Windows 10

Лабораторный практикум

Службы приложений

Октябрь 2015 года

Microsoft

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 заверешения 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.

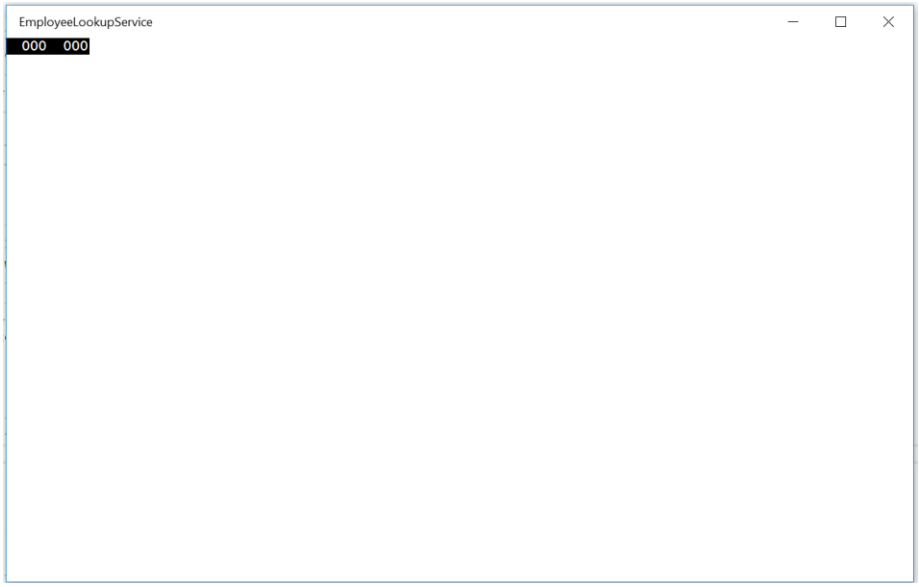

Рисунок 1

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"**.

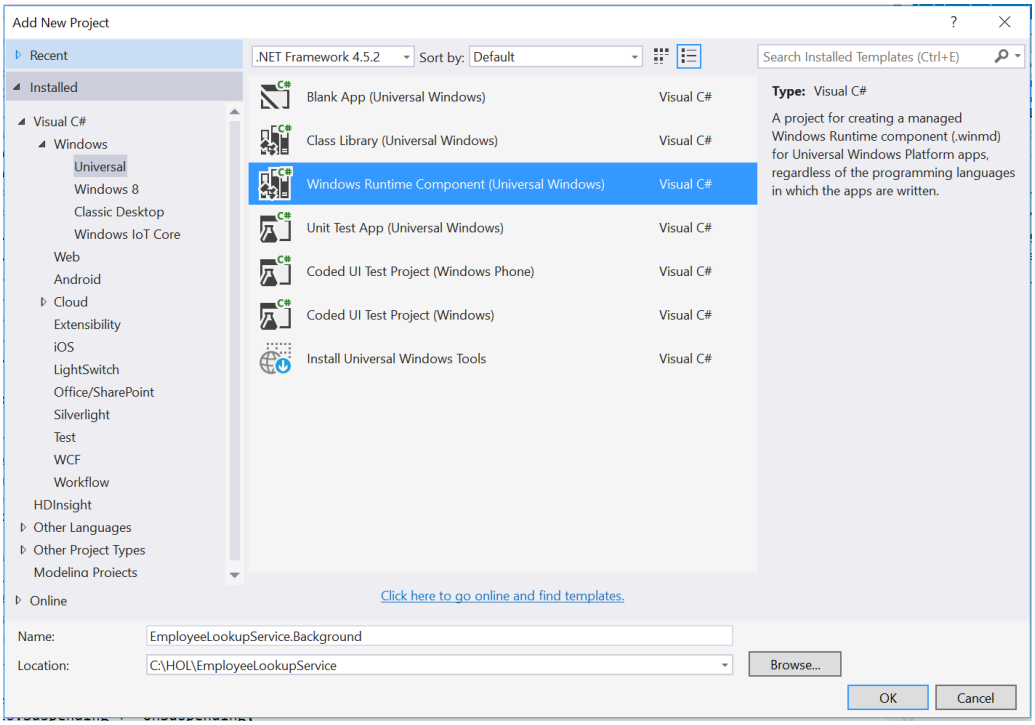

Рисунок 2

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)**.

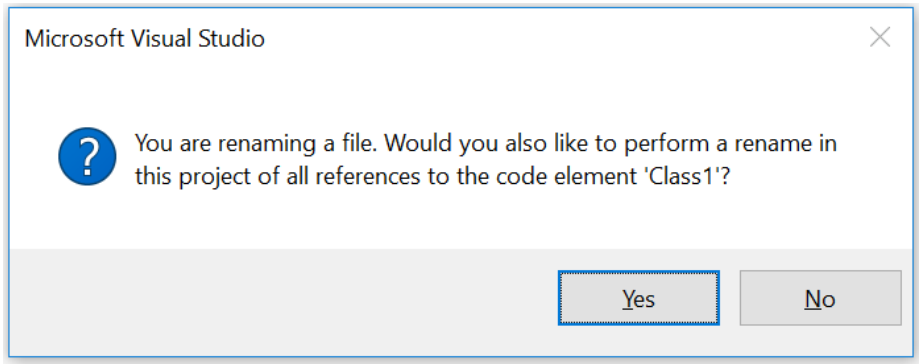

Рисунок 3

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()]);
```

```
        }
    }

    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.

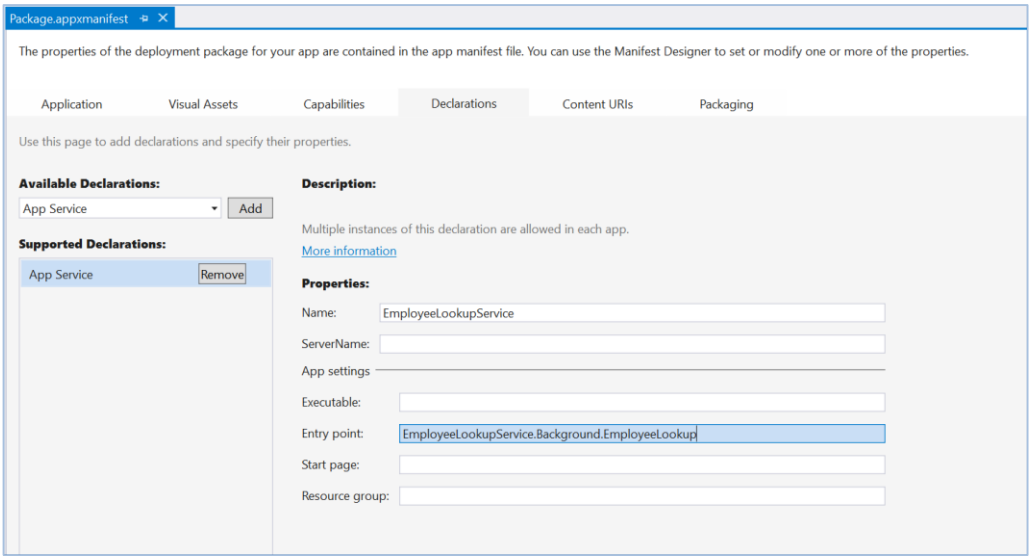

Рисунок 4

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**.

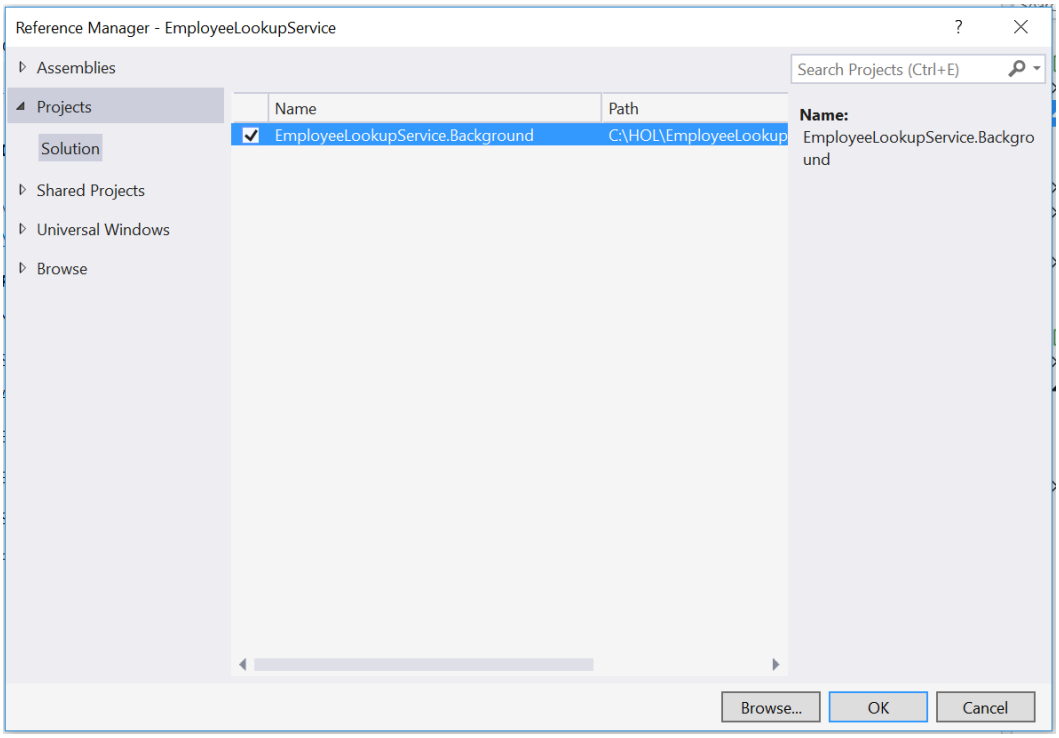

Рисунок 5

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 будет 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.

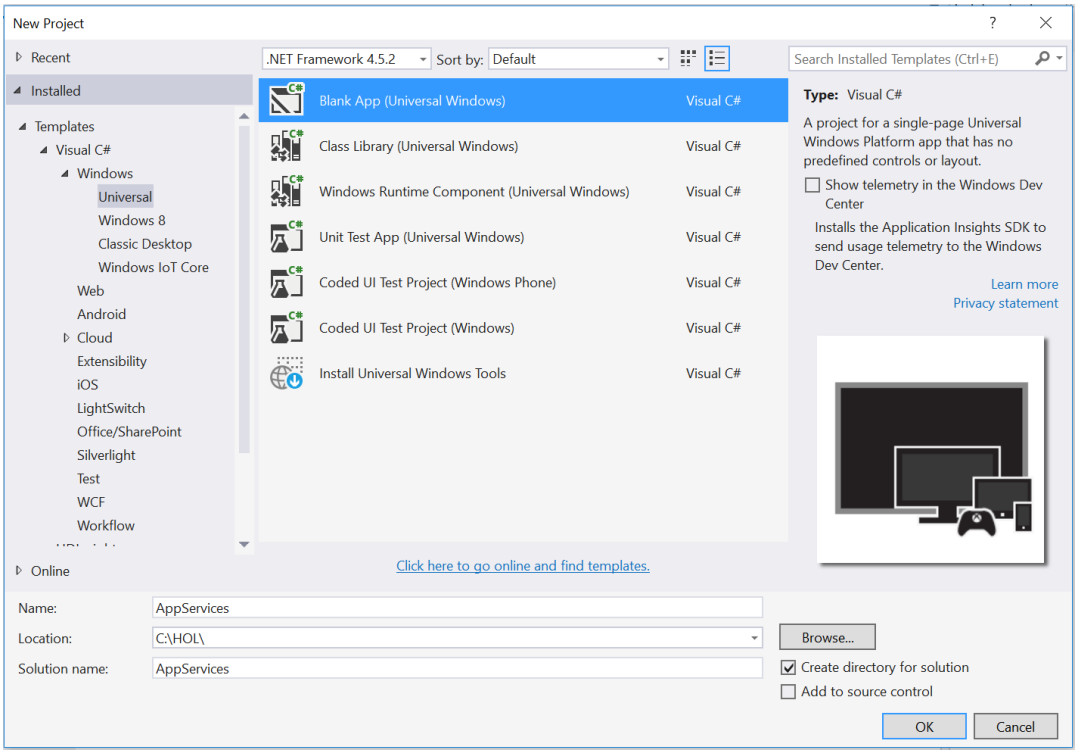

Рисунок 6

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).

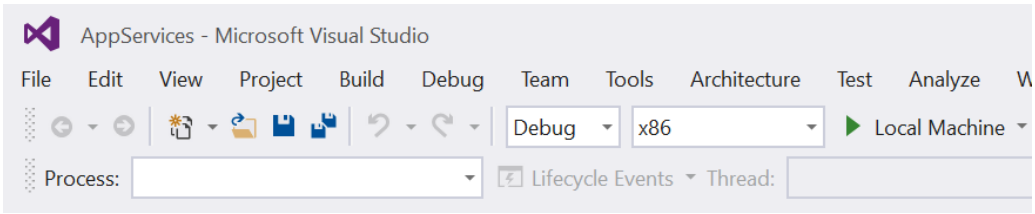

Рисунок 7

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.

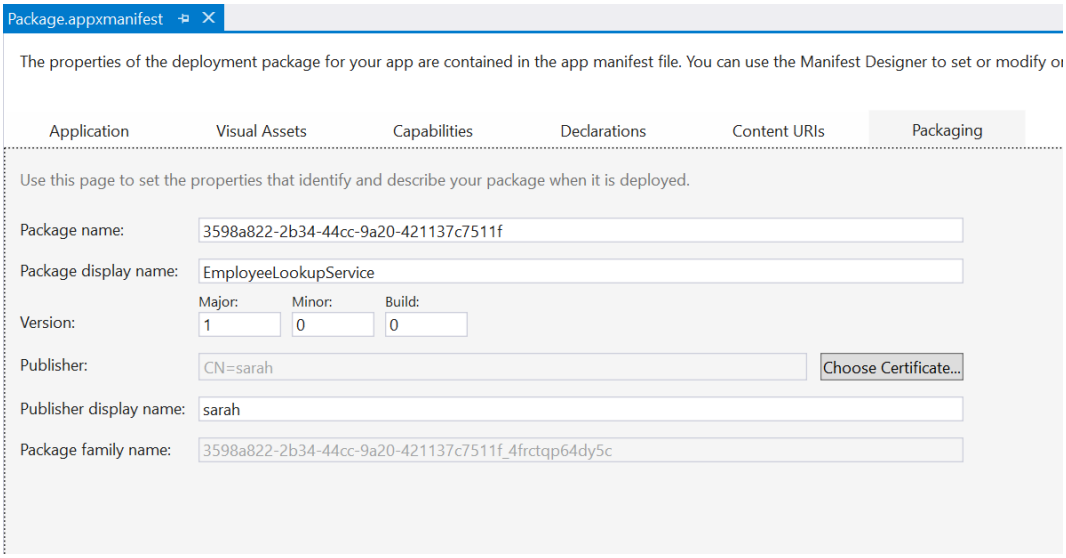

Рисунок 9

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 откройте 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 дожидайтесь 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()
    });
}
```

4. 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.

5. 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.

6. Поздравляем, если ваша служба приложений отображает ФИО сотрудника при ответе на запрос! Результат будет подобен представленному ниже:

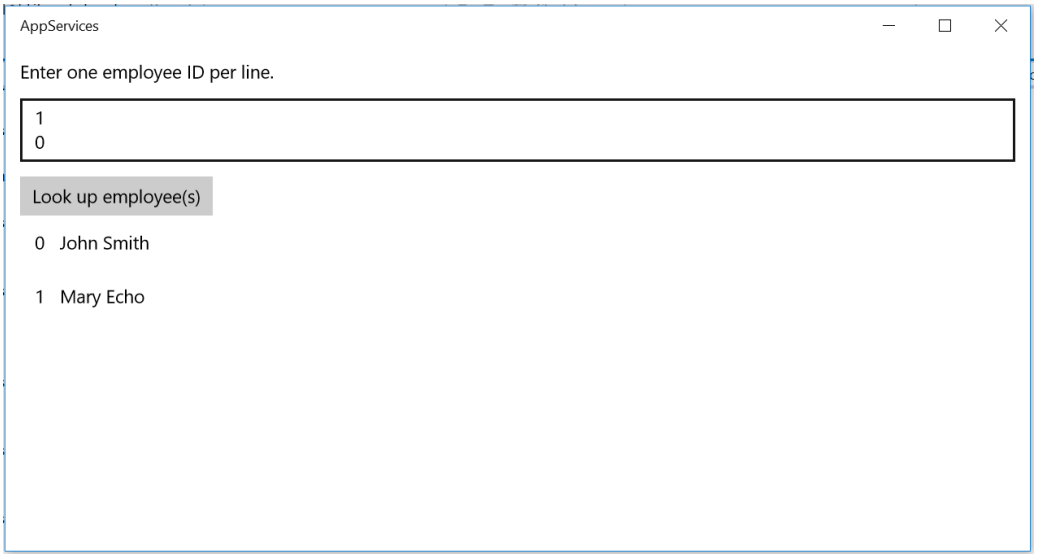

Рисунок 10

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.

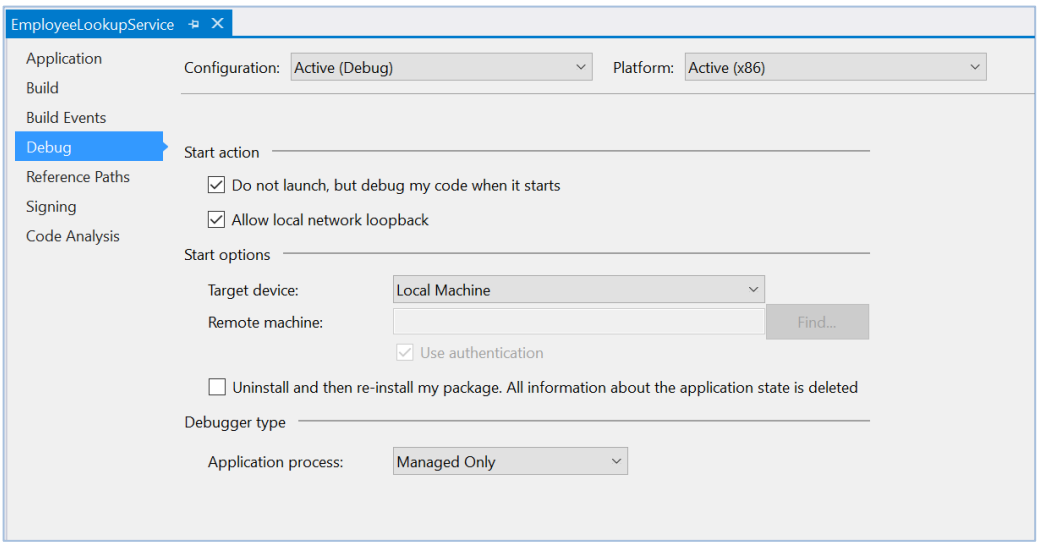

Рисунок 11

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.