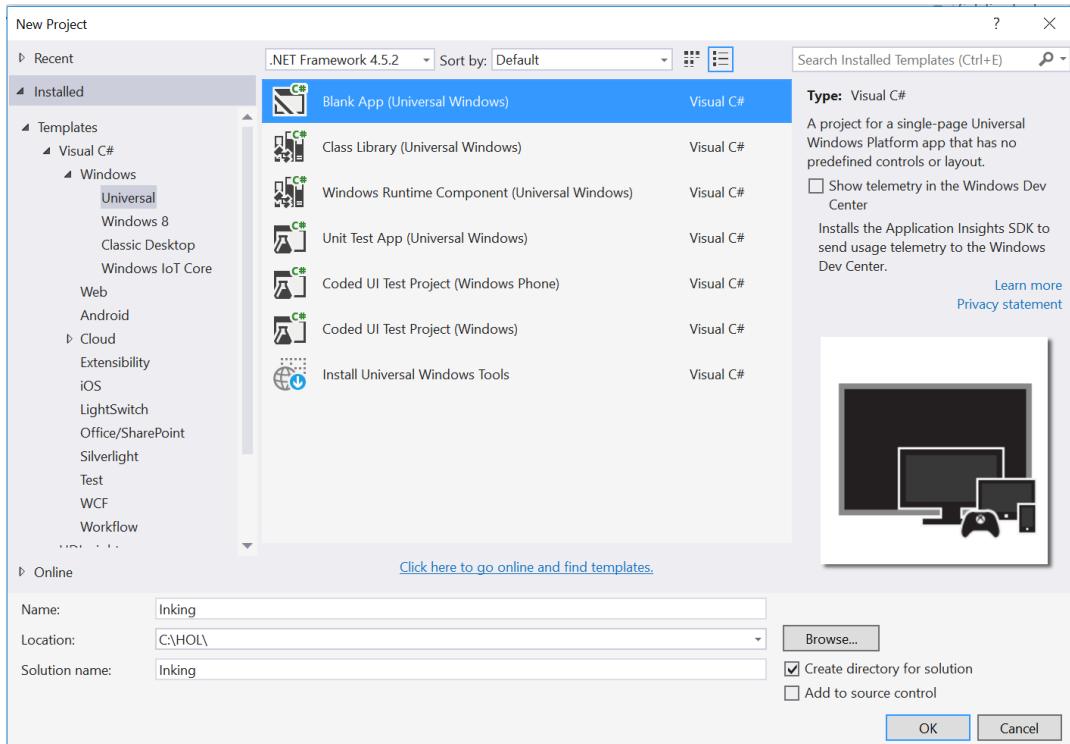


Рисунок 1

Создание нового проекта приложения в Visual Studio 2015.

- Настройте Текущую конфигурацию решения на **отладку** и платформу решений на **x86**. Выберите **Локальный компьютер** из выпадающего меню **цели отладки**.

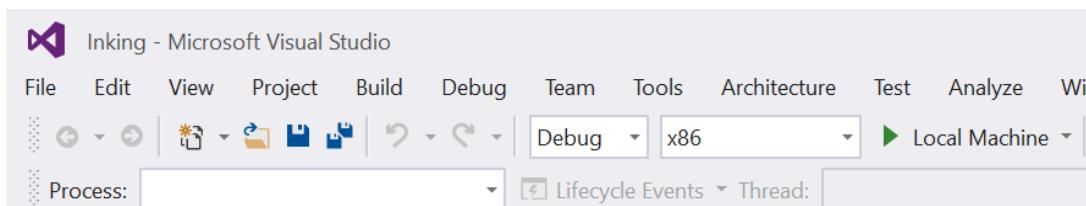


Рисунок 2

*Сконфигурируйте свое приложение таким образом, чтобы оно запускалось на **Локальном компьютере**.*

- Скомпилируйте и запустите своё приложение. Вы увидите окно шаблона приложения со счетчиком частоты кадров, активированном по умолчанию для отладки.

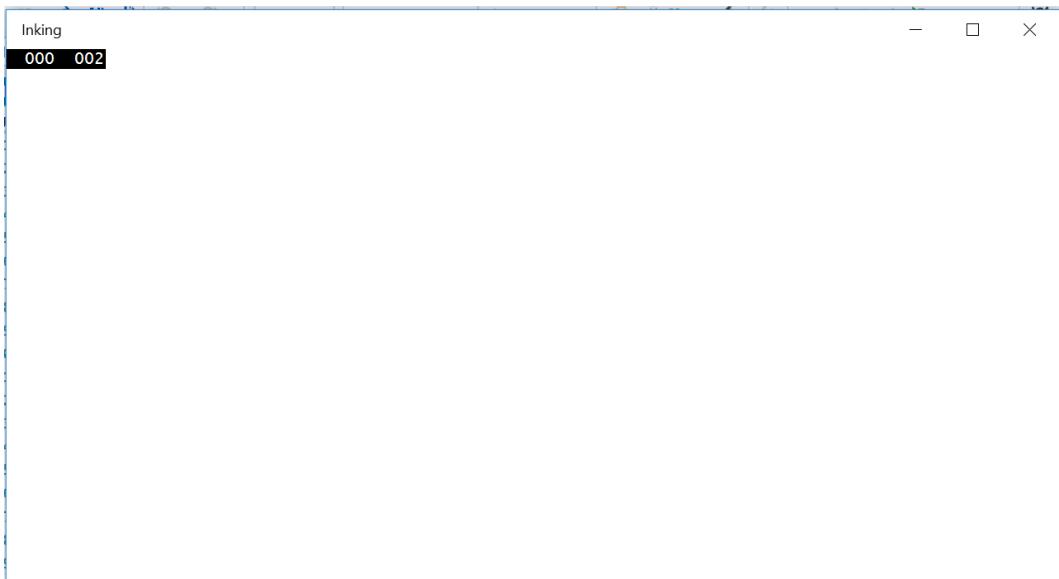


Рисунок 3

Пустое универсальное приложение, запущенное в режиме настольного компьютера.

Примечание: Счетчик частоты кадров является инструментом, используемым в процессе отладки, который помогает следить за производительностью вашего приложения. Он полезен для тех приложений, которые требуют интенсивной графической обработки, однако не подходит для простых приложений, которые будут создаваться вами в данном практическом курсе.

В шаблоне пустого приложения директива препроцессора для активации или отключения счетчика частоты кадров находится на **App.xaml.cs**. Счётчик скорости кадра может перекрывать или скрывать содержание вашего приложения, если вы оставляете его включённым. Для целей настоящего курса вы можете отключить его, установив значение **DebugSettings.EnableFrameRateCounter** как **false**.

5. Вернитесь в Visual Studio и остановите отладку.

Задача 2 – Создайте InkCanvas

Вы можете начать использование чернил в своём приложении используя всего несколько строчек кода. В этой задаче мы будем использовать большое полотно чернил и две строки вспомогательных элементов управления выше и ниже его.

1. Откройте **MainPage.xaml**. Мы настроили фон **Grid** на **LightGray (Светло-серый)** и добавили **RowDefinitions (Определения строк)** для трёх строк. Верхние и нижние строки будут содержать опции и кнопки стиля, в большой центральной строке будет находиться **InkCanvas**.

XAML

```
<Grid Background="LightGray">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
```

```

        <RowDefinition Height="2*" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
</Grid>
```

- Добавьте разметку для этих трех строк. Первая строка будет содержать **Grid** шириной в два столбца, в каждом из которых будет горизонтальная StackPanel. Вторая строка будет содержать **InkCanvas**, третья строка будет состоять еще из одной StackPanel. Назовите свой InkCanvas атрибутом **x:Name**.

XAML

```

<Grid Background="LightGray">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="2*" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
    <Grid Grid.Row="0" Margin="12">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="1*" />
            <ColumnDefinition Width="1*" />
        </Grid.ColumnDefinitions>
        <StackPanel Grid.Column="0" Orientation="Horizontal">
        </StackPanel>
        <StackPanel Grid.Column="1" Orientation="Horizontal"
HorizontalAlignment="Right">
        </StackPanel>
    </Grid>
    <Grid Grid.Row="1" Background="White" Grid.ColumnSpan="2">
        <InkCanvas x:Name="InkCanvas" />
    </Grid>
    <StackPanel Grid.Row="2" Orientation="Horizontal" Margin="12">
    </StackPanel>
</Grid>
```

- На этом этапе ваше приложение готово к простому чернильному вводу с помощью стилуса. Если ваш компьютер не поддерживает стилус, мы научимся использовать другие устройства ввода ниже.
- Во вспомогательном коде MainPage добавьте пространство имен **Windows.UI.Input.Inking**.

Примечание: InkCanvas по умолчанию принимает только ввод стилусом. Вы можете использовать мышь или прикосновения в качестве устройств дополнительного ввода.

- Добавить **мышь и прикосновения**, как типы устройств ввода для **InkCanvas**.

C#

```

public MainPage()
{
    this.InitializeComponent();
```

```
InkCanvas.InkPresenter.InputDeviceTypes =  
    Windows.UI.Core.CoreInputDeviceTypes.Mouse |  
    Windows.UI.Core.CoreInputDeviceTypes.Pen |  
    Windows.UI.Core.CoreInputDeviceTypes.Touch;  
}
```

Примечание: InkCanvas содержит одно свойство обведения чернилами – InkPresenter. InkPresenter отображает чернила на полотне и позволяет устанавливать атрибуты для нанесения штрихов.

6. Скомпилируйте и запустите приложение. Сделайте рисунок на белом **InkCanvas** с помощью мышки, стилуса или прикосновений. Вы увидите цвет чернил по умолчанию и атрибуты размеров пера.

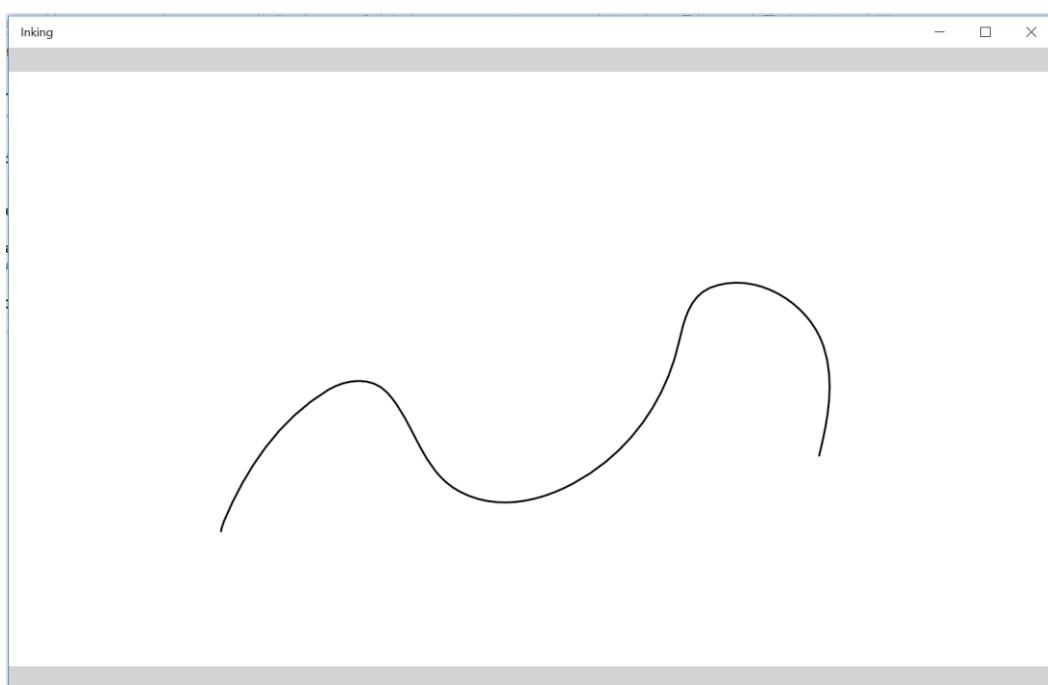


Рисунок 4

Графические атрибуты по умолчанию для InkCanvas.

7. Завершите отладку и вернитесь в Visual Studio.

Задача 3 – Настройте графические атрибуты

Вы можете настраивать графические атрибуты для **InkCanvas** с помощью экземпляра **InkDrawingAttributes**. Вы также можете по выбору пользователей программным путём обновлять атрибуты. В этой задаче вы сначала добавите атрибуты по умолчанию в стиль рукописного фрагмента, а затем кнопки, чтобы изменять атрибуты в процессе ввода.

1. Вы можете установить атрибуты сразу при создании экземпляра **InkDrawingAttributes**. Добавьте атрибуты **цвета, размера, надавливания и подогонки под кривую** в конструктор **MainPage**. После установки обновите графические атрибуты **InkCanvas**.

C#

```
public MainPage()
{
    this.InitializeComponent();

    InkDrawingAttributes drawingAttributes = new InkDrawingAttributes();

    drawingAttributes.Color = Windows.UI.Colors.Red;
    drawingAttributes.Size = new Size(4, 4);
    drawingAttributes.IgnorePressure = false;
    drawingAttributes.FitToCurve = true;

    InkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(drawingAttributes);

    InkCanvas.InkPresenter.InputDeviceTypes =
        Windows.UI.Core.CoreInputDeviceTypes.Mouse |
        Windows.UI.Core.CoreInputDeviceTypes.Pen |
        Windows.UI.Core.CoreInputDeviceTypes.Touch;
}
```

Примечание: Хотя IgnorePressure является опцией графических атрибутов чертежа, **InkCanvas** на данный момент не отображает чувствительные к надавливанию штрихи. Вместо этого значение надавливания фиксируется в каждой точке.

Другие графические атрибуты включают PenTip и DrawAsHighlighter. Больше информации [по классам графических атрибутов](#) для рисования вы можете найти по ссылке <https://msdn.microsoft.com/en-us/library/windows.ui.input.inking.inkdrawingattributes.aspx>

2. Скомпилируйте и запустите приложение. В этот раз ваш рукописный стиль отобразит более широкий штрих красного цвета.

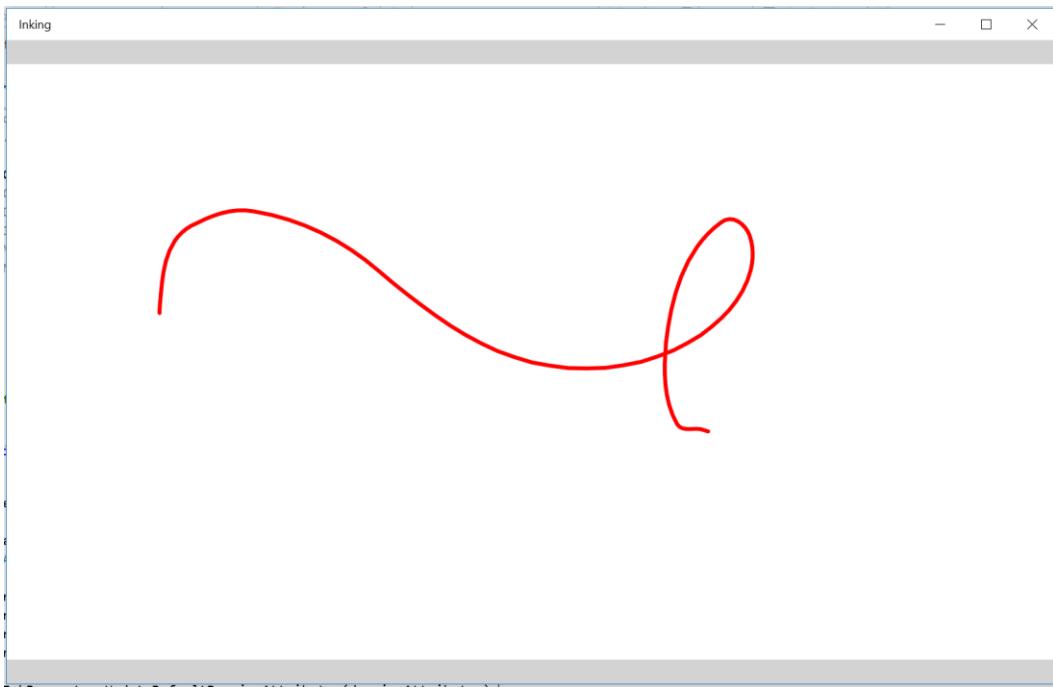


Рисунок 5

Более широкий рукописный штрих красного цвета.

3. Завершите отладку и вернитесь в Visual Studio.
4. Добавьте набор кнопок выбора цвета в StackPanel в первом ряду сетки MainPage. Установите событие Click для каждого **OnPenColorChanged** и стиль для **ColorButtonStyle StaticResource**. В одном из следующих этапов вы создадите обработчик события и ColorButtonStyle.

XAML

```
<StackPanel Grid.Column="0" Orientation="Horizontal">
    <TextBlock Text="Color" Margin="0,0,12,0" VerticalAlignment="Center" />
    <Button Background="Crimson" Foreground="Crimson"
Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}"
Margin="0,0,3,0"/>
    <Button Background="Orange" Foreground="Orange"
Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}"
Margin="0,0,3,0"/>
    <Button Background="Gold" Foreground="Gold"
Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}"
Margin="0,0,3,0"/>
    <Button Background="LimeGreen" Foreground="LimeGreen"
Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}"
Margin="0,0,3,0"/>
    <Button Background="DeepSkyBlue" Foreground="DeepSkyBlue"
Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}"
Margin="0,0,3,0"/>
    <Button Background="MediumOrchid" Foreground="MediumOrchid"
Click="OnPenColorChanged" Style="{StaticResource ColorButtonStyle}"
Margin="0,0,3,0"/>
```

```
<Button Background="Black" Foreground="Black" Click="OnPenColorChanged"
Style="{StaticResource ColorButtonStyle}" Margin="0,0,3,0"/>
</StackPanel>
```

- Добавьте стили в элемент **Grid.Resources** для установки высоты, ширины, полей, размера шрифта и заполнения для кнопок.

XAML

```
<Grid Background="LightGray">
    <Grid.Resources>
        <Style x:Key="ColorButtonStyle" TargetType="Button">
            <Setter Property="MinWidth" Value="28"/>
            <Setter Property="MinHeight" Value="28"/>
            <Setter Property="Width" Value="28"/>
            <Setter Property="Height" Value="28"/>
            <Setter Property="Margin" Value="0,0,0,0"/>
            <Setter Property="FontSize" Value="0"/>
            <Setter Property="Padding" Value="0,0,0,0"/>
        </Style>
    </Grid.Resources>
```

- В выделенном коде добавьте обработчик событий **OnPenColorChanged**. Этот метод копирует графические атрибуты по умолчанию из InkCanvas и устанавливает цвет чернил в соответствии с фоновым цветом кнопки.

C#

```
private void OnPenColorChanged(object sender, RoutedEventArgs e)
{
    if (InkCanvas != null)
    {
        InkDrawingAttributes drawingAttributes =
InkCanvas.InkPresenter.CopyDefaultDrawingAttributes();

        // Используйте фон кнопки, чтобы установить цвет нового пера
        var btnSender = sender as Button;
        var brush = btnSender.Background as SolidColorBrush;

        drawingAttributes.Color = brush.Color;
        InkCanvas.InkPresenter
        .UpdateDefaultDrawingAttributes(drawingAttributes);
    }
}
```

- Скомпилируйте и запустите приложение. Используйте кнопки выбора цвета для изменения цвета чернил во время рисования.

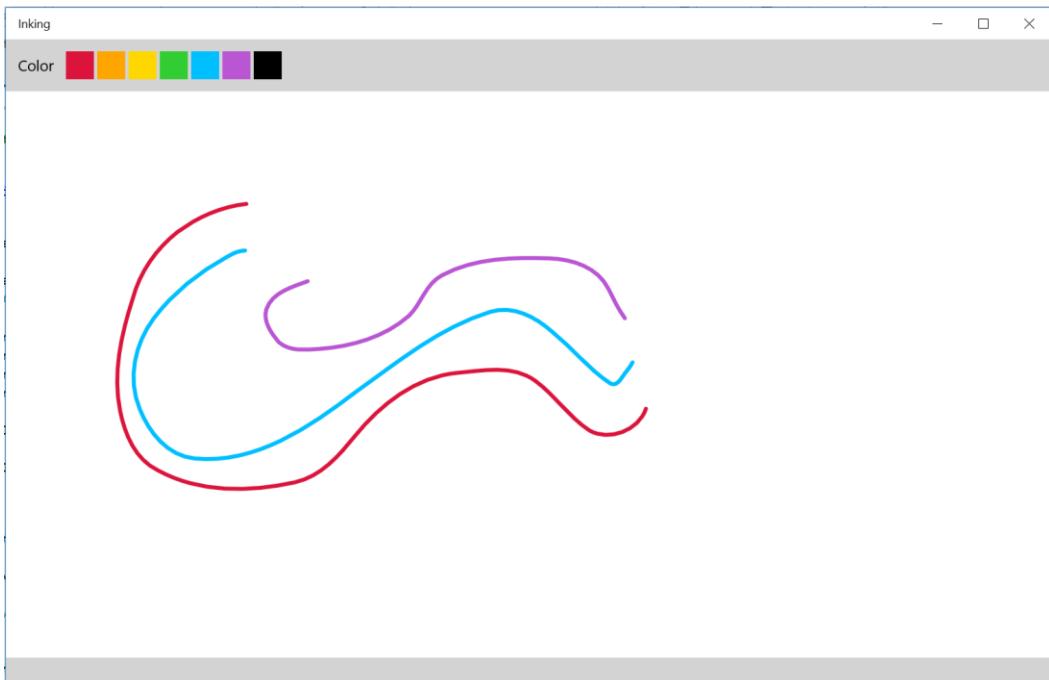


Рисунок 6

Замените цвет чернил на выбранные пользователем цвета.

8. Завершите отладку и вернитесь в Visual Studio.

Примечание: UI InkToolbar, используемый в Edge Browser Web Notes, доступен в виде расширения Visual Studio. Панель инструментов содержит в готовом виде палитру, выбор наконечников пера и режимы чернил, маркеров и ластиков. Дополнительную информацию по панели инструментов рукописного стиля см. по ссылке

<https://visualstudiogallery.msdn.microsoft.com/58194dfe-df44-4c4e-893a-1eca40675269>

Задача 4 – Стереть и очистить

Сейчас, когда вы умеете настраивать штрихи, поговорим про стирание нарисованного. Вы можете стирать одиночные штрихи или очищать весь холст сразу.

1. В **MainPage.xaml** добавьте чекбокс внутрь **StackPanel**, который находится во втором столбце основной сетки. Отмеченная графа появится справа от кнопок выбора цвета, чтобы переключить режим стирания. Настройте события Checked и Unchecked на **ErasingModeCheckBox_Checked** и **ErasingModeCheckBox_Unchecked** соответственно. На следующем этапе вы создадите обработчики.

XAML

```
<StackPanel Grid.Column="1" Orientation="Horizontal"
HorizontalAlignment="Right">
    <CheckBox Content="Enable Erasing Mode" Margin="20,0,4,0"
        Checked="ErasingModeCheckBox_Checked"
        Unchecked="ErasingModeCheckBox_Unchecked"/>
```

```
</StackPanel>
```

2. Создайте обработчики для событий в коде страницы. В отмеченном событии **InputProcessingConfiguration** настраивается на режим **Erasing (стирания)**.

C#

```
private void ErasingModeCheckBox_Checked(object sender, RoutedEventArgs e)
{
    InkCanvas.InkPresenter.InputProcessingConfiguration.Mode =
    InkInputProcessingMode.Erasing
}

private void ErasingModeCheckBox_Unchecked(object sender, RoutedEventArgs e)
{
    InkCanvas.InkPresenter.InputProcessingConfiguration.Mode =
    InkInputProcessingMode.Inking;
}
```

Примечание: Штрих, сделанный в режиме стирания, стирает любой рукописный штрих, с которым он пересекается. Весь штрих сотрется.

3. Скомпилируйте и запустите своё приложение. Рисуя в режиме обведения чернилами, запустите режим стирания путем проставления галочки в соответствующем окне. Перечеркните один из сделанных вами штрихов, чтобы увидеть, как он исчезает.
4. Остановите отладку и вернитесь в Visual Studio.
5. В **MainPage.xaml** добавьте кнопку **Очистить** в третьем ряду **StackPanel**. Настройте Click-событие на **OnClear**. На следующем этапе вы создадите обработчик события.

XAML

```
<StackPanel Grid.Row="2" Orientation="Horizontal" Margin="12">
    <Button Content="Clear" Click="OnClear" />
</StackPanel>
```

6. Добавьте обработчик **OnClear()** в код **MainPage**.

C#

```
void OnClear(object sender, RoutedEventArgs e)
{
    InkCanvas.InkPresenter.StrokeContainer.Clear();
}
```

7. Скомпилируйте и запустите приложение. Нарисуйте несколько штрихов и используйте кнопку **Очистить**, чтобы убрать их из InkCanvas в одно касание.

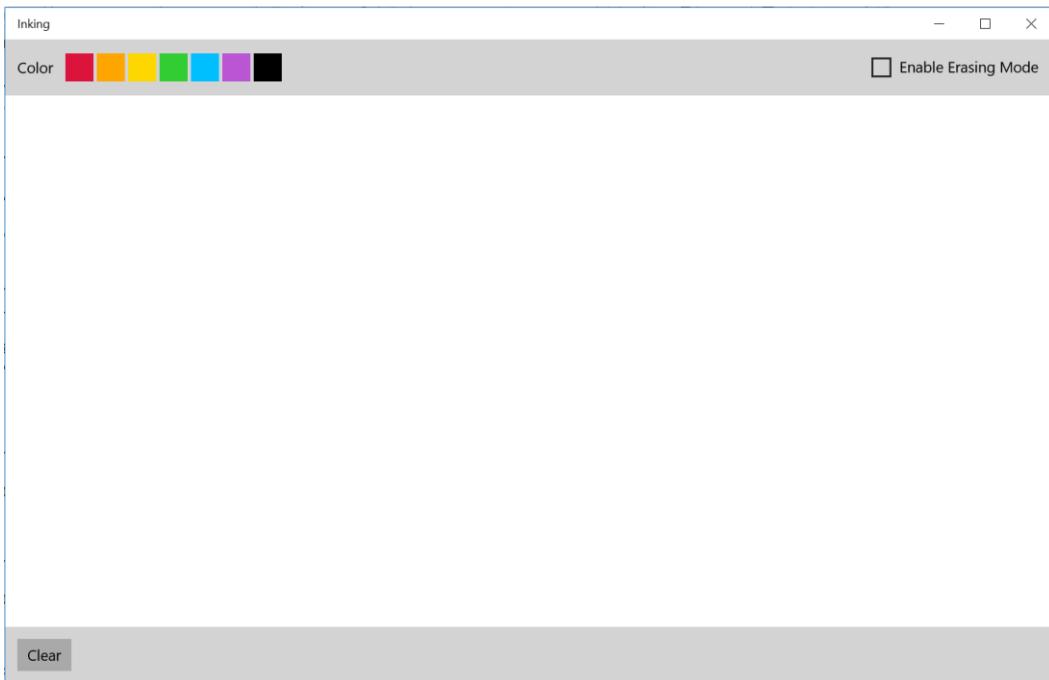


Рисунок 7

Режим стирания позволяет вам убирать одиночные штрихи, в то время как кнопка **Очистить** стирает всё, что вы нарисовали.

8. Завершите отладку и вернитесь в Visual Studio.

Задача 5 – Сохранить и загрузить

Режим работы с чернилами полезен тем, что он позволяет преобразовывать физические действия пользователя в пиксельные изображения на устройстве. Для повторного использования созданной информации, вы можете сохранить штрихи из InkCanvas и повторно загрузить их позднее. В этой задаче вы сохраните свои рукописные штрихи в GIF-файле и повторно загрузите их на полотно.

1. Добавьте кнопки **Сохранить** и **Загрузить** возле кнопки **Очистить** в MainPage.xaml. Настройте события щелчка на **OnSaveAsync** и **OnLoadAsync**, соответственно.

XAML

```
<StackPanel Grid.Row="2" Orientation="Horizontal" Margin="12">
    <Button Content="Clear" Click="OnClear" />
    <Button Content="Save" Margin="12,0,0,0" Click="OnSaveAsync"/>
    <Button Content="Load" Margin="12,0,0,0" Click="OnLoadAsync"/>
</StackPanel>
```

2. Во коде MainPage добавьте пространства имен **Windows.Storage** и **Windows.Storage.Streams**.

C#

```
using Windows.Storage;
```

```
using Windows.Storage.Streams;
```

3. Добавьте методы **OnSaveAsync** и **OnLoadAsync** в код страницы MainPage. Метод **OnSaveAsync** сохраняет штрихи в виде GIF-файла. Метод загрузки считывает эти штрихи обратно на InkCanvas.

C#

```
async void OnSaveAsync(object sender, RoutedEventArgs e)
{
    // Мы не хотим сохранять пустой файл
    if (InkCanvas.InkPresenter.StrokeContainer.GetStrokes().Count > 0)
    {
        var savePicker = new Windows.Storage.Pickers.FileSavePicker();
        savePicker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.PicturesLibrary;
        savePicker.FileTypeChoices.Add(
            "Gif with embedded ISF",
            new System.Collections.Generic.List<string> { ".gif" });

        Windows.Storage.StorageFile file =
            await savePicker.PickSaveFileAsync();
        if (null != file)
        {
            using (IRandomAccessStream stream =
                await file.OpenAsync(FileAccessMode.ReadWrite))
            {
                await InkCanvas.InkPresenter.StrokeContainer.SaveAsync(
                    stream);
            }
        }
    }
}

async void OnLoadAsync(object sender, RoutedEventArgs e)
{
    var openPicker = new Windows.Storage.Pickers.FileOpenPicker();
    openPicker.SuggestedStartLocation =
        Windows.Storage.Pickers.PickerLocationId.PicturesLibrary;
    openPicker.FileTypeFilter.Add(".gif");
    openPicker.FileTypeFilter.Add(".isf");
    Windows.Storage.StorageFile file = await
openPicker.PickSingleFileAsync();
    if (null != file)
    {
        using (var stream = await file.OpenSequentialReadAsync())
        {
            await InkCanvas.InkPresenter.StrokeContainer.LoadAsync(stream);
        }
    }
}
```

4. Скомпилируйте и запустите приложение. Рисуя на холсте, сохраняйте сделанные вами штрихи в виде GIF-файла в своей файловой системе. В процессе работы приложения очистите холст и используйте кнопку **Загрузка**, чтобы открыть браузер файлов. Выберите свой GIF-файл, чтобы перезагрузить штрихи обратно на InkCanvas.

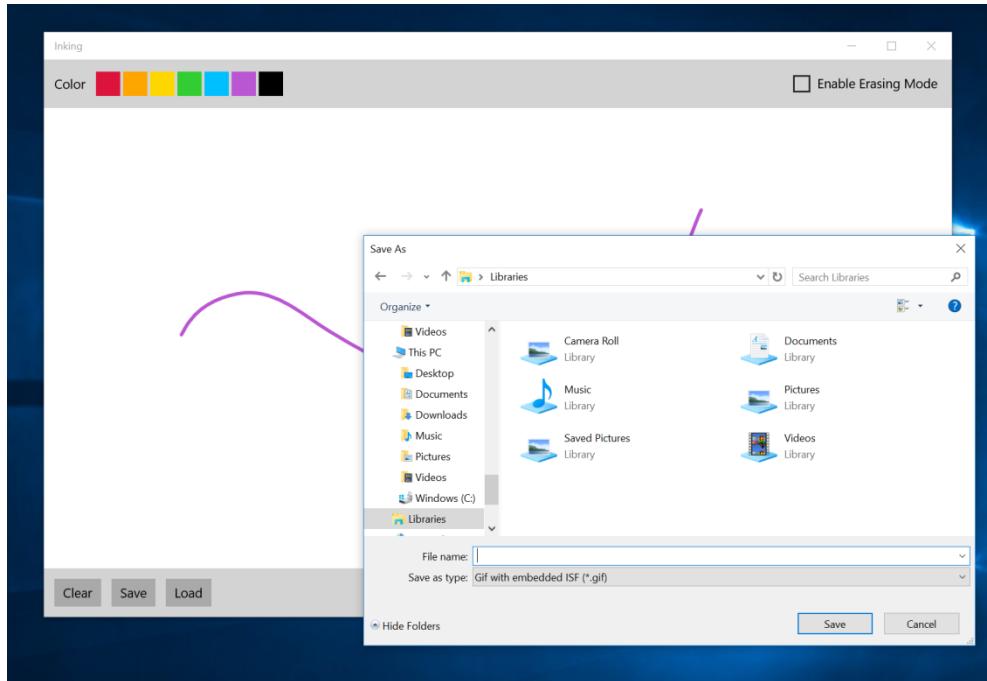


Рисунок 8

Сохраните сделанные вами штрихи в GIF-файле.

5. Остановите отладку и вернитесь в Visual Studio.

Упражнение 2: Распознавание рукописных текстов

Приложения с поддержкой рукописного ввода обычно включает чертёжи, аннотации к изображениям и функцию распознавания рукописных текстов. В этом упражнении мы рассмотрим распознавание рукописных текстов с InkCanvas, используя специфические для языка распознаватели, доступные на вашем устройстве.

Задача 1 – Создайте новое решение для проекта Распознавания рукописных текстов

Создайте новый проект для Распознавания рукописных текстов.

1. В новой версии Visual Studio 2015 выберите **File (Файл) -> New (Новый) -> Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed**

(Установленное) > Templates (Шаблоны) > Visual C# > Windows > Universal, а затем выберите шаблон **Blank App** приложения (Universal Windows). Назовите проект **Handwriting** (Рукописный текст). Сохраните проект в папку, в которой вы храните свои работы по настоящему курсу.

2. Настройте Текущую конфигурацию решения на **отладку** и Платформу решений в соответствии с **x86**. Выберите **Локальный компьютер** из выпадающего меню Цели отладки.
3. Скомпилируйте и запустите своё приложение. Вы увидите окно шаблона приложения со счетчиком частоты кадров, активированном по умолчанию в режиме отладки.

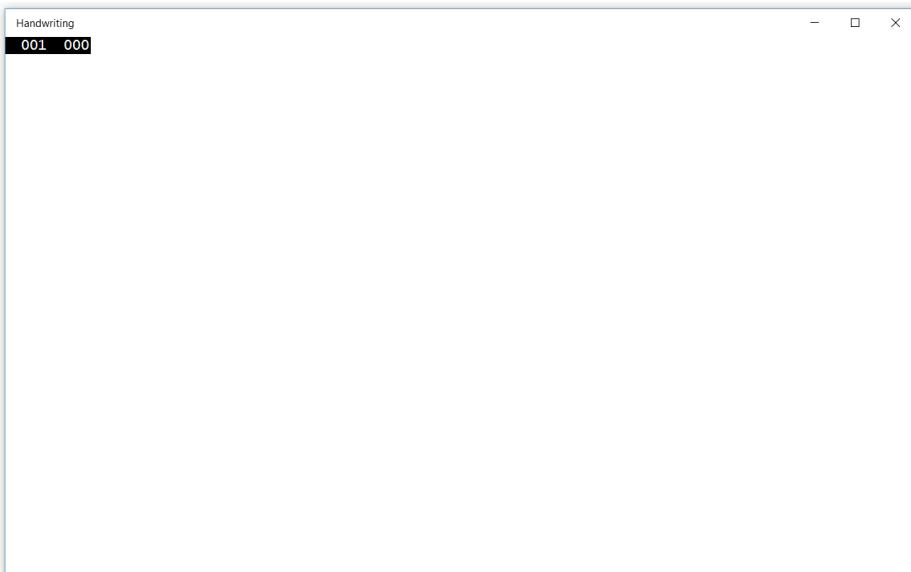


Рисунок 9

Пустое универсальное приложение, запущенное в режиме настольного компьютера.

Примечание: Директива препроцессора для активации или отключения счётчика частоты кадров находится в **App.xaml.cs**. Счётчик частоты кадра может перекрывать содержимое вашего приложения, если вы оставляете его включённым. Для целей настоящего курса вы можете отключить его, установив значение **this.DebugSettings.EnableFrameRateCounter** в **false**.

4. Вернитесь в Visual Studio и остановите отладку.

Задача 2 – Создайте дизайн окна

MainPage для этого упражнения будет выглядеть так же, как и для проекта в упражнении 1. В этой задаче мы создадим макет и установим InkCanvas.

1. Измените фон Grid на LightGray (светло-серый). Создайте такой же макет, который вы использовали в упражнении 1, используя определения строк, столбцов и элементы управления. Добавьте InkCanvas ко второй строке и назовите его с помощью **x:Name**.

XAML

```
<Grid Background="LightGray">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="2*" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
    <Grid Grid.Row="0" Margin="12">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="2*" />
            <ColumnDefinition Width="1*" />
        </Grid.ColumnDefinitions>
        <StackPanel Orientation="Horizontal" Grid.Column="0">
        </StackPanel>
        <StackPanel Grid.Column="1" HorizontalAlignment="Right"
            Orientation="Horizontal">
        </StackPanel>
    </Grid>
    <Grid Grid.Row="1" Background="White" Grid.ColumnSpan="2">
        <InkCanvas x:Name="InkCanvas" />
    </Grid>
    <StackPanel Grid.Row="2" Orientation="Horizontal" Margin="12">
    </StackPanel>
</Grid>
```

- Добавьте пространство имен **Windows.UI.Input.Inking** в выделенный код.

C#

```
using Windows.UI.Input.Inking;
```

- Инициализируйте графические атрибуты в конструкторе MainPage.

C#

```
public MainPage()
{
    this.InitializeComponent();

    InkDrawingAttributes drawingAttributes = new InkDrawingAttributes();
    drawingAttributes.Color = Windows.UI.Colors.Black;
    drawingAttributes.Size = new Size(4, 4);
    drawingAttributes.IgnorePressure = false;
    drawingAttributes.FitToCurve = true;

    InkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(
        drawingAttributes);
    InkCanvas.InkPresenter.InputDeviceTypes =
        Windows.UI.Core.CoreInputDeviceTypes.Mouse |
        Windows.UI.Core.CoreInputDeviceTypes.Pen |
        Windows.UI.Core.CoreInputDeviceTypes.Touch;
}
```

4. Скомпилируйте и запустите своё приложение. Вы увидите простые возможности режима обведения чернилами, аналогичные тем, которые мы делали в предыдущем упражнении.

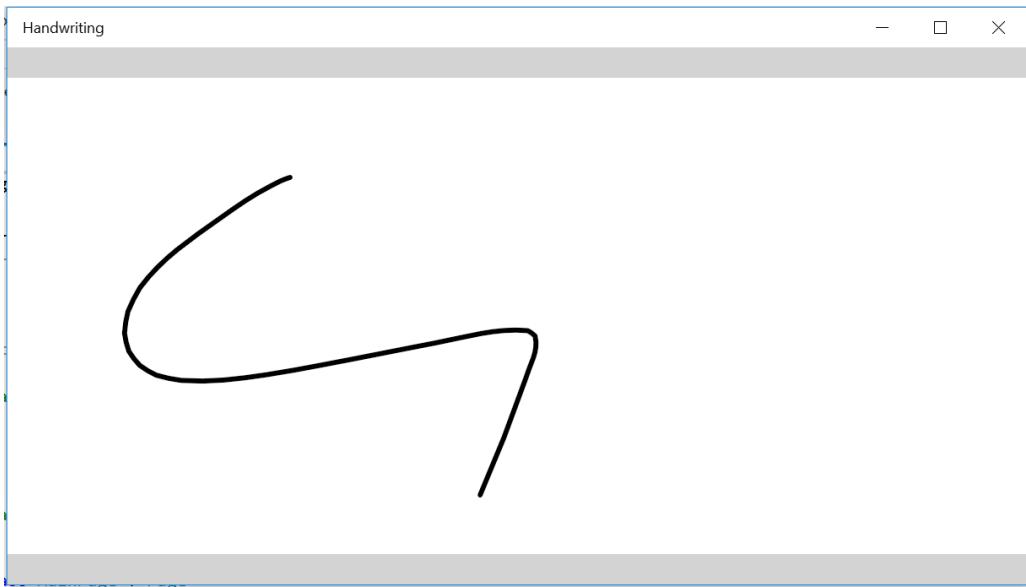


Рисунок 10

Активация приложения распознавания рукописных текстов.

5. Завершите отладку и вернитесь в Visual Studio.

Задача 3 – Настройте распознаватель чернил

InkRecognizer отвечает за все этапы распознавания рукописных текстов. В этой задаче вы настроите словарь RecognizerHelper и InkRecognizer.

1. Щёлкните правой кнопкой мыши по наименованию проекта **Рукописного текста** и выберите Add (Добавить) > Existing Item (Существующий элемент). Найдите в папке **Lab Assets** и добавьте в проект файл **RecognizerHelper.cs**.
2. Откройте **RecognizerHelper.cs**. В нем описан вспомогательный класс, содержащий словарь распознавателей, которые могут быть установлены на вашем устройстве. Он переводит теги языка распознавания на более длинные, удобочитаемые имена, чтобы помочь пользователям идентифицировать распознаватель, который они выбрали. Сохраните и закройте вспомогательный элемент.
3. Добавьте метки TextBlock и ComboBox к **StackPanel** в строке 0, столбце 0 в MainPage.xaml.

XAML

```
<StackPanel Orientation="Horizontal" Grid.Column="0">
    <TextBlock Text="Available Recognizers:" Margin="0,8"/>
    <ComboBox
        x:Name="RecoName"
```

```

        MaxWidth="500"
        SelectionChanged="OnRecognizerChanged">
    </ComboBox>
</StackPanel>
```

4. Создайте строку состояния в третьей строке, используя **TextBlock**. Текст состояния отобразит результаты распознавания рукописных текстов или сообщение, если в системе не было обнаружено никаких распознавателей.

XAML

```
<StackPanel Grid.Row="2" Orientation="Horizontal" Margin="12">
    <TextBlock x:Name="Status" Margin="20,0,0,0" />
</StackPanel>
```

5. Добавьте пространство имен **Windows.Globalization** в код **MainPage**.

C#

```
using Windows.Globalization;
```

6. Создайте экземпляры **InkRecognizers** и список **InkRecognizerContainer**, доступный только для чтения.

C#

```
Рукописный текст пространства имён
{
    public sealed partial class MainPage : Page
    {
        InkRecognizerContainer inkRecognizerContainer = null;
        private IReadOnlyList<InkRecognizer> recoView = null;
```

7. Загрузите доступные распознаватели системы в конструкторе **MainPage**.

C#

```
public MainPage()
{
    this.InitializeComponent();
    InkDrawingAttributes drawingAttributes = new
        InkDrawingAttributes();
    drawingAttributes.Color = Windows.UI.Colors.Black;
    drawingAttributes.Size = new Size(4, 4);
    drawingAttributes.IgnorePressure = false;
    drawingAttributes.FitToCurve = true;

    inkRecognizerContainer = new InkRecognizerContainer();
    recoView = inkRecognizerContainer.GetRecognizers();
    if (recoView.Count > 0)
    {
        foreach (InkRecognizer recognizer in recoView)
        {
            RecoName.Items.Add(recognizer.Name);
        }
    }
}
```

```

    }
    else
    {
        RecoName.IsEnabled = false;
        RecoName.Items.Add("No Recognizer Available");
    }
RecoName.SelectedIndex = 0;

InkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(
    drawingAttributes);
InkCanvas.InkPresenter.InputDeviceTypes =
    Windows.UI.Core.CoreInputDeviceTypes.Mouse |
    Windows.UI.Core.CoreInputDeviceTypes.Pen |
    Windows.UI.Core.CoreInputDeviceTypes.Touch;
}

```

- Создайте обработчик **OnRecognizerChanged** для выбора значения распознавателя из ComboBox и метод **SetRecognizerByName** для установки распознавателя по имени.

C#

```

void OnRecognizerChanged(object sender, RoutedEventArgs e)
{
    string selectedValue = (string)RecoName.SelectedValue;
    SetRecognizerByName(selectedValue);
}

bool SetRecognizerByName(string recognizerName)
{
    bool recognizerFound = false;

    foreach (InkRecognizer reco in recoView)
    {
        if (recognizerName == reco.Name)
        {
            inkRecognizerContainer.SetDefaultRecognizer(reco);
            recognizerFound = true;
            break;
        }
    }

    if (!recognizerFound)
    {
        Status.Text = "Could not find target recognizer.";
    }

    return recognizerFound;
}

```

- Скомпилируйте и запустите приложение. Используйте ComboBox для просмотра списка распознавателей, установленных в системе.

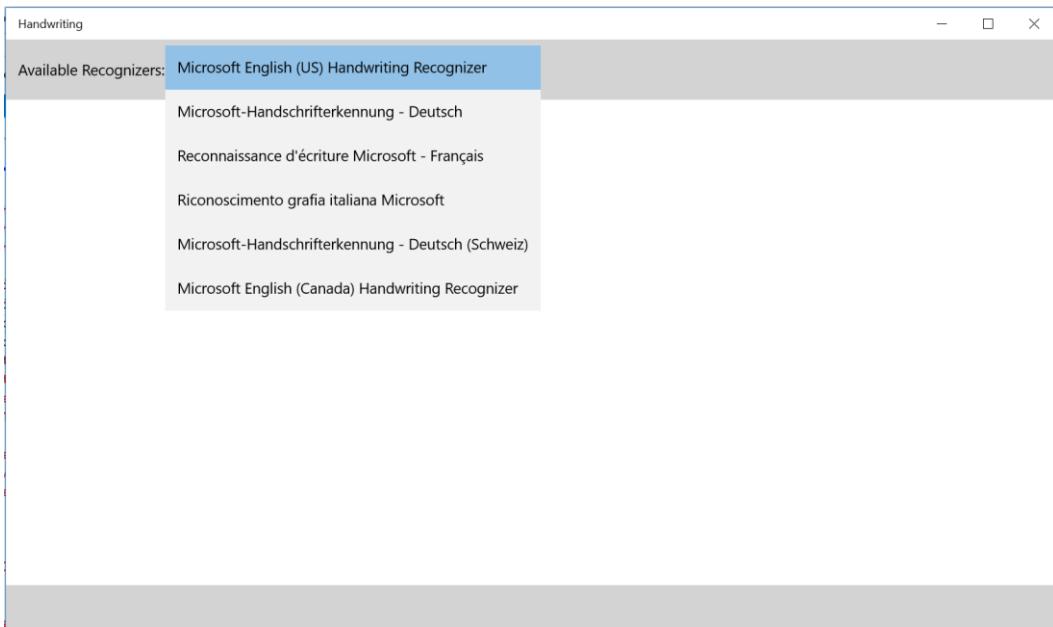


Рисунок 11

ComboBox предлагает распознаватели, доступные на устройстве.

10. Остановите отладку и вернитесь в Visual Studio.

Задача 4 – Примените OnRecognizeAsync()

Теперь, когда настроены распознаватели, вы можете осуществить распознавание рукописного текста в своём приложении.

1. Добавьте кнопки **Распознать** и **Очистить** ко второму столбцу первой строки в MainPage.

XAML

```
<StackPanel Grid.Column="1" HorizontalAlignment="Right"
Orientation="Horizontal">
    <Button x:Name="RecognizeBtn" Content="Recognize" Width="100"
        Margin="0,0,4,0" Click="OnRecognizeAsync"/>
    <Button x:Name="ClearBtn" Content="Clear" Width="65" Margin="0,0,4,0"
        Click="OnClear"/>
</StackPanel>
```

2. Добавьте обработчик **OnClear()** в код MainPage. Этот метод идентичен методу **OnClear()**, который мы использовали в упражнении 1.

C#

```
void OnClear(object sender, RoutedEventArgs e)
{
    InkCanvas.InkPresenter.StrokeContainer.Clear();
}
```

3. Добавьте обработчик **OnRecognizeAsync()** в выделенный код. Этот метод асинхронно распознает штрихи, нанесенные в InkCanvas, с использованием выбранного

распознавателя. Если будут результаты, они отображаются в строке состояния. Если результаты будут ошибочными, вы получите сообщение **нет распознанных текстов**.

C#

```
async void OnRecognizeAsync(object sender, RoutedEventArgs e)
{
    IReadOnlyList<InkStroke> currentStrokes =
    InkCanvas.InkPresenter.StrokeContainer.GetStrokes();
    if (currentStrokes.Count > 0)
    {
        RecognizeBtn.IsEnabled = false;
        ClearBtn.IsEnabled = false;
        RecoName.IsEnabled = false;

        var recognitionResults = await
inkRecognizerContainer.RecognizeAsync(
            InkCanvas.InkPresenter.StrokeContainer,
            InkRecognitionTarget.All);

        if (recognitionResults.Count > 0)
        {
            // Показать результат распознавания
            string str = "Recognition result:";
            foreach (var r in recognitionResults)
            {
                str += " " + r.GetTextCandidates()[0];
            }
            Status.Text = str;
        }
        else
        {
            Status.Text = "No text recognized.";
        }

        RecognizeBtn.IsEnabled = true;
        ClearBtn.IsEnabled = true;
        RecoName.IsEnabled = true;
    }
    else
    {
        Status.Text = "Must first write something.";
    }
}
```

4. Скомпилируйте и запустите приложение. Выберите распознаватель из списка и запишите слово из соответствующего языка на полотне. Используйте кнопку распознавания для просмотра результатов



Рисунок 12

Успешное распознавание рукописного текста.

5. Завершите отладку и вернитесь в Visual Studio.

Краткий обзор

Чернила – естественный и удобный способ взаимодействия с пользователем на устройствах Windows 10, которые имеют опции ввода данных с помощью мыши, прикосновений или пера. Рисование чернилами быстро и легко реализовать в приложениях UWP. В этой работе мы рассмотрели выбор настройки атрибутов чернил и стилей, очистки и стирания с полотна и сохранения и загрузки штрихов. Мы также применили распознавание рукописных текстов для языковых пакетов, установленных на устройстве.



Лабораторный практикум

Службы приложений

Октябрь 2015 года



Обзор

Службы приложений – это «фоновые» службы, не содержащие пользовательского интерфейса, которые могут инкапсулировать бизнес-логику, данные и транзакции для приложений без потери производительности для запуска приложения. Они обеспечивают функциональность, сходную с веб-службами, но для приложений из магазина приложений. Некоторые из сценариев использования служб приложений - извлечение и кэширование данных.

В случае извлечения данных вы можете сделать так, чтобы ваше приложение имело доступ или могло обрабатывать данные другого приложения без запуска этого приложения. Службы приложений, использующие фоновые задачи, являются эффективным способом получения доступа к таким данным.

Геолокационные данные являются великолепным примером использования кэшированных данных с целью сохранения ресурсов. Таким образом ваше приложение может получить доступ к локальным данным карт при минимальном использовании передачи данных. Если другое приложение, например, карты Bing, уже загрузило и кэшировало информацию, необходимую вашему приложению, вы можете использовать службу приложения для извлечения такого рода информации без использования дополнительной пропускной способности.

В рамках данной работы вы создадите службу для приложения по поиску сотрудников, которое считывает один или несколько идентификаторов сотрудников в качестве входных данных и выдает соответствующие ФИО сотрудников. Затем вы научитесь создавать приложение для запроса службы и отображения результатов.

Цели

Настоящий курс научит вас:

- Создавать и регистрировать службу приложения
- Вызывать службу приложения из другого приложения
- Передавать данные обратно в вызывающее приложение
- Отлаживать службу приложения

Системные требования

Для завершения настоящего курса необходимы:

- Microsoft Windows 10
- Microsoft Visual Studio 2015

Настройка

Вам следует выполнить следующие действия для подготовки компьютера:

1. Установить Microsoft Windows 10.
 2. Установить Microsoft Visual Studio 2015.
-

Упражнения

Настоящая работа включает в себя следующие упражнения:

1. Создание и регистрация службы приложения
 2. Вызов службы приложения из другого приложения
-

Расчетное время для завершения курса: **45-60 минут.**

Упражнение 1: Создание и регистрация службы приложения

В рамках данного упражнения вы создадите решение для службы приложения по поиску сотрудников. Решение будет содержать компонент Windows Runtime Component со службой приложения, а также приложение, с помощью которого вы сможете зарегистрировать службу приложения. Мы не будем создавать пользовательский интерфейс для этого приложения, но мы будем использовать наименование соответствующего пакета позже для того, чтобы другие приложения могли взаимодействовать со службой приложения.

Задача 1 – Создать проект EmployeeLookupService

Создайте новое решение для проекта службы приложения.

1. В новой версии Visual Studio 2015 выберите **File (Файл) -> New (Новый) -> Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed (Установленное) > Templates (Шаблоны) > Visual C# > Windows > Universal**, а затем выберите шаблон **Blank App (Universal Windows)**.
2. Назовите свой проект "**EmployeeLookupService**". Сохраните проект в папку, в которой вы храните свои работы по настоящему курсу.

3. Настройте Solution Configuration (Текущую конфигурацию решения) на **Debug (Отладку)** и Solution Platform (Платформу решений) на **x86**. Выберите **Local Machine (Локальный компьютер)** из выпадающего меню Debug Target.
4. Создайте и запустите свое приложение. Вы увидите окно Blank App со счетчиком частоты кадров.

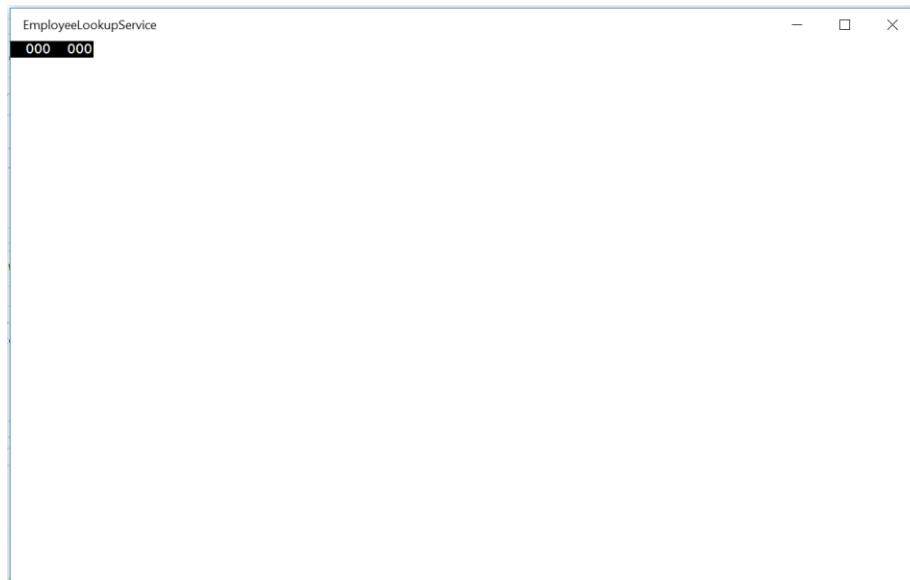


Рисунок 1

Приложение Blank universal выполняется в режиме Рабочего стола.

Примечание: В рамках данного курса мы будем использовать приложение EmployeeLookupService как контейнер для службы приложения, выполняемого в качестве фоновой задачи. Отображение главной страницы будет оставаться пустым.

5. Вернитесь к Visual Studio и завершите отладку.

Задача 2 – Создайте компонент Windows Runtime Component

В рамках проекта служба приложения будет функционировать в качестве фоновой задачи с компонентом Windows Runtime Component. Мы начнем с создания компонента Windows Runtime Component.

1. Щелкните правой кнопкой мыши на решение EmployeeLookupService и выберите **Add (Добавить) > New Project (Новый проект)**. Добавьте проект: **Visual C# > Windows > Universal > Windows Runtime Component (Universal Windows)**. Назовите его "EmployeeLookupService.Background".

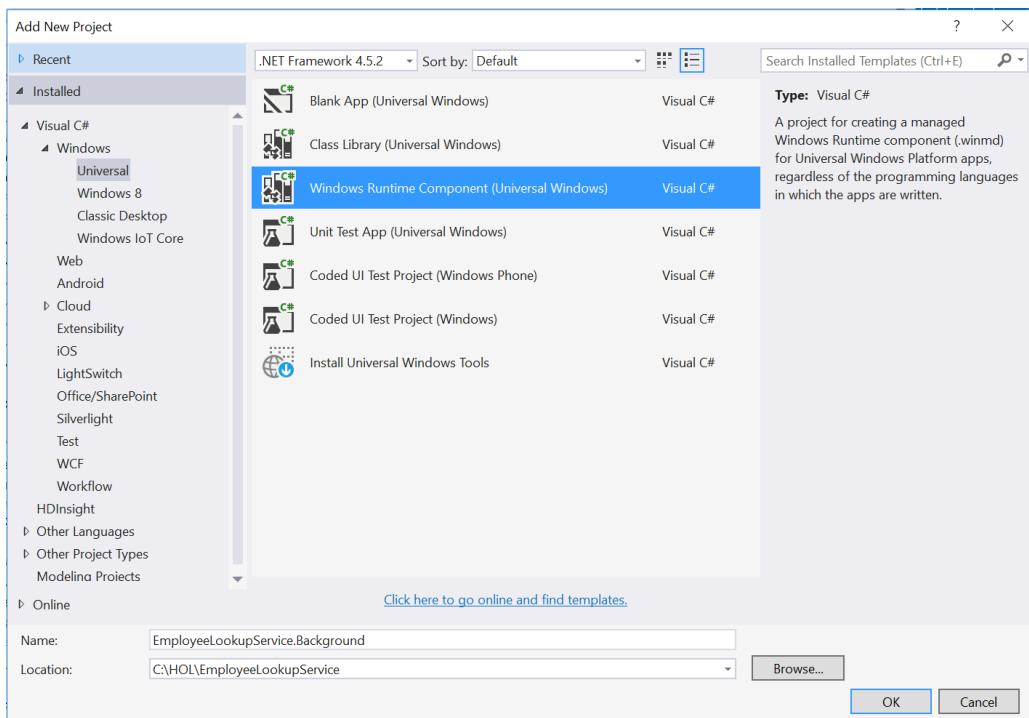


Рисунок 2

Добавьте компонент WinRT для Службы приложения.

2. Щелкните правой кнопкой мыши по Class1.cs в проекте EmployeeLookupService.Background и выберите Rename (Переименовать). Назовите проект "EmployeeLookup.cs". Если необходимо выполнить переименование всех ссылок на "Class1", выберите Yes (Да).

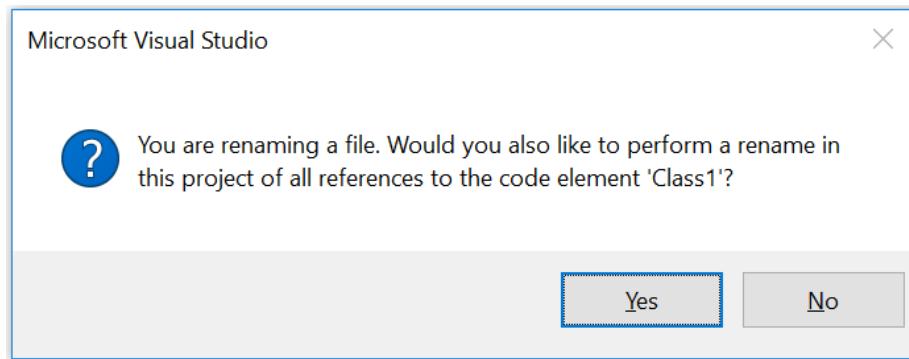


Рисунок 3

Переименуйте "Class1.cs" в "EmployeeLookup.cs".

Задача 3 – Реализовать интерфейс IBackgroundTask

Сейчас, когда ваш компонент WinRT подготовлен, вы можете реализовать базовую структуру фоновой задачи.

1. Откройте EmployeeLookup.cs и добавьте пространство имен Windows.ApplicationModel.Background.

C#

```
using Windows.ApplicationModel.Background;
```

2. Реализуйте интерфейс IBackgroundTask.

C#

```
public sealed class EmployeeLookup : IBackgroundTask
{
    public void Run(IBackgroundTaskInstance taskInstance)
    {
        throw new NotImplementedException();
    }
}
```

3. Создайте экземпляр deferral на уровне класса и получите экземпляр deferral в методе **Run**. Подпишитесь на событие **taskInstance.Canceled**, чтобы обработать событие отмены задачи.

C#

```
private BackgroundTaskDeferral deferral;

public void Run(IBackgroundTaskInstance taskInstance)
{
    this.deferral = taskInstance.GetDeferral();

    taskInstance.Canceled += TaskInstance_Canceled;

}

private void TaskInstance_Canceled(IBackgroundTaskInstance sender,
BackgroundTaskCancellationReason reason)
{
    if (this.deferral != null)
    {
        this.deferral.Complete();
    }
}
```

Задача 4 – Осуществить управление соединением со Службой приложения

Служба приложения, которую мы создаем, считает идентификатор сотрудника в качестве входных данных и выдает соответствующее ФИО сотрудника. В рамках данной задачи вы сможете создать простой справочник с указанием информации о сотрудниках, обнаруживать входящее соединение со службой приложения, а также возвращать сообщение с запрашиваемой информацией.

Примечание: AppServiceConnection – это асинхронное соединение с точкой входа службы приложения, открытое вызывающим приложением. Вызывающее приложение может отправить сообщение в службу приложения, которая может обработать сообщение и вернуть ответ. Вы научитесь открывать соединение в следующем упражнении. Для ознакомления с дополнительной информацией об AppServiceConnection посетите страницу:
<https://msdn.microsoft.com/en-us/library/windows/apps/windows.applicationmodel.appservice.appserviceconnection.aspx>

1. Создайте закрытый справочник на уровне классов для хранения данных о списке сотрудников. Возможно предоставление информации одного из четырех сотрудников.

C#

```
public sealed class EmployeeLookup : IBackgroundTask
{
    private BackgroundTaskDeferral deferral;

    private Dictionary<string, string> employees = new Dictionary<string,
        string> { { "0", "John Smith" }, { "1", "Mary Echo" }, { "2", "Jane
        Doe" }, { "3", "Bob Harvey" } };
```

2. Добавьте пространство имен **Windows.ApplicationModel.AppService** в **EmployeeLookup.cs**.

C#

```
using Windows.ApplicationModel.AppService;
```

3. Добавьте переменную **appServiceConnection** на уровень класса. В реализации метода **Run** установите **appServiceConnection** на входящее соединение со службой приложения для данной задачи.

C#

```
public sealed class EmployeeLookup : IBackgroundTask
{
    private BackgroundTaskDeferral deferral;

    private AppServiceConnection appServiceConnection;

    private Dictionary<string, string> employees = new Dictionary<string,
        string> { { "0", "John Smith" }, { "1", "Mary Echo" }, { "2", "Jane
        Doe" }, { "3", "Bob Harvey" } };

    public void Run(IBackgroundTaskInstance taskInstance)
    {
        this.deferral = taskInstance.GetDeferral();

        taskInstance.Canceled += TaskInstance_Canceled;
```

```
        var trigger = taskInstance.TriggerDetails as  
AppServiceTriggerDetails;  
        this.appServiceConnection = trigger.AppServiceConnection;  
    }
```

4. Подпишитесь на событие подключения к службе приложения **RequestReceived** в конце метода **Run**. После обработки запроса мы должны уведомить об этом через объект deferral.

C#

```
this.appServiceConnection = trigger.AppServiceConnection;  
this.appServiceConnection.RequestReceived +=  
    AppServiceConnection_RequestReceived;  
}  
  
private void AppServiceConnection_RequestReceived(AppServiceConnection  
sender, AppServiceRequestReceivedEventArgs args)  
{  
    var deferral = args.GetDeferral();  
  
    deferral.Complete();  
}
```

5. Добавьте пространство имен **Windows.Foundation.Collections** к классу EmployeeLookup.

C#

```
using Windows.Foundation.Collections;
```

6. Любое приложение, которое вызывает службу поиска сотрудников, должно будет отправить сообщение с ID сотрудника для последующего поиска. Простой объект может быть передан в фоновую задачу с использованием ValueSet. В обработчике событий AppServiceConnection_RequestReceived получите входящее сообщение и возвратите ValueSet, содержащий результаты поиска сотрудников, в качестве ответа. ID сотрудника будет являться ключом для ответа, а его или ее ФИО будет являться значением.

C#

```
private async void  
AppServiceConnection_RequestReceived(AppServiceConnection sender,  
AppServiceRequestReceivedEventArgs args)  
{  
    var deferral = args.GetDeferral();  
    var requestMessage = args.Request.Message;  
    var responseMessage = new ValueSet();  
  
    foreach (var item in requestMessage)  
    {  
        if (employees.ContainsKey(item.Value.ToString()))  
        {  
            responseMessage.Add(item.Value.ToString(),  
                employees[item.Value.ToString()]);  
        }  
    }  
    deferral.Complete();  
}
```

```

        }

        await args.Request.SendResponseAsync(responseMessage);

        deferral.Complete();
    }
}

```

Примечание: ValueSet представляет собой словарь с ключом типа string и значением типа object. Только сериализуемые типы могут использоваться в качестве значений. Для ознакомления с дополнительной информацией о ValueSet посетите страницу:

<https://msdn.microsoft.com/en-us/library/windows/apps/windows.foundation.collections.valueset.aspx>

Задача 5 – Зарегистрируйте Службу приложения

Служба приложения должна быть зарегистрирована, прежде чем она станет доступной. Мы зарегистрируем службу приложения EmployeeLookup в соответствующем приложении и определим точку входа в манифесте.

1. Откройте пакет **package.appxmanifest** приложения EmployeeLookupService в редакторе манифеста.
2. На вкладке **Declarations (Объявления)** добавьте объявление **App Service (Служба приложения)**. Присвойте ему имя "**EmployeeLookupService**" и установите точку входа на **EmployeeLookupService.Background.EmployeeLookup**. Сохраните манифест.

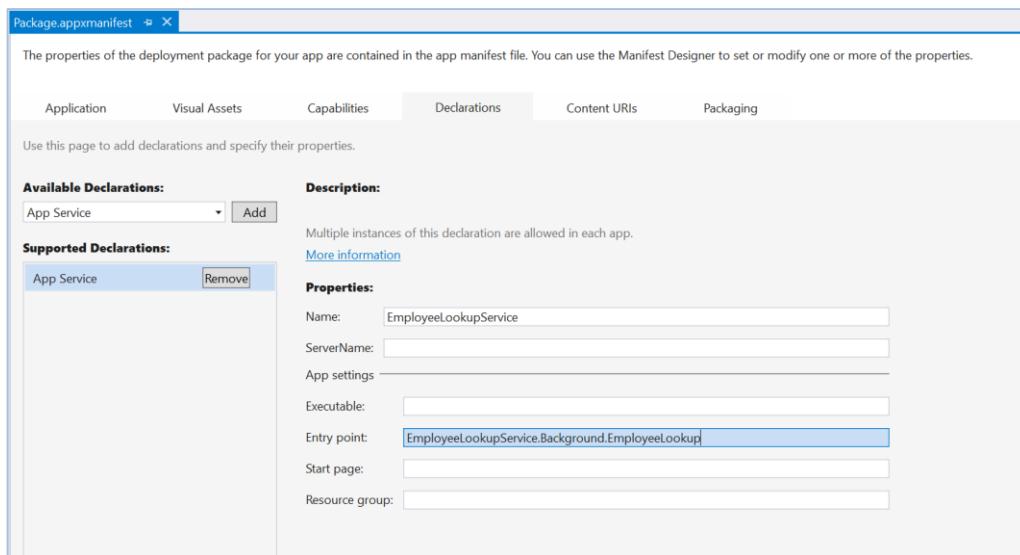


Рисунок 4

Зарегистрируйте службу приложения в манифесте EmployeeLookupService.

3. Щелкните правой кнопкой мыши на папку **References (Ссылки)** в приложении EmployeeLookupService, а также выберите **Add (Добавить) > Reference (Ссылка)**. Добавьте ссылку в проект **EmployeeLookupService.Background**.

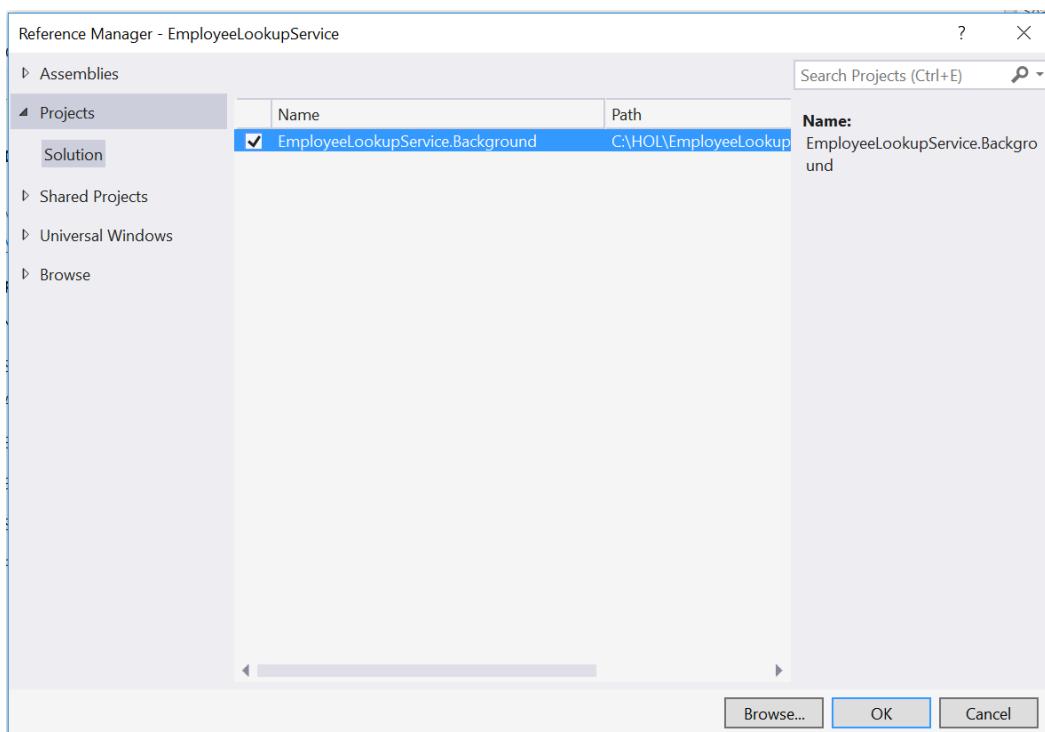


Рисунок 5

Добавьте компонент WinRT в качестве ссылки.

4. Создайте и запустите приложение EmployeeLookupService, чтобы развернуть свою фоновую задачу. Главная страница приложения будет оставаться пустой, так как мы не реализовали пользовательский интерфейс для этого приложения.

Примечание: Мы будем рассматривать способы отладки фоновой задачи в следующем упражнении.

5. Завершите отладку и вернитесь в Visual Studio.

Упражнение 2: Вызов службы приложения из другого приложения

Сейчас, когда вы уже создали и зарегистрировали Службу приложения, вы можете вызвать ее из другого приложения. В этом упражнении вы создадите приложение, в которое вы сможете ввести ID сотрудников, вызвать службу EmployeeLookup и получить результаты с соответствующим ФИО сотрудника.

Задача 1 – Создать пустое приложение Universal Windows

Мы начнем с создания проекта на основе шаблона Blank App (Пустого приложения).

1. В новой версии Visual Studio 2015 выберите **File (Файл) -> New (Новый) -> Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed (Установленное) > Templates (Шаблоны) > Visual C# > Windows > Universal**, а затем выберите шаблон **Blank App (Universal Windows)**.
2. Назовите свой проект "**AppServices**" и выберите местоположение файловой системы, в которое будет осуществлено сохранение результатов прохождения Лабораторного практикума.

Не изменяйте настройки, установленные для **Create new solution (Создания нового решения)** и **Create directory for solution (Создания папки для решения)**. Вы можете снять галочки как с **Add to source control (Добавить в систему контроля версий)**, так и **Show telemetry in the Windows Dev Center (Отобразить телеметрию в Windows Dev Center)**, если не хотите использовать соответствующие возможности. Нажмите **OK** для создания проекта.

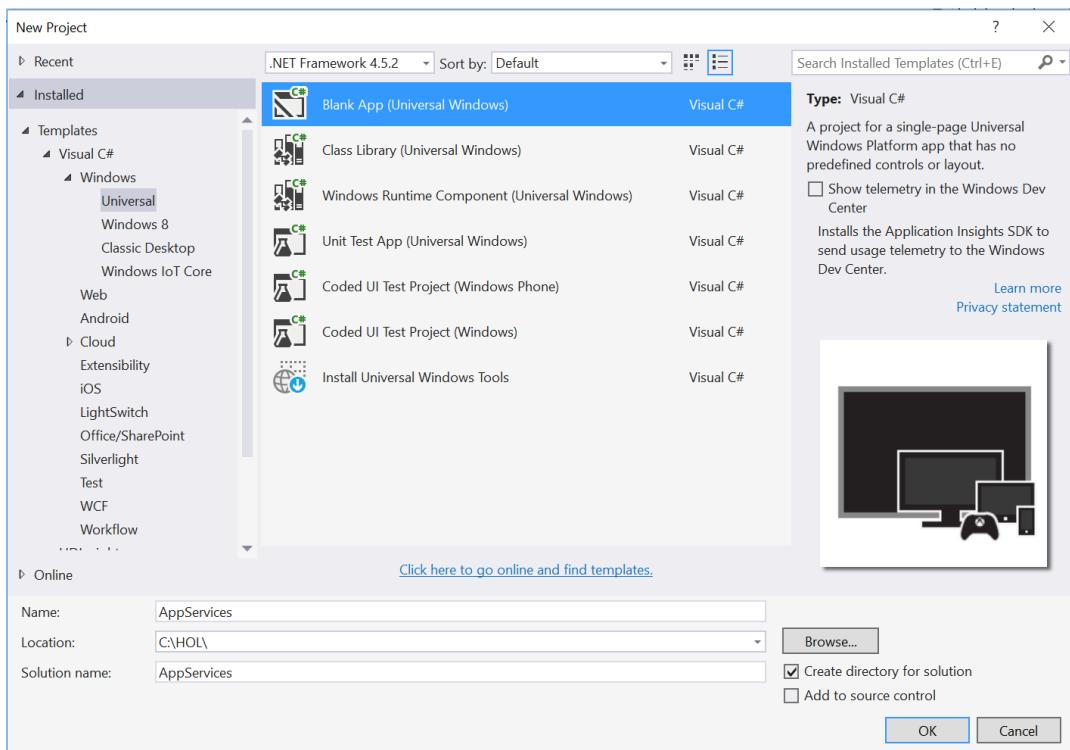


Рисунок 6

Создайте новый проект Blank App (Пустое приложение) в Visual Studio 2015.

3. Настройте Solution Configuration (Текущую конфигурацию решения) на Debug (Отладку) и Solution Platform (Платформу решений) на x86. Выберите Local Machine (Локальный компьютер) из выпадающего меню Debug Target (Цели отладки).

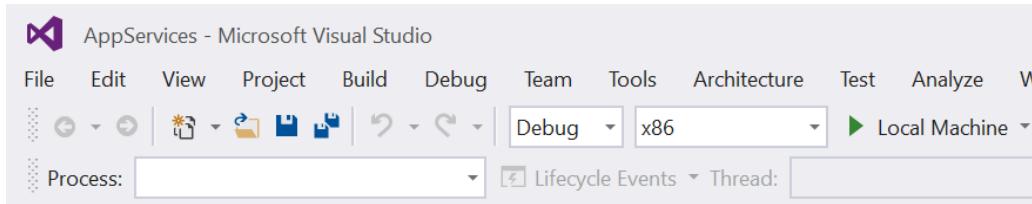


Рисунок 7

Сконфигурируйте свое приложение, которое должно выполниться на Локальном компьютере.

4. Создайте и запустите свое приложение. Вы увидите окно Blank App со счетчиком частоты кадров.

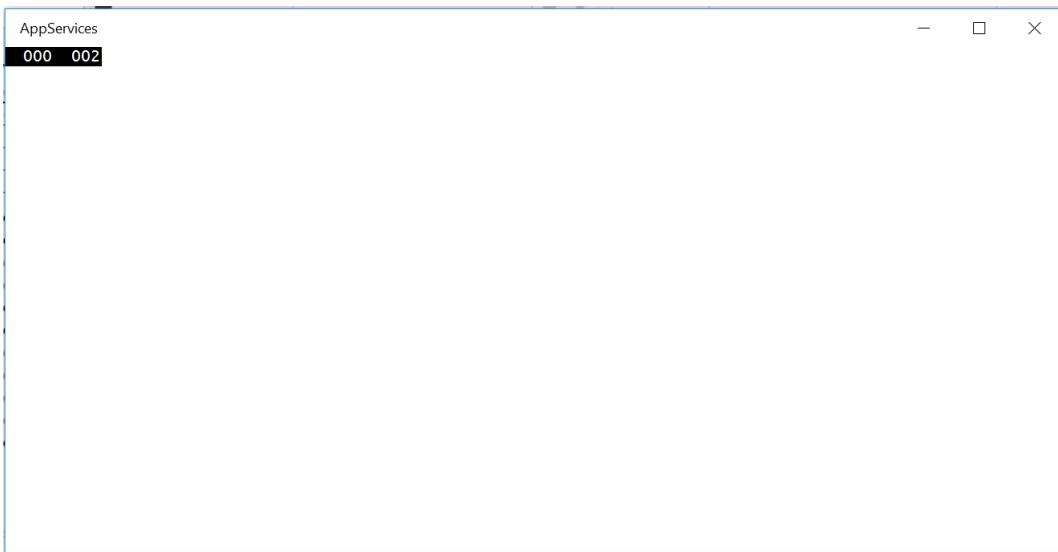


Рисунок 8

Приложение *Blank universal* выполняется в режиме Рабочего стола.

Примечание: Счетчик частоты кадров является инструментом, используемым в процессе отладки, который помогает следить за производительностью вашего приложения. Он полезен для тех приложений, которые требуют интенсивной графической обработки, однако не нужен для простых приложений, которые будут создаваться вами на данный момент.

В шаблоне Blank App директива препроцессора активирует или отключает счетчик частоты кадров посредством **App.xaml.cs**. Счетчик частоты кадров может перекрывать контент вашего приложения, если не убрать его. При выполнении данных работ вы можете отключить его, установив **this.DebugSettings.EnableFrameRateCounter** в **False**.

5. Вернитесь к Visual Studio и завершите отладку.

Задача 2 – Создайте пользовательский интерфейс

Мы создадим простой класс Employee со свойствами ID и ФИО для упрощения процедуры отображения результатов поиска сотрудника на основе службы поиска. Затем мы создадим простой пользовательский интерфейс, чтобы разрешить пользователям вводить ID и запрашивать службу приложения.

1. Щелкните правой кнопкой мыши по проекту AppServices и выберите **Add (Добавить) > Class (Класс)**. Назовите класс "Employee.cs".
2. Сделайте класс Employee открытым и добавьте в него строковые свойства **Id** и **Name**.

```
C#
public class Employee
{
    public string Id { get; set; }
    public string Name { get; set; }
}
```

3. Сохраните и закройте класс Employee.
4. Добавьте панель **StackPanel**, содержащую элементы управления **TextBlock** (**Текстовый блок**), **TextBox** (**Текстовое окно**) и **Button** (**Кнопку**), к элементам Grid для MainPage (Главной страницы) приложения AppServices.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel>
        <TextBlock Text="Enter one employee ID per line." Margin="12" />
        <TextBox x:Name="EmployeeId" AcceptsReturn="True"
            Margin="12,0,12,12"/>
        <Button Content="Look up employee(s)" Margin="12,0,0,0"/>
    </StackPanel>
</Grid>
```

5. Добавьте событие Click кнопки, чтобы вызвать обработчик **GetEmployeeById**. Вы создадите обработчик на следующем этапе.

XAML

```
<Button Content="Look up employee(s)" Click="GetEmployeeById"
    Margin="12,0,0,0"/>
```

6. Создайте обработчик **GetEmployeeById** в коде MainPage.

C#

```
public MainPage()
{
    this.InitializeComponent();
}

private void GetEmployeeById(object sender, RoutedEventArgs e)
{}
```

Задача 3 – Открыть соединение со Службой приложения

Служба приложения, созданная вами в упражнении 1, настроена на получение и обработку входящего соединения. В рамках данной задачи вы откроете соединение со службой приложения и отправите сообщение в службу приложения с запросом на поиск сотрудников.

1. Добавьте пространство имен **Windows.ApplicationModel.AppService** к коду MainPage.

C#

```
using Windows.ApplicationModel.AppService;
```

2. Создайте переменную для хранения соединения со службой приложения на уровне класса MainPage.

C#

```
public sealed partial class MainPage : Страница
{
    private AppServiceConnection appServiceConnection;
```

3. В отдельный версии Visual Studio вы можете загрузить проект **EmployeeLookupService** и открыть манифест приложения в редакторе манифеста. Под вкладкой **Packaging** (**Пакет**) найдите **Package family name** и скопируйте данное наименование в буфер обмена.

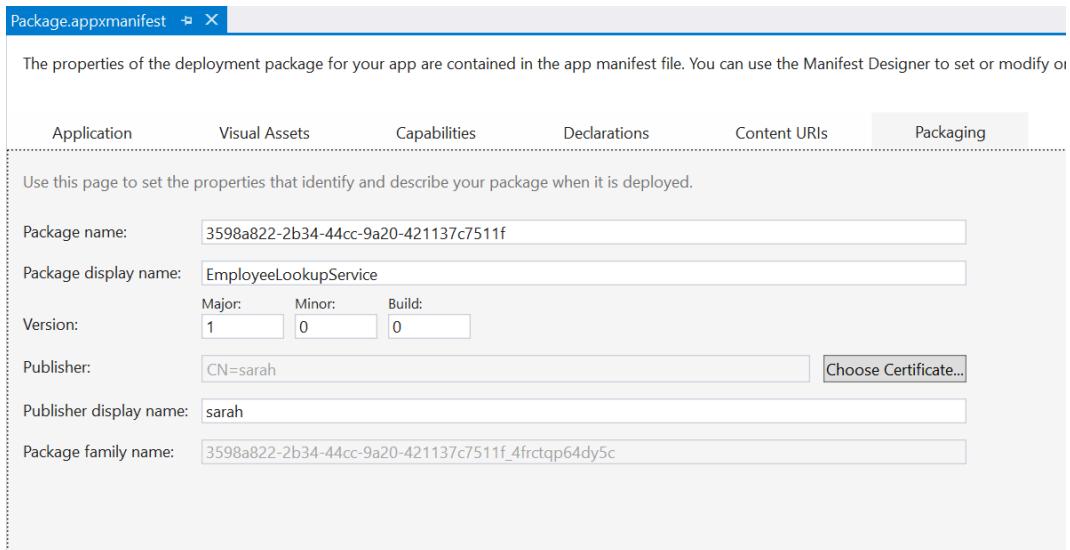


Рисунок 9

Скопируйте **Package family name** из манифеста приложения **EmployeeLookupService**.

4. Настройте соединение со службой приложения в обработчике **GetEmployeeById()**, чтобы иметь доступ к службе приложения **EmployeeLookup**. Вам понадобится наименование службы, заданной в манифесте Службы приложения, а также наименование пакета для семейства программных систем, которое вы сохранили на последнем этапе.

C#

```
private async void GetEmployeeById(object sender, RoutedEventArgs e)
{
    appServiceConnection = new AppServiceConnection
    {
        // Замените AppServiceName и PackageFamilyName на значения
        // соответствующие сервису EmployeeLookupServiceApp.

        AppServiceName = "EmployeeLookup",
        PackageFamilyName =
            "3598a822-2b34-44cc-9a20-421137c7511f_4frctqp64dy5c"
    };
}
```

5. Сделайте обработчик **GetEmployeeById()** асинхронным, а также откроите соединение со службой приложения. На основе статуса соединения мы уведомим пользователя о наличии любых ошибок. Вы создадите метод **.LogError** на следующем этапе.

C#

```
private async void GetEmployeeById(object sender, RoutedEventArgs e)
{
    appServiceConnection = new AppServiceConnection
    {
        AppServiceName = "EmployeeLookup",
        PackageFamilyName =
            "3598a822-2b34-44cc-9a20-421137c7511f_4frctqp64dy5c"
    };

    var status = await appServiceConnection.OpenAsync();

    switch (status)
    {
        case AppServiceConnectionStatus.AppNotInstalled:
            await LogError("Приложение EmployeeLookup не установлено.
                            Установите и попробуйте снова.");
            return;
        case AppServiceConnectionStatus.AppServiceUnavailable:
            await LogError("Приложение EmployeeLookup не поддерживает
                            AppService");
            return;
        case AppServiceConnectionStatus.AppUnavailable:
            await LogError("Пакет для реализации AppService недоступен.
                            Вы не забыли добавить компонент WinRT в
                            качестве ссылки (reference) приложении
                            EmployeeLookupService?");
            return;
        case AppServiceConnectionStatus.Unknown:
            await LogError("Unknown Error.");
            return;
    }
}
```

6. Добавьте пространства имен **System.Threading.Tasks** и **Windows.UI.Popups** к коду **MainPage**.

C#

```
using System.Threading.Tasks;
using Windows.UI.Popups;
```

7. Добавьте метод **.LogError** в код **MainPage** для показа сообщений, выводимых вами на этапе 5. Настоящий метод откроет диалоговое окно с указанием статуса соединения со службой приложения в случае возникновения ошибки.

C#

```
private async Task LogError(string errorMessage)
```

```
{  
    await new MessageDialog(errorMessage).ShowAsync();  
}
```

8. В обработчике **GetEmployeeById()** выделите ID сотрудников из текста, который пользователь ввёл в элемент EmployeeId, созданный ранее на главной странице. Обратите внимание, что у этого текстового окна установлен атрибут **AcceptsReturn="True"**, что позволяет пользователю вводить несколько ID, по одному в каждой строке.

C#

```
case AppServiceConnectionStatus.Unknown:  
    await LogError("Unknown Error.");  
    return;  
}  
  
var items = this.EmployeeId.Text.Split(new string[] { Environment.NewLine  
,  
    StringSplitOptions.RemoveEmptyEntries);
```

9. Создайте сообщение, состоящее из ValueSet(), чтобы отправить его в службу приложения. Добавьте каждый отобранный ID в сообщение.

C#

```
var items = this.EmployeeId.Text.Split(new string[] { Environment.NewLine  
,  
    StringSplitOptions.RemoveEmptyEntries);  
  
var message = new ValueSet();  
  
for (int i = 0; i < items.Length; i++)  
{  
    message.Add(i.ToString(), items[i]);  
}
```

10. Отправьте сообщение в службу приложения и дождитесь ответа. Используйте конструкцию switch для обработки потенциального статуса ответа.

C#

```
for (int i = 0; i < items.Length; i++)  
{  
    message.Add(i.ToString(), items[i]);  
}  
  
var response = await appServiceConnection.SendMessageAsync(message);  
  
switch (response.Status)  
{  
    case AppServiceResponseStatus.ResourceLimitsExceeded:  
        await LogError("Недостаточно ресурсов. AppService был уничтожен.");  
        return;
```

```

        case AppServiceResponseStatus.Failure:
            await LogError("Ответ не получен.");
            return;
        case AppServiceResponseStatus.Unknown:
            await LogError("Неизвестная ошибка.");
            return;
    }
}

```

Задача 4 – Отобразите результаты и научитесь отлаживать фоновую задачу

Пользовательский интерфейс может принять входные данные, а ваша служба приложения готова к запросам. Последний шаг заключается в отображении результатов запроса и отладке фоновой задачи, если это необходимо. В рамках этой задачи вы добавите ListView в пользовательский интерфейс, чтобы отобразить один или несколько результатов запроса.

1. К **MainPage.xaml.cs** добавьте следующее пространство имен:
System.Collections.ObjectModel.

```
C#
using System.Collections.ObjectModel;
```

2. Создайте новую версию **ObservableCollection**, которая будет содержать данные о сотрудниках. Вы создали простой класс Employee в задаче 2 настоящего упражнения.

```
C#
public sealed partial class MainPage : Страница
{
    private AppServiceConnection appServiceConnection;
    public ObservableCollection<Employee> Items { get; set; } = new
        ObservableCollection<Employee>();
```

3. В обработчике **GetEmployeeById** добавьте каждый элемент ответа от службы приложения в коллекцию Items.

```
C#
switch (response.Status)
{
    case AppServiceResponseStatus.ResourceLimitsExceeded:
        await LogError("Недостаточно ресурсов. AppService уничтожен.");
        return;
    case AppServiceResponseStatus.Failure:
        await LogError("Не получили ответа от AppService.");
        return;
    case AppServiceResponseStatus.Unknown:
        await LogError("Неизвестная ошибка.");
        return;
}

foreach (var item in response.Message)
{
    this.Items.Add(new Employee
```

```

    {
        Id = item.Key,
        Name = item.Value.ToString()
    });
}

```

- Добавьте **ListView** к панели **StackPanel** в MainPage.xaml. ListView отобразит ID и ФИО сотрудника при проведении запроса.

XAML

```

<StackPanel>
    <TextBlock Text="Enter one employee ID per line." Margin="12" />
    <TextBox x:Name="EmployeeId" AcceptsReturn="True" Margin="12,0,12,12"/>
    <Button Content="Look up employee(s)" Click="GetEmployeeById"
        Margin="12,0,0,0"/>
    <ListView ItemsSource="{x:Bind Items}" Grid.Column="1">
        <ListView.ItemTemplate>
            <DataTemplate x:DataType="local:Employee">
                <StackPanel>
                    <StackPanel Orientation="Horizontal" Margin="12">
                        <TextBlock Text="{x:Bind Id}" Margin="0,0,12,0" />
                        <TextBlock Text="{x:Bind Name}" />
                    </StackPanel>
                </StackPanel>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</StackPanel>

```

Примечание: Мы используем x:Bind для того, чтобы задать привязку данных для Items внутри ListView.

- Создайте и запустите приложение **AppServices**. Введите ID сотрудника в текстовое окно и используйте кнопку для поиска сотрудников **Look up employee(s)**, чтобы запросить службу поиска сотрудников.
- Поздравляем, если ваша служба приложений отображает ФИО сотрудника при ответе на запрос! Результат будет подобен представленному ниже:

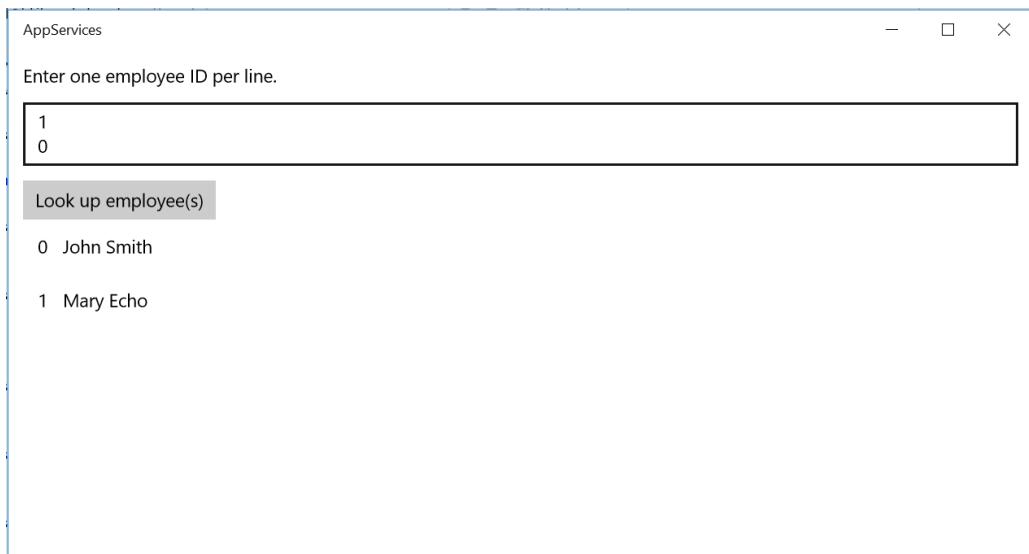


Рисунок 10

Результаты многострочного поиска сотрудников возвращаются службой приложения.

7. Остановите отладку и вернитесь в Visual Studio.
8. В случае возникновения ошибки при проведении поиска, давайте исправим имеющиеся ошибки. Если вы получаете ошибку статуса **AppUnavailable**, возможно, вы забыли добавить проект EmployeeLookupService.Background как ссылку на приложение EmployeeLookupService.

Также ошибка статуса **AppUnavailable** может указывать на то, что объявленная вами входная точка службы приложения не соответствует наименованию класса в классе **EmployeeLookup.cs**.

9. Для отладки самой фоновой задачи, откройте решение EmployeeLookupService.
10. Откройте свойства проекта **EmployeeLookupService** и перейдите во вкладку **Debug (Отладка)**. Отметьте пункт "**Do not launch, but debug my code when it starts**" («**Не запускать, но произвести отладку моего кода в качестве начального шага**»), а затем сохраните свойства файла.

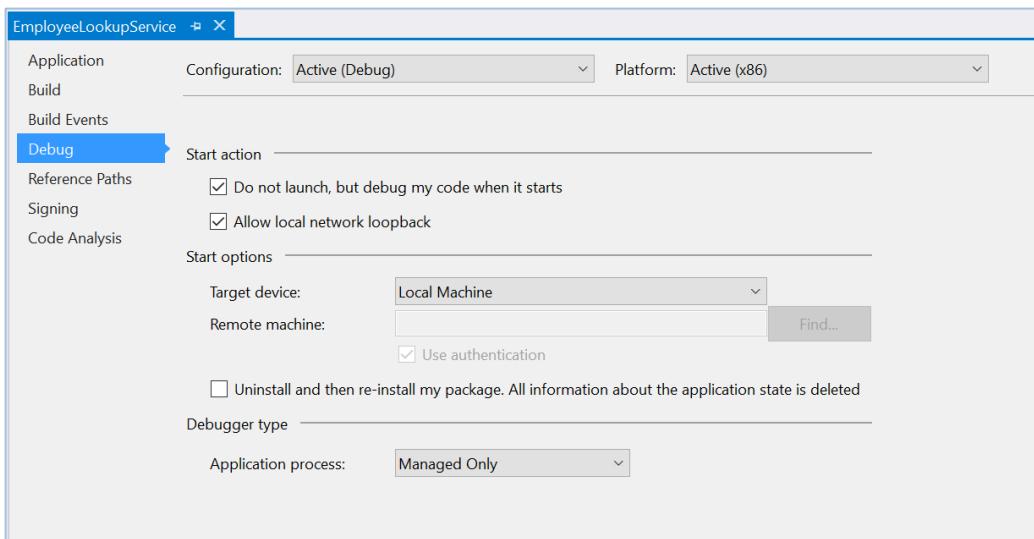


Рисунок 11

Настройте опции отладки таким образом, чтобы можно было произвести отладку фоновой задачи.

11. Задайте точку останова в своей фоновой задаче и используйте кнопку **Start Debugging** (**Начать отладку**), чтобы подключить отладчик. Приложение EmployeeLookupService не запускается.
12. Создайте и запустите приложение **AppServices**, и вновь запросите службу поиска. Вы наткнетесь на точку останова в своей фоновой задаче.

Краткий обзор

В рамках настоящего курса вы научились создавать и регистрировать службы приложений, а также асинхронно использовать их из других приложений.

Для более масштабных корпоративных приложений вам может понадобиться оформить свою службу приложения в виде SDK с последующим предоставлением доступа через библиотеку классов. Для этих целей лучше всего использовать систему поддержки версий, как в протоколе REST, чтобы обеспечить возможность постепенных обновлений и переходов на новую версию API.



Лабораторный практикум

Встраиваемая реклама

Октябрь 2015 года



Обзор

Пакет Universal Ad SDK прост в интегрировании, а также позволяет продвигать и монетизировать ваше приложение на различных рынках по всему миру.

В данном курсе вы научитесь устанавливать пакет Windows 10 Ad Mediator, который включает Microsoft Advertising SDK для XAML. Библиотеки Microsoft Advertising libraries для XAML/JavaScript представляют собой различные расширения из элемента управления AdMediator (Microsoft Advertising Universal SDK версии 1.0 в Справочном менеджере Visual Studio). Для ознакомления с дополнительной информацией посетите страницу:

[https://msdn.microsoft.com/en-us/library/mt313199\(v=msads.30\).aspx](https://msdn.microsoft.com/en-us/library/mt313199(v=msads.30).aspx).

Вы будете использовать Advertising SDK для реализации встроенной рекламы на основе демо-версий, предоставляемых Microsoft.

Цели

Настоящий курс научит вас:

- Устанавливать Windows 10 Ad Mediator
- Добавлять рекламные вставки в приложение
- Добавлять рекламный банер в контент приложения

Системные требования

Для завершения настоящего курса необходимы:

- Microsoft Windows 10
- Microsoft Visual Studio 2015

Настройка

Вам следует выполнить следующие действия для подготовки компьютера:

1. Установить Microsoft Windows 10.
2. Установить Microsoft Visual Studio 2015.
3. Установить Windows Ad Mediator.

Инструкция по установке Windows Ad Mediator представлена в Упражнении 1: Задача 2.

Расчетное время для завершения курса: **15-30 минут.**

Упражнение 1: Реализация рекламы в приложении

В рамках данного упражнения вы установите пакет Windows 10 Advertising SDK и будете использовать для добавления встроенной рекламы в свое приложение.

Задача 1 – Создать пустое приложение Universal Windows

Мы начнём с создания проекта на основе шаблона Blank App (Пустого приложения).

1. В новой версии Visual Studio 2015 выберите **File (Файл) -> New (Новый) -> Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed (Установленное) > Templates (Шаблоны) > Visual C# > Windows > Universal**, а затем выберите шаблон **Blank App приложения (Universal Windows)**.
2. Назовите свой проект "**Advertising**" и выберите местоположение, в которое будет осуществлено сохранение результатов прохождения Лабораторного практикума. На диске **C** создана папка под именем "**HOL**", информация о которой будет представлена в скриншотах.
3. Не изменяйте настройки, установленные для **Create new solution (Создания нового решения)** и **Create directory for solution (Создания папки для решения)**. Вы можете снять галочки как с **Add to source control (Добавить в систему контроля версий)**, так и **Show telemetry in the Windows Dev Center (Отобразить телеметрию в Windows Dev Center)**, если не хотите обновлять версию своей работы или использовать инструмент Application Insights. Нажмите **OK** для создания проекта.

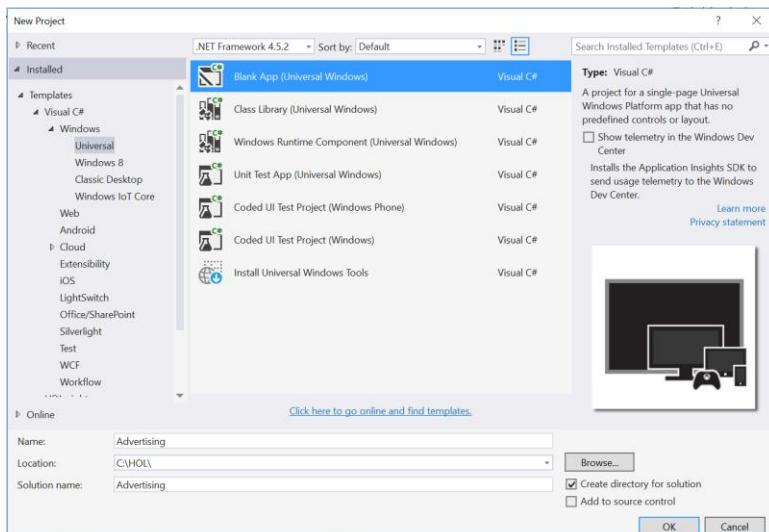


Рисунок 1

Создайте в Visual Studio 2015 новый проект Blank App.

- Настройте Solution Configuration (Текущую конфигурацию решения) на **Debug (Отладку)** и Solution Platform (Платформу решений) в соответствии с **x86**. Выберите **Local Machine (Локальный компьютер)** из выпадающего меню Debug Target (Цели отладки).

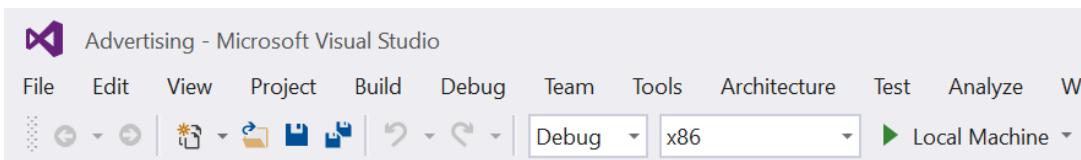


Рисунок 2

Сконфигурируйте свое приложение таким образом, чтобы оно запускалось на *Local Machine* (Локальном компьютере).

- Создайте и запустите свое приложение. Вы увидите окно Blank App со счетчиком частоты кадров, активированном по умолчанию для отладки.

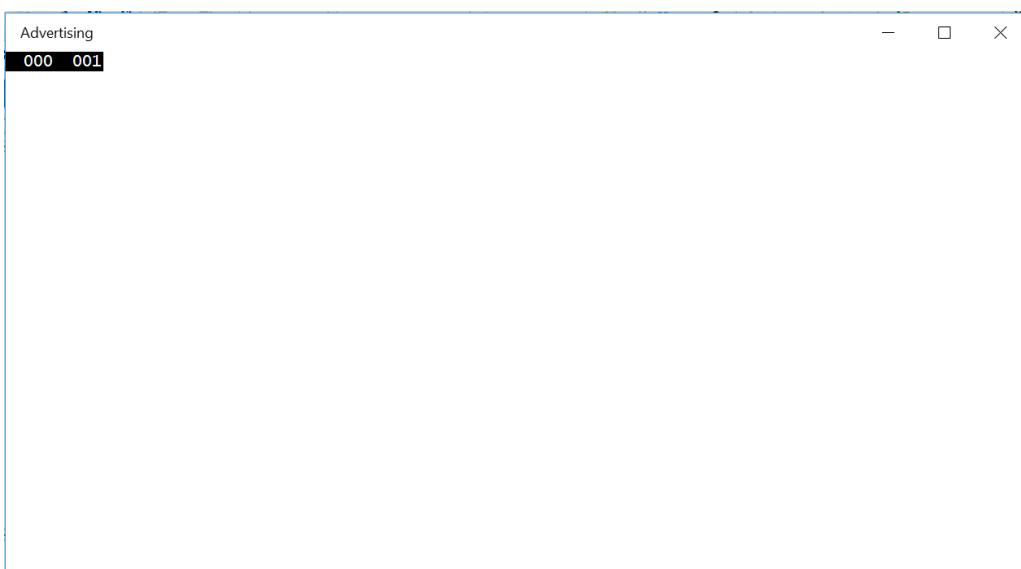


Рисунок 3

Приложение *Blank universal* выполняется в режиме Рабочего стола.

Примечание: Счетчик частоты кадров является инструментом, используемым в процессе отладки, который помогает следить за производительностью вашего приложения. Он полезен для тех приложений, которые требуют интенсивной графической обработки, однако не подходит для простых приложений, которые будут создаваться вами на данный момент.

В шаблоне Blank App директива препроцессора активирует или отключает счетчик частоты кадров внутри **App.xaml.cs**. Счетчик частоты кадров может перекрывать или скрывать контент вашего приложения, если не свернуть его. При выполнении данных работ вы можете отключить его, отметив **this.DebugSettings.EnableFrameRateCounter** как **False (Ложное)**.

Вернитесь в Visual Studio и остановите отладку.

Задача 2 – Установите пакет SDK для рекламы в Windows 10

Перед добавлением рекламы в ваше приложение, вам необходимо установить Windows Ad Mediator.

Примечание: Windows Ad Mediator включает пакет Microsoft Advertising SDK для XAML.

1. В Visual Studio откройте диалоговое окно Tools (Инструменты) > Extensions and Updates (Расширения и обновления).
2. Перейдите в раздел Online (Онлайн) в меню и используйте окно поиска для нахождения Ad Mediation.

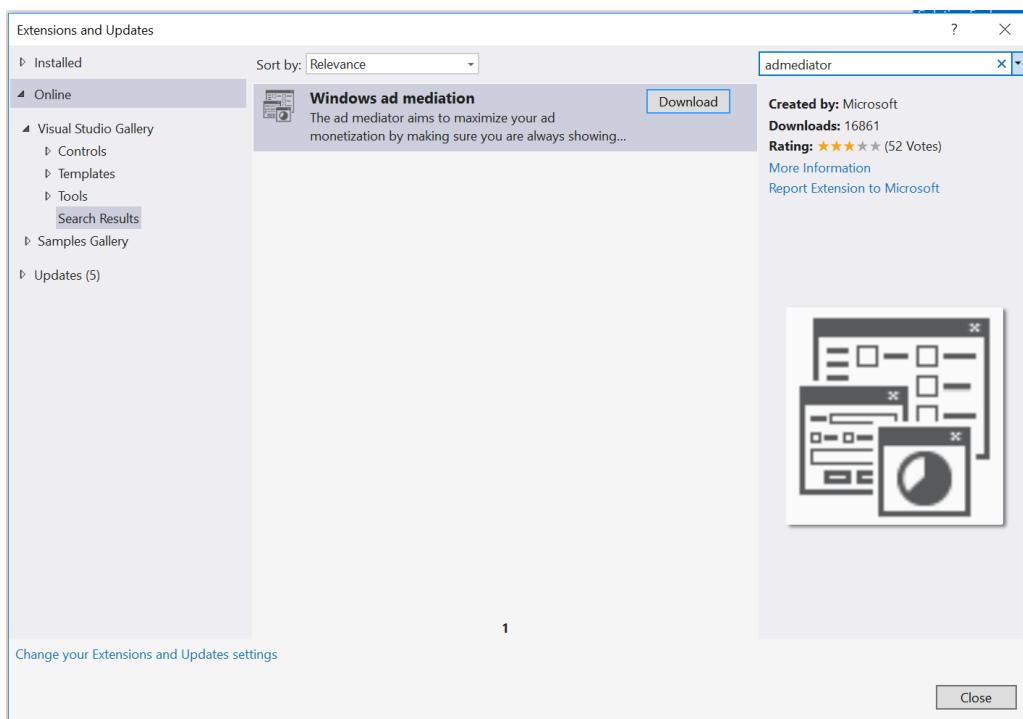


Рисунок 4

Расширение Windows Ad Mediation находится в диалоговом окне Extensions and Updates (Расширения и обновления).

3. Выберите расширение **Windows Ad Mediation**, а затем **Download (Загрузить)**. Ваш браузер начнет загрузку установочного файла.
4. Запустите установочный файл **msi**. При запуске настроек Windows Ad Mediator установите его, используя опции по умолчанию. Если всплывает окно системы User Account Control (UAC) выберите **Yes (Да)**, чтобы разрешить приложению установить программное обеспечение на ваш ПК. Когда установка будет завершена, используйте кнопку **Finish (Завершить)**, чтобы выйти из мастера настройки.



Рисунок 5

Мастер настройки Windows Ad Mediator

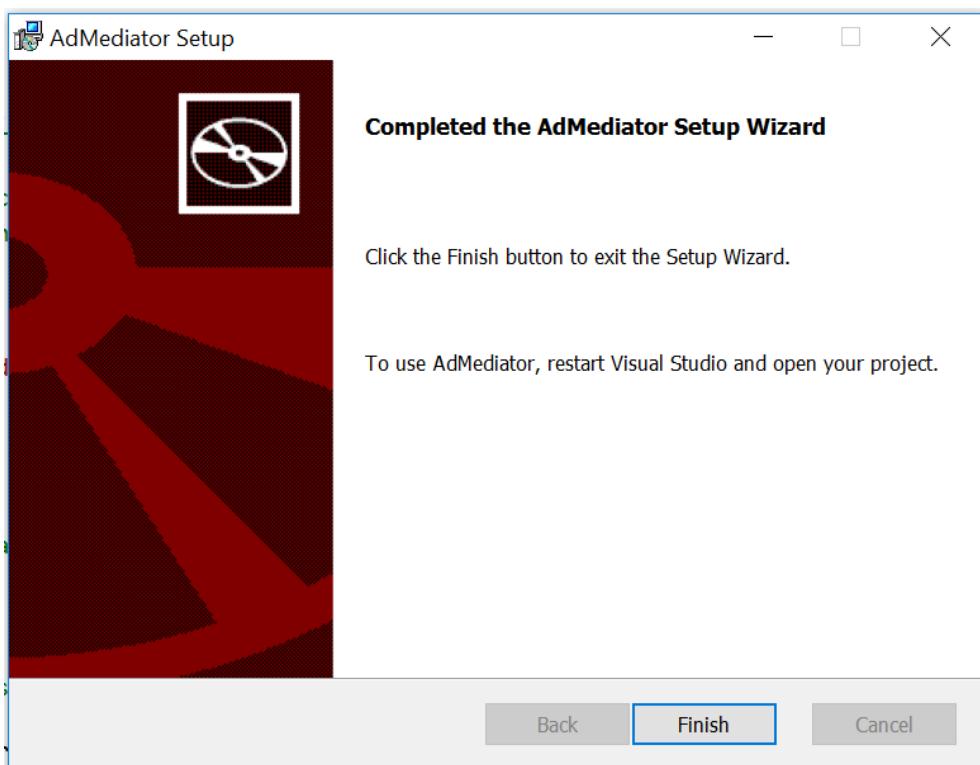


Рисунок 6

Завершите установку Windows Ad Mediator.

- Перезапустите Visual Studio, а также откройте проект Advertising, который вы создали при выполнении Задачи 1. Когда проект будет открыт, щелкните правой кнопкой мыши по References (Ссылки) в Solutions Explorer (Обозревателе решений) и выберите Add Reference (Добавить ссылку).

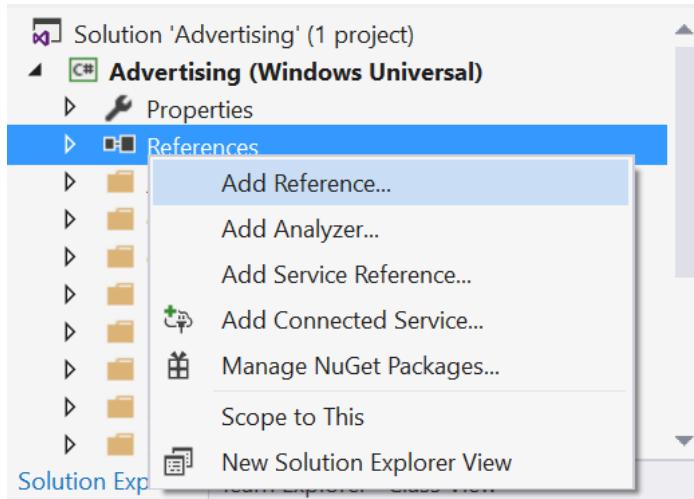


Рисунок 7

Перейдите в диалоговое окно Add Reference.

- Разверните раздел Universal Windows и выберите Extensions (Расширения). Вы увидите список пакетов SDK, применимых к вашему проекту. Отметьте галочкой Microsoft Advertising SDK для XAML, чтобы выбрать данный пакет и нажмите OK, чтобы добавить его в проект в качестве ссылки. Внимательно выбирайте корректный SDK!

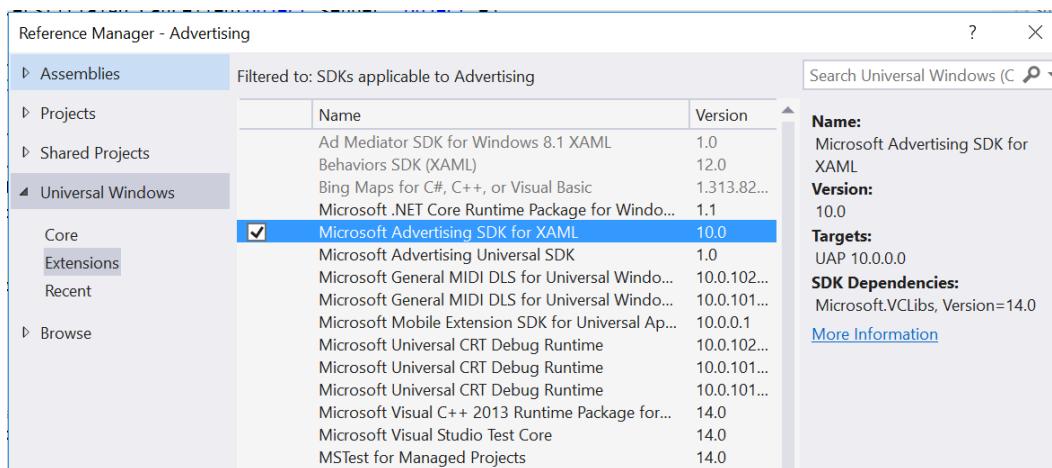


Рисунок 8

Добавьте пакет Microsoft Advertising SDK для XAML в качестве ссылки проекта.

- После закрытия диалогового окна Add Reference, вы сможете увидеть, что Microsoft Advertising SDK появился в списке ссылок проекта.

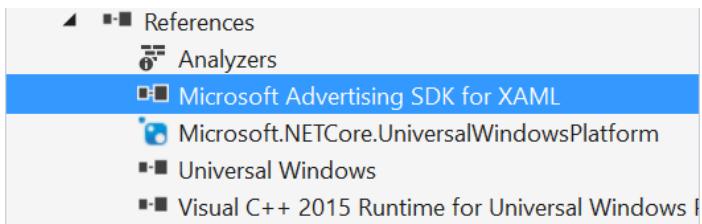


Рисунок 9

Добавьте пакет Microsoft Advertising SDK для XAML в качестве ссылки проекта.

Задача 3 – Добавление рекламной вставки в приложение

После добавления Microsoft Advertising SDK в виде ссылки к проекту Advertising, вы можете приступить к интеграции рекламы в свое приложение. Создайте новый класс, называемый DemoAds, который будет использовать тестовые файлы AppIds и AdUnits, предоставляемые Microsoft, для отображения рекламной вставки в вашем приложении.

1. Щелкните правой кнопкой мыши на наименование проекта в Solution Explorer (Обозреватель решений), а затем **Add (Добавить) > New Folder (Новая папка)**. Назовите папку "**Models**".
2. Чтобы создать новый класс **DemoAds**, щелкните правой кнопкой мыши на папку **Models (Модели)** и выберите **Add (Добавить) > New Item (Новый элемент)**. При появлении диалогового окна **Add New Item (Добавить новый элемент)** выберите класс **Visual C#** в качестве нового элемента (Рисунок 17). Назовите его **DemoAds.cs**.

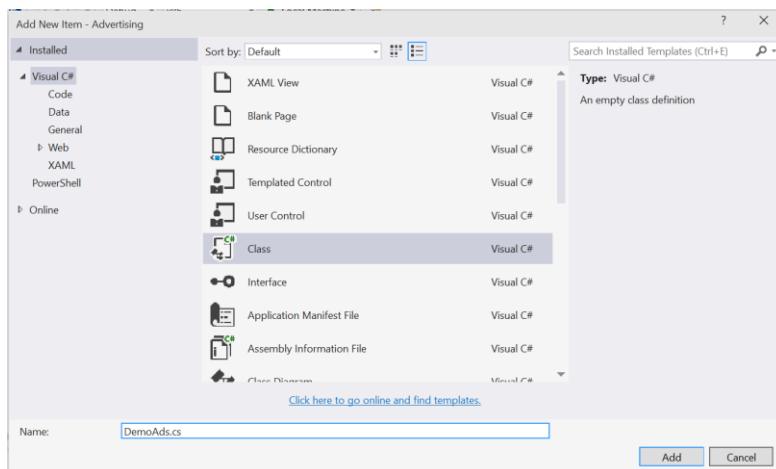


Рисунок 10

Создайте новый класс Visual C#, называемый "DemoAds".

3. Откройте **DemoAds.cs**. На данном этапе вы замените определение пустых классов рабочими классами: DemoAds и AdUnit. Код ниже отражает начальный шаблон класса, предоставляемый Visual Studio.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Advertising.Models
{
    class DemoAds
    {
    }
}
```

4. Добавьте код, выделенный красным, в файл DemoAds.cs и сохраните.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Advertising.Models
{

    /*
        These demo ad values are drawn from: https://msdn.microsoft.com/en-US/library/mt125365(v=msads.100).aspx
    */
    public static class DemoAds
    {
        public static Dictionary<string, AdUnit> ImageAdUnits { get; private set; }
        public static AdUnit VideoAdUnit { get; private set; }

        static DemoAds()
        {
            ImageAdUnits = new Dictionary<string, AdUnit>();

            ImageAdUnits.Add("300 x 50",
                new AdUnit { Size = "300 x 50", AdUnitId = "10865275", AppId = "3f83fe91-d6be-434d-a0ae-7351c5a997f1" });
            ImageAdUnits.Add("320 x 50",
                new AdUnit { Size = "320 x 50", AdUnitId = "10865270", AppId = "3f83fe91-d6be-434d-a0ae-7351c5a997f1" });
            ImageAdUnits.Add("300 x 250",
                new AdUnit { Size = "300 x 250", AdUnitId = "10043121", AppId = "d25517cb-12d4-4699-8bdc-52040c712cab" });
            ImageAdUnits.Add("300 x 600",

```

```

        new AdUnit { Size = "300 x 600", AdUnitId = "10043122",
    AppId = "d25517cb-12d4-4699-8bdc-52040c712cab" });
        ImageAdUnits.Add("480 x 80",
            new AdUnit { Size = "480 x 80", AdUnitId = "10865272", AppId
= "3f83fe91-d6be-434d-a0ae-7351c5a997f1" });
        ImageAdUnits.Add("640 x 100",
            new AdUnit { Size = "640 x 100", AdUnitId = "10865273",
    AppId = "3f83fe91-d6be-434d-a0ae-7351c5a997f1" });
        ImageAdUnits.Add("728 x 90",
            new AdUnit { Size = "728 x 90", AdUnitId = "10043123", AppId
= "d25517cb-12d4-4699-8bdc-52040c712cab" });

        VideoAdUnit = new AdUnit { Size = "Video", AdUnitId =
"11389925", AppId = "d25517cb-12d4-4699-8bdc-52040c712cab" };
    }
}

public class AdUnit
{
    public string Size { get; set; }
    public string AdUnitId { get; set; }
    public string AppId { get; set; }
}
}

```

Примечание: Настоящий класс ссылается на набор демо-рекламы, созданной и предоставляемой Microsoft в целях тестирования «живой рекламы» в ваших приложениях. Настоящая реклама была взята со страницы: [https://msdn.microsoft.com/en-US/library/mt125365\(v=msads.100\).aspx](https://msdn.microsoft.com/en-US/library/mt125365(v=msads.100).aspx)

5. Откройте **MainPage.xaml.cs**. Добавьте свойства для **ShowAds** и **ViewedFullInterstitial**. Для начала сделаем так, чтобы пользователи посмотрели рекламу целиком при запуске приложения

C#

```

public sealed partial class MainPage: Страница
{
    bool _showAds = true;
    public bool ShowAds
    {
        get { return _showAds; }
        set { _showAds = value; }
    }

    bool _viewedFullInterstitial = true;
    public bool ViewedFullInterstitial
    {
        get { return _viewedFullInterstitial; }
        set { _viewedFullInterstitial = value; }
    }
}

```

6. Добавьте пространства имен **Microsoft.Advertising.WinRT.UI** и **Advertising.Models** для ссылки на DemoAds и Microsoft Advertising UI.

C#

```
using Microsoft.Advertising.WinRT.UI;
using Advertising.Models;

namespace Advertising
{
```

7. После определения классов добавьте приватное поле InterstitialAd.

C#

```
namespace Advertising
{
    public sealed partial class MainPage : Страница
    {
        private InterstitialAd _interstitialAd;

        bool _showAds = true;
```

8. В конструкторе страницы привяжитесь к событиям AdReady (Реклама готова), Cancelled (Отмененное), Completed (Завершенное), and ErrorOccurred (Обнаружена ошибка) и инициализируйте рекламу.

C#

```
public MainPage()
{
    this.InitializeComponent();

    if (ShowAds)
    {
        // initialize the interstitial class
        _interstitialAd = new InterstitialAd();

        // wire up all 4 events
        _interstitialAd.AdReady += interstitialAd_AdReady;
        _interstitialAd.Cancelled += interstitialAd_Cancelled;
        _interstitialAd.Completed += interstitialAd_Completed;
        _interstitialAd.ErrorOccurred +=
interstitialAd_ErrorOccurred;

        RequestAd();
    }
    else
    {
        // start normally
    }
}
```

9. Создайте обработчики событий AdReady, Cancelled, Completed и ErrorOccurred после конструктора.

```
C#
        RequestAd();
    }
    else
    {
        // start normally
    }
}

private void interstitialAd_ErrorOccurred(object sender,
AdEventArgs e)
{
    // handle errors here
}
private void interstitialAd_Completed(object sender, object e)
{
    // raised when the user has watched the full video
}
private void interstitialAd_Cancelled(object sender, object e)
{
    // raised if the user interrupts the video
}
private void interstitialAd_AdReady(object sender, object e)
{
    // raised when an ad is ready to show
}
```

10. Добавьте код в метод **RequestAd** под конструктором. Метод будет запрашивать видео рекламу из набора демо данных, который мы определили в классе **DemoAds**.

```
C#
        RequestAd();
    }
    else
    {
        // start normally
    }
}

private void RequestAd()
{
    _interstitialAd.RequestAd(AdType.Video,
DemoAds.VideoAdUnit.AppId, DemoAds.VideoAdUnit.AdUnitId);
}

private void interstitialAd_ErrorOccurred(object sender,
AdEventArgs e) {
```

Примечание: В целях демонстрации мы покажем рекламную вставку, как только реклама будет готова, посредством ее добавления в обработчик событий AdReady.

11. Добавьте обработчик для события **ErrorOccurred**. Не забудьте добавить **async** в обработчик событий для ожидания завершения **await**.

C#

```
private async void interstitialAd_ErrorOccurred(object sender,  
AdErrorEventArgs e)  
{  
    // handle errors here  
    var dialog = new ContentDialog  
    {  
        Title = "An Error",  
        Content = e.ErrorMessage,  
        PrimaryButtonText = "OK",  
        IsPrimaryButtonEnabled = true  
    };  
  
    await dialog.ShowAsync();  
}
```

Примечание: В настоящих приложениях может потребоваться более сложный обработчик событий.

12. Добавьте обработчик для события **AdReady**.

C#

```
private void interstitialAd_AdReady(object sender, object e)  
{  
    //raised when an ad is ready to show  
  
    if (_interstitialAd.State == InterstitialAdState.Ready)  
    {  
        _interstitialAd.Show();  
    }  
}
```

13. Создайте и запустите приложение. Рекламная видео вставка будет воспроизводиться при загрузке приложения.

Задача 4 – Запросите рекламу

Давайте сделаем так, чтобы пользователь просмотрел рекламу полностью до продолжения пользования приложением.

1. Мы используем **Cancelled event** посредством отображения сообщения, направленного пользователю, и перезапуска рекламы.

C#

```
private void interstitialAd_Completed(object sender, object e)
{
    // raised when the user has watched the full video
    ViewedFullInterstitial = true;

}

private async void interstitialAd_Cancelled(object sender, object e)
{
    // raised if the user interrupts the video
    var dialog = new ContentDialog
    {
        Title = "Ad Interrupted",
        Content = "You must watch the complete ad!",
        PrimaryButtonText = "OK",
        IsPrimaryButtonEnabled = true
    };

    await dialog.ShowAsync();

    RequestAd();
}
```

Примечание: Поскольку мы собираемся использовать элемент управления ContentDialog для отображения сообщения об ошибке посредством асинхронного вызова, необходимо добавить **async** в определение метода обработчика событий interstitialAd_Cancelled. ContentDialog является новым элементом управления в Windows 10, который облегчает отображение разнообразного контента через модальное диалоговое окно приложения. Для ознакомления с дополнительной информацией о ContentDialogs посетите страницу:

<https://msdn.microsoft.com/library/windows/apps/windows.ui.xaml.controls.contentdialog.aspx>

2. Постройте и запустите приложение. Щелкните на запущенное видео для отображения кнопки Back (Назад). Нажмите кнопку Back во время просмотра рекламы чтобы посмотреть выполнение метода interstitialAd_Cancelled.
3. Отключите отладку и вернитесь в Visual Studio.

Было бы неудобно не иметь возможности пропустить рекламу во время выполнения данных упражнений. Чтобы предоставить возможность пропуска рекламы, закомментируйте обработчик события.

C#

```
private async void interstitialAd_Cancelled(object sender, object e)
{
    // raised if the user interrupts the video
    //var dialog = new ContentDialog
    //{
    //    Title = "Ad Interrupted",
    //    Content = "You must watch the complete ad!",
```

```

        // PrimaryButtonText = "OK",
        // IsPrimaryButtonEnabled = true
        //};

        //await dialog.ShowAsync();

        //RequestAd();
    }
}

```

4. Создайте и повторно запустите свое приложение. Убедитесь, что вы можете пропустить рекламу, выбрав кнопку Back во время ее воспроизведения.

5. Отключите отладку и вернитесь в Visual Studio.

Задача 5 – Отобразите рекламный банер

В рамках этой задачи вы будете использовать Microsoft Advertising SDK для отображения встроенной рекламы сбоку от контента вашего приложения. Схожие приемы могут быть использованы, чтобы разместить рекламу, встроенную в другие элементы управления.

1. Откройте **MainPage.xaml**. Добавьте пространство имен Microsoft.Advertising.WinRT.UI.

XAML

```
xmlns:UI="using:Microsoft.Advertising.WinRT.UI"
```

2. Чтобы создать контейнер для вашей встроенной рекламы, замените элементы **Grid** в **MainPage.xaml** на **Hub**. Добавьте hub-раздел **Advertising (Реклама)** и соседний раздел **Content (Контент)** в концентратор (hub). Hub-раздел Advertising использует **AdControl** из Advertising SDK, чтобы отобразить статичный рекламный банер слева от контента приложения. В окне дизайнера интерфейса, данный hub-раздел отобразит синий прямоугольник с теми же размерами, что и сама реклама. Установите параметр **Visibility** в значение **Collapsed**.

XAML

```
<Hub Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <HubSection
        VerticalContentAlignment="Stretch"
        x:Name="AdvertisingSection"
        Header="Advertising"
        Visibility="Collapsed">
        <DataTemplate>
            <Grid VerticalAlignment="Top">
                <!-- The rectangle acts as a place holder so we can see
where the ad control is located-->
                <Rectangle Fill="Blue" Width="300" Height="600"/>
                <UI:AdControl
                    ApplicationId="d25517cb-12d4-4699-8bdc-
52040c712cab"
                    AdUnitId="10043122">

```

```
        Height="600"
        VerticalAlignment="Top"
        Width="300"/>
    </Grid>
</DataTemplate>
</HubSection>
<HubSection Header="Content" />
</Hub>
```

3. Во вспомогательном коде MainPage добавьте метод ShowInlineAds() для отображения AdvertisingSection, если **ShowAds** – True.

C#

```
private void ShowInlineAds()
{
    if (ShowAds)
    {
        AdvertisingSection.Visibility = Visibility.Visible;
    }
    else
    {
        AdvertisingSection.Visibility = Visibility.Collapsed;
    }
}
```

4. Вызовите метод **ShowInlineAds()** после **RequestAd()** в конструкторе.

C#

```
RequestAd();
ShowInlineAds();
}
```

5. Постройте и запустите свое приложение. После того окончания воспроизведения рекламной вставки вы увидите, что рекламный банер появится слева от контента вашего приложения.

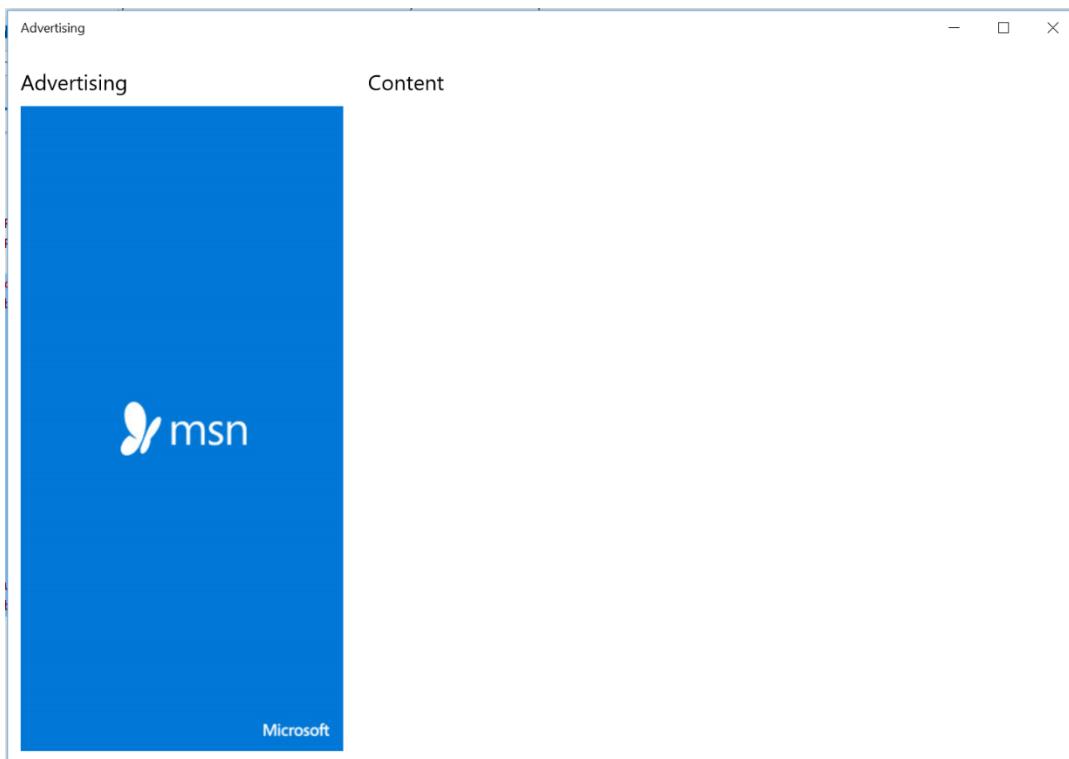


Рисунок 11

Встроенная реклама в приложении.

6. Завершите отладку и вернитесь в Visual Studio.

Краткий обзор

В рамках настоящего курса вы научились загружать и устанавливать Advertising SDK, и отображать новые рекламные видео вставки. Теперь вы можете делать так, чтобы пользователи в обязательном порядке просматривали вашу рекламу, а также добавлять рекламу на основе изображений в элемент управления Hub.