Министерство образования и науки республики Казахстан

Костанайский государственный университет имени А.Байтурсынова

Кафедра программного обеспечения

Т.Г.Бортко, В.Б. Синицина

**Облачное программирование**

Учебно- методическое пособие

(часть 2)

Костанай, 2013

**ББК85.63**

**Б 65**

**Рецензенты**

Уаисова Майра Маликовна–зам. декана инженерного факультета КИПУ имени М. Дулатова

Исмаилов Арман Оразалинович - кандидат физико-математических наук, доцент кафедры программное обеспечение

Иванова Ирина Владимировна - кандидат педагогических наук, ст. преподаватель кафедра программного обеспечения КГУ

**Составители:**

Бортко Татьяна Георгиевна, доктор педагогических наук, профессор кафедры программного обеспечения

Синицина Виктория БРисуноковна, ст.преподаватель кафедры программного обеспечения

Б65 Бортко Т.Г.

Облачное программирование. Учебно-методическое пособие (часть 2) – Костанай, 2013. 92 с.

В учебно-методическом пособии описываются различные облачные решения, их возможности, преимущества и Рисунокки, рассматриваются Azure Services Platform: Windows Azure, Windows .Net Services, широко освещен вопрос Windows Azure Queue: технология, программирование, модели данных. Вторая часть пособия раскрывает практические вопросы облачного программирования.

Предназначено для магистрантов технических специальностей; может быть рекомендовано преподавателям высших учебных заведений, изучающих вопросы облачных технологий.

ББК 85.63

Утверждено и рекомендовано к изданию Учебно-методическим советом Костанайского государственного университета имени А.Байтурсынова,

­­­\_\_ \_\_\_\_\_\_2013г., протокол №\_\_

© Бортко Т.Г., 2013

Содержание

|  |  |
| --- | --- |
| Практическая работа №1…………………………………………………… | 4 |
| Практическая работа №2. Создание первого проекта……………………. | 12 |
| Практическая работа №3. Настройка хранилища разработки в Visual Studio 2010…………………………………………………………………… | 17 |
| Практическая работа №4. Хранилище данных с реляционной структурой…………………………………………………………………… | 26 |
| Практическая работа №5. Работа с Windows Azure Table……………… | 32 |
| Практическая работа №6. Работа с Windows Azure Blob ……………… | 54 |
| Практическая работ №7. Работа с Windows AzureQueue ……………… | 74 |
| Практическая работа №8. Mmicrosoft .Net Service Bus: обзор, обмен сообщениями, управление доступом……………………………………… | 88 |

**Практическая работа №1**

Загрузить инструменты для работы можно по следующей ссылке: <http://www.microsoft.com/downloads/ru-ru/details.aspx?FamilyID=7a1089b6-4050-4307-86c4-9dadaa5ed018#QuickDetails>

**Требования к установке Windows Azure SDK**

Операционная система.

Windows Azure SDK 1.3 от ноября 2010 года официально поддерживается следующими операционными системами:

* Windows 7 (в т.ч. Enterprise; Home Premiumx86, x64; Professional x86, x64; Ultimate x86, x64)
* Windows Server 2008 + SP2, Windows Server 2008 R2
* Windows Vista + SP2

Программное обеспечение

**для Windows Azure SDK**:

1. .Net Framework 3.5 + SP1
2. IIS 7.0
3. Microsoft SQL Server 2005 Express
4. Microsoft SQL Server 2008 (в т.ч. Express)
5. Исправление WCF <http://support.microsoft.com/kb/971842> (при отсутствии SP1 для Windows 7 и Windows Server 2008 R2 - <http://www.microsoft.com/downloads/ru-ru/details.aspx?FamilyID=c3202ce6-4056-4059-8a1b-3a9b77cdfdda>)
6. Исправление FastCGI (не для Windows 7) <http://support.microsoft.com/kb/967131>
7. для Visual Studio:
8. для 32битных ОС - исправление Windows Aure IntelliTrace <http://archive.msdn.microsoft.com/KB983301>
9. Windows PowerShell (не для Windows 7. <http://support.microsoft.com/kb/968929>)
10. Модуль URL Rewrite <http://www.iis.net/download/URLRewrite>

Примечание: в материалах данной практической работе будет рассмотрено формирование рабочего места Visual Studio 2010. Установка и настройка компонентов для формирования рабочего места на основе Visual Web Developer хорошо описана здесь: <http://blogs.technet.com/b/isv_team/archive/2010/12/27/3377315.aspx>

Убедитесь, что все вышеперечисленные требования соблюдены. После этого можно пРисуноктупать к непосредственной установке инструментов для работы с Visual Studio (в данном курсе мы рассматриваем версию 2010).

**Подготовка рабочего места**:

1. Установите VS 2010 и MS SQL Server 2008. Установка данного инструментария подробно описывается во множестве ресурсов и, как правило, не вызывает затруднений. (см. п№1-3 списка вспомогательных материалов)

2. Настройка IIS

для Windows 7

Откройте панель управления (Пуск - Панель управления)

Выберите "Программы и компоненты"

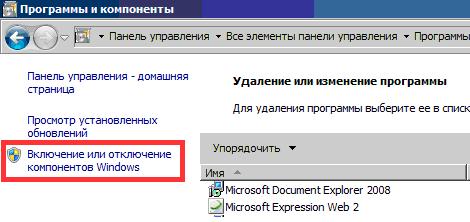


Рисунок 1.1 - Программы и компоненты

Раскройте узел "Microsoft .Ner Framework 3.5" и включите элемент "Windows Communication Foundation HTTP Activation"

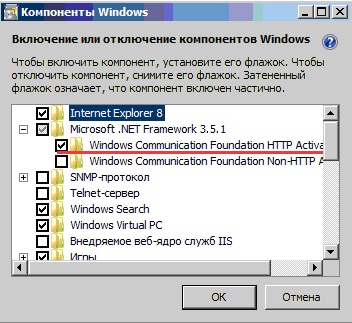


Рисунок 1.2 - Компоненты Windows

Последовательно раскройте узлы "Службы IIS", "Службы Интернета" и "Компоненты разработки приложений", отметьте элементы "ASP.NET" и "CGI".

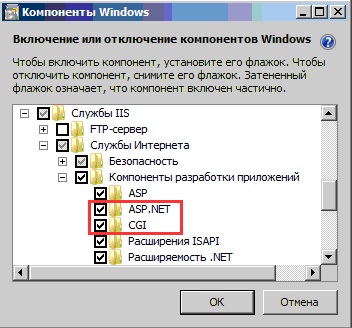


Рисунок 1.3 - Компоненты Windows

В узле "Службы IIS" разверните "Службы Интернета" и "Общие функции HTTP". Отметьте элемент "Статическое содержимое".

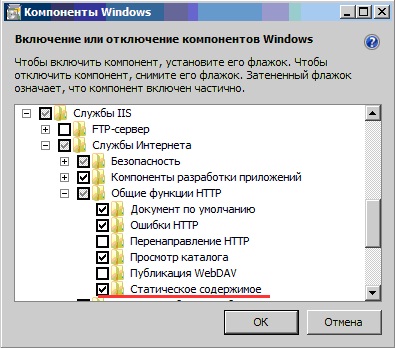


Рисунок 1.4 - Компоненты Windows

В узле "Службы IIS" разверните "Средства управления веб-сайтом" и отметьте "Консоль управления IIS".

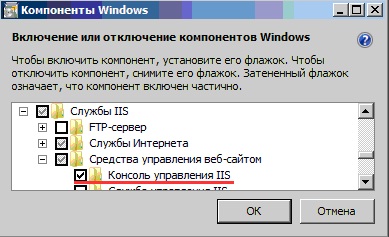


Рисунок 1.5 - Компоненты Windows

После этого нажмите "OK" и дождитесь завершения процесса установки отмеченных компонент.

Для Windows 2008

Запустите диспетчер сервера (Пуск - Администрирование - Диспетчер сервера)

Щелкните правой кнопкой мыши на узле "Компоненты" и выберите "Добавить компоненты"

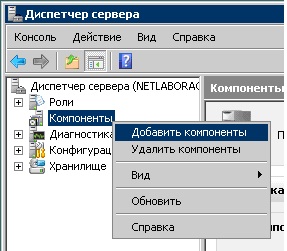


Рисунок 1.6 - Диспетчер сервера

В списке компонентов, в узле "Возможности .Net Framework 3.0" отметьте ".Net Framework 3.0".

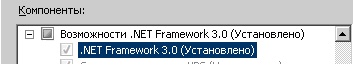


Рисунок 1.7 - Компоненты

Аналогичным образом выберите элемент "Активация HTTP" в узле "Активация WCF". И нажмите "Далее". В случае, если появится диалоговое окно установки служб для данных компонент, установите их.

Дождитесь окончания установки и перейдите к узлу "Роли" диспетчера сервера.

Установите роль "Веб-сервер", нажав "Добавить роли"

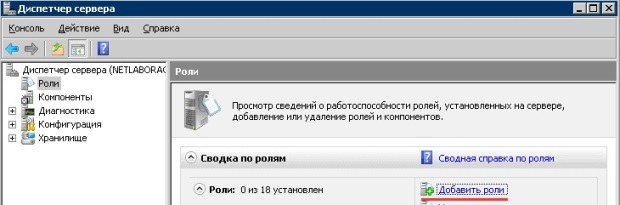


Рисунок 1.8 - Диспетчер сервера

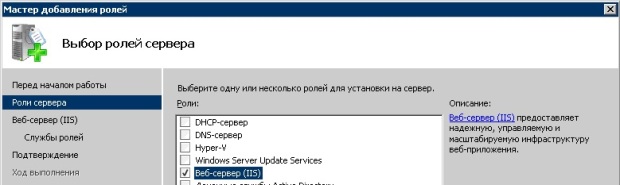


Рисунок 1.9 - Мастер добавления ролей

Последовательно нажимая Далее" установите необходимую роль.

В узле "Роли" выберите элемент "Веб - сервер(IIS) и нажмите "Добавить службы ролей"

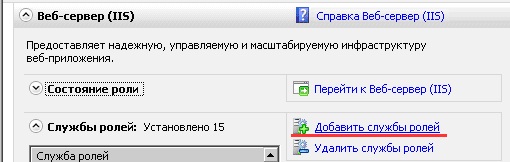


Рисунок 1.10 - Веб-сервер

Выделите службы "Статическое содержимое", "ASP.NET" и "CGI" в узлах "Веб-сервер" и "Разработка приложений" соответственно, если они еще не установлены. Нажмите "Далее", затем "Установить".

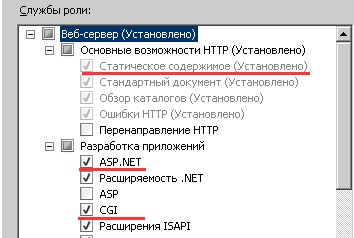


Рисунок 1.11 - Службы роли

Дождитесь завершения процесса установки.

3. Установите VSCloudService.exe или Windows Azure SDK 1.3. Отметим, что SDK входит в состав VSCloudService.exe.

4. В случае, если у вас 32битная ОС установите исправление №8 списка требуемого ПО.

5. Установите исправления №№6-7 из списка требуемого программного обеспечения.

На этом установку инструментария stand-alone разработки облачных приложений можно считать завершенной.

**Знакомство с инструментарием**

Эмуляторы Compute Emulator (Develop,ent Fabric) и Storage Emulator (development Storage) можно найти в папке Windows Azure SDK\v1.3 (по умолчанию C:\Program Files\Windows Azure SDK\v1.3\). Эмуляторы располагаются в подкаталоге bin.

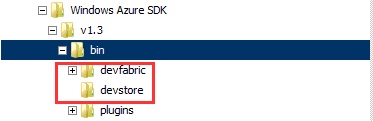


Рисунок 1.12 - Открытие папок

Если установка инструментария завершена корректно, то в списке проектов VS2010 появится шаблон Cloud ([Рисунок. 1.13](http://www.intuit.ru/studies/courses/677/533/lecture/6919?page=3#image.11.13)). Выбор проекта Windows Azure приведет к появлению списка доступных ролей ([Рисунок.1.14](http://www.intuit.ru/studies/courses/677/533/lecture/6919?page=3#image.11.14)).

|  |  |
| --- | --- |
| Таблица 1.1 - Список поддерживаемых ролей: | |
| **Роль** | **Описание** |
| Веб-роль ASP.NET | Основанное на ASP.NET приложение с веб-интерфейсом |
| Веб-роль ASP.NET MVC 2 | Основанное на ASP.NET MVC 2 приложение с веб-интерфейсом |
| Веб-роль ASP.NET службы WCF | WCF - сервис |
| Рабочая роль | Создание фоновой задачи |
| Веб-роль CGI | Хостинг приложения с использованием FastCGI |

Создание первого облачного приложения будет рассмотрено в следующей практической работе.

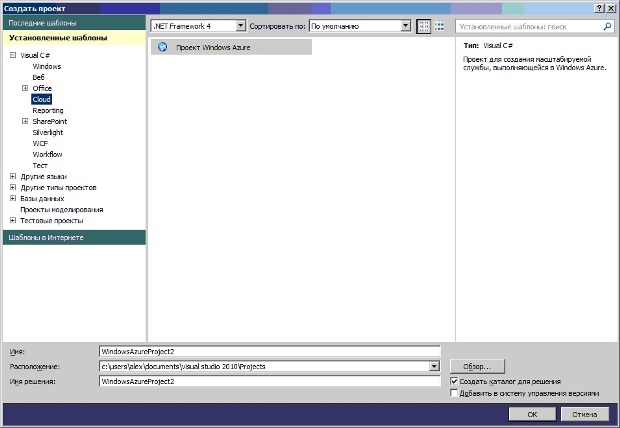
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/11/files/1_13.jpg)

Рисунок 1.13 - Шаблон проекта Cloud

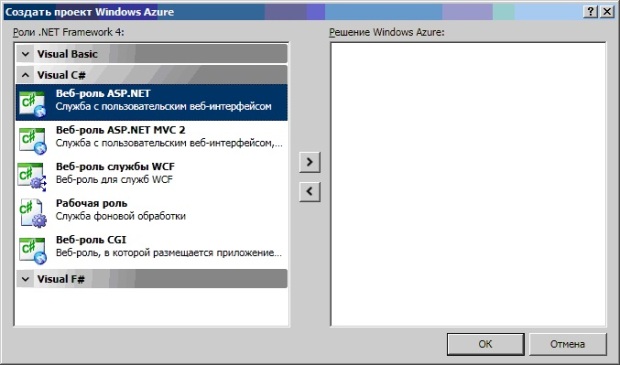


Рисунок 1.14 - Список ролей облачного приложения

**Список вспомогательных материалов**

**Установка Visual Studio 2010**

1. <http://msdn.microsoft.com/ru-ru/library/e2h7fzkw.aspx>

**Установка MS SQL Server 2008**

1. <http://www.alta.ru/mssqlserver2008.php>
2. [http://itband.ru/2010/07/install-microsoft-sql-server-2008-r2/ (версии R2)](http://itband.ru/2010/07/install-microsoft-sql-server-2008-r2/%20%28%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B8%20R2%29)

**Руководства по устранению неполадок**

1. <http://msdn.microsoft.com/ru-ru/library/ee460770.aspx>

**Практическая работа №2. Создание первого проекта**

Отметим, что для работы инструментов создания облачных решений необходимо запустить VS с правами администратора [(Рисунок.2.1)](http://www.intuit.ru/studies/courses/677/533/lecture/6920?page=1#image.12.1).

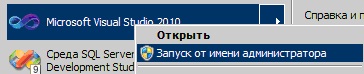


Рисунок 2.1 - Права администратора

Запустим VS 2010 и создадим проект облачной службы с именем WA\_first. [(Рисунок.2.2)](http://www.intuit.ru/studies/courses/677/533/lecture/6920?page=1#image.12.2)

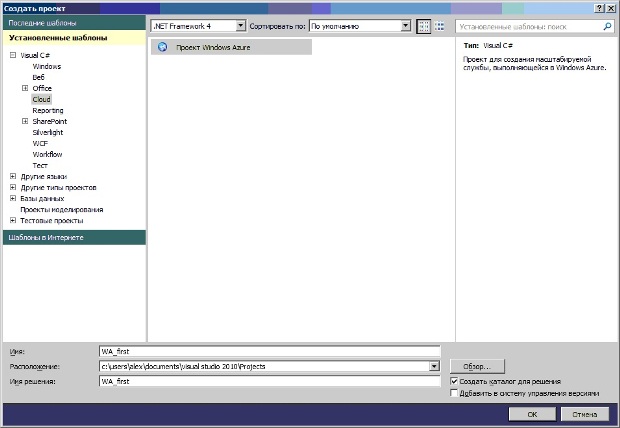
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/12/files/2_2.jpg)

Рисунок 2.2 - Создание проекта

Добавим к проекту решения веб-роль ASP.NET и рабочую роль [(Рисунок.2.3)](http://www.intuit.ru/studies/courses/677/533/lecture/6920?page=1#image.12.3)

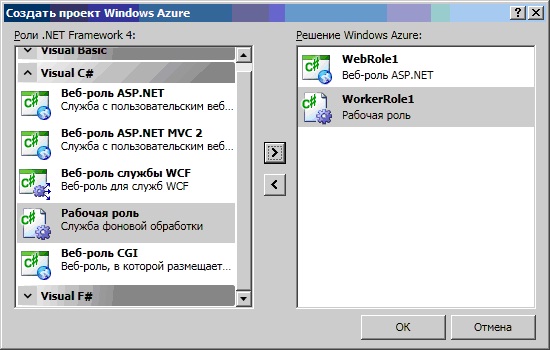


Рисунок 12.3 - Создание проекта

Результатом будет решение, состоящее из трех проектов [(Рисунок.2.4)](http://www.intuit.ru/studies/courses/677/533/lecture/6920?page=1#image.12.4)

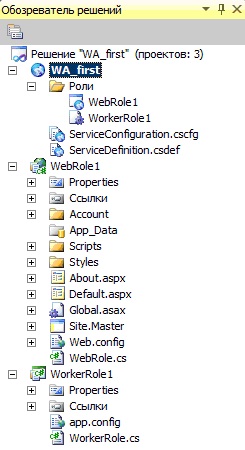


Рисунок 2.4 - Обозреватель решений

Файлы ServiceConfiguration.cscfg и ServiceDefinition.csdef являются конфигурационными и используются для определения характеРисуноктик облачного решения и его ролей. Конфигурационные файлы упаковываются вместе с кодом и разворачиаются в Windows Azure.

ServiceDefinition.csdef - используется для описания приложения и ролей, хранит настройки одинаковые для всех экземпляров ролей. После запуска приложения содержимое данного файла не может быть изменено.

ServiceConfiguration.cscfg - задает значения настроек, описанных в ServiceDefinition.csdef, указывает число экземпляров каждой роли. Содержимое данного файла может быть изменено и после запуска роли.

Страницу свойств роли Windows Azure можно открыть щелкнув правой кнопкой мыши на роли и выбрав "Свойства" [(Рисунок.2.5)](http://www.intuit.ru/studies/courses/677/533/lecture/6920?page=1#image.12.5).

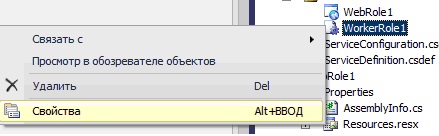


Рисунок 2.5 - Выбор свойств

Рассмотрим свойства нашей рабочей роли [(Рисунок.2.6)](http://www.intuit.ru/studies/courses/677/533/lecture/6920?page=1#image.12.6)

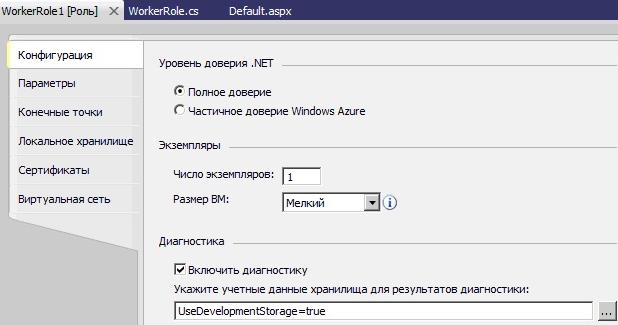


Рисунок 2.6 - Свойства рабочей роли

В разделе "Конфигурация" можно указать уровень доверия .NET. Полное доверие необходимо для выполнения собственного кода приложений FastCGI. Частичное доверие отключает возможность загрузки и использования клиентский библиотек Windows Azure.

Значение настройки "число экземпляров" определяет количество экземпляров, которые должны быть запущены для службы этой роли.

Свойства размер виртуальной машины определяет характеРисуноктики автоматически создаваемой виртуальной машины (Подробнее см. Лекцию №7).

Поскольку мы создаем приложение локально, в учетных данных хранилища в разделе "Диагностика" нужно отметить пункт "Использование эмулятора хранилища Windows Azure" [(Рисунок.2.7)](http://www.intuit.ru/studies/courses/677/533/lecture/6920?page=2#image.12.7), либо вручную ввести строку "UseDevelopmentStorage=true".

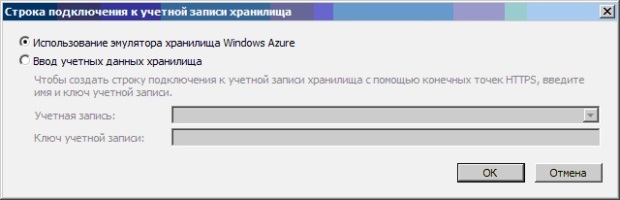


Рисунок 2.7 - Строка подключения к учетной записи

Запуск приложения в режиме эмуляции

Нажав кнопку F5, дождемся окончания построения облачного решения. Результатом будет веб-страница на локальном сервере. В правой части панели задач должна появиться иконка Windows Azure [(Рисунок.2.8)](http://www.intuit.ru/studies/courses/677/533/lecture/6920?page=2#image.12.8)

http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/12/files/2_8.jpg

Рисунок 2.8 - Иконка Windows Azure

Для просмотра информации о работе приложения в режиме эмуляции, нужно щелкнуть правой кнопкой мыши по значку Windows Azure и выбрать "Show Compute Emulator UI" [(Рисунок.2.9)](http://www.intuit.ru/studies/courses/677/533/lecture/6920?page=2#image.12.9)

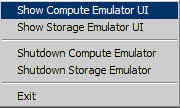


Рисунок 2.9 - "Show Compute Emulator UI"

Появится окно для просмотра статусов ролей нашего приложения [(Рисунок.2.10)](http://www.intuit.ru/studies/courses/677/533/lecture/6920?page=2#image.12.10).

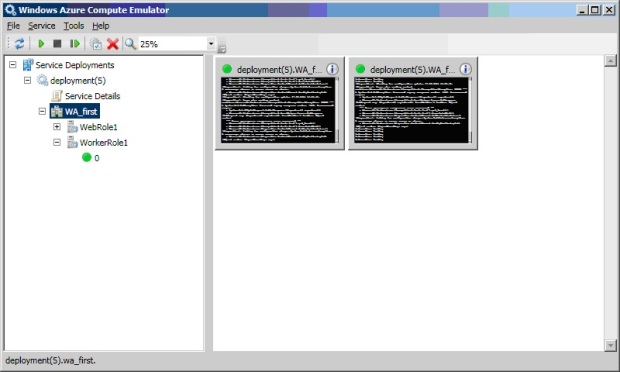
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/12/files/2_10.jpg)

Рисунок 2.10 - Окно для просмотра статусов ролей

Compute emulator – это основной компонент Windows Azure, можно сказать, ядро "облачной" операционной системы, отвечающий за управление виртуальными машинами и экземплярами ролей. Развертывание и старт экземпляра роли начинают ее жизненный цикл.

Подробнее работа по созданию более сложных приложений будет рассмотрена в следующих лабораторных работах.

**Список вспомогательных материалов**

**Настройка облачной службы**

1. <http://msdn.microsoft.com/ru-ru/library/ee405486.aspx>

**Уровни доверия .Net**

1. <http://msdn.microsoft.com/ru-ru/library/dd573355.aspx>
2. <http://msdn.microsoft.com/ru-ru/library/dd573345.aspx>

**Руководства по устранению неполадок**

1. <http://msdn.microsoft.com/ru-ru/library/ee460770.aspx>

**Практическая работа №3. Настройка хранилища разработки в Visual Studio 2010**

**Запуск хранилища разработки**

Рассмотрим более подробно работу с Storage Emulator. По умолчанию Storage Emulator устанавливается в папку devstore подкаталога bin, папки Windows Azure SDK.

Рисунок 3.1

Здесь можно найти два .exe файла:

1. **DSInit** - инициализирует локальное хранилище и устанавливает права доступа к нему. Запустив этот файл, при отсутствии ошибок, должно появиться следующее окно:

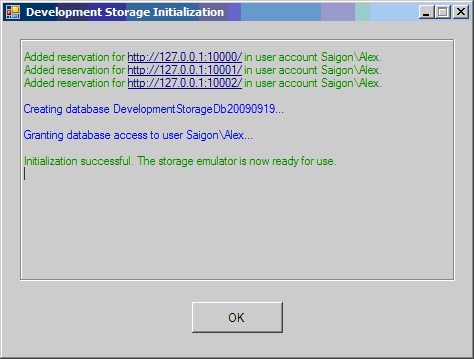


Рисунок 3.2 - Development Storage Initialization

Как видно, была создана локальная база данных для разработки, и зарезервированы порты 10000-10002. В том, что база создана можно также убедиться, запустив SQL Management Studio

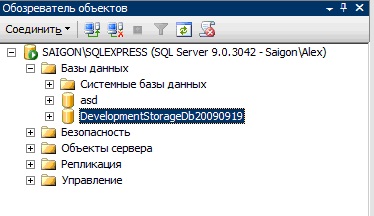


Рисунок 3.3 - Обозреватель объектов

2. **DSService.exe** - непосредственно запускает эмулятор облачного хранилища. Запустив его и подождав некоторое время, можно заметить, что в правом нижнем углу появился значок Windows Azure. Для того, чтобы открыть интерфейс эмулятора, нужно щелкнуть правой кнопкой мыши по значку Windows Azure и выбрать "Show Compute Emulator UI". При отсутствии ошибок, должно появиться следующее окно:

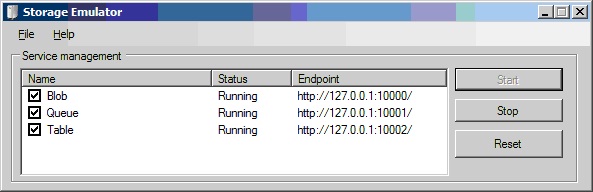


Рисунок 3.4 - Storage Emulator

В окне Storage Emulator отображается состояние и конечные точки сервисов эмулятора: Blob, Queue и Table. Сервисы можно запустить, остановить, либо сбросить, с потерей данных, хранящихся в них.

Важно отследить следующий момент: порты, указанные для каждого из сервисов должны быть свободны. В случае, если при запуске хранилища разработки вы видите следующую ошибку:

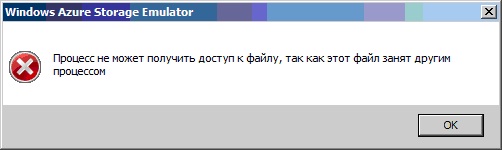


Рисунок 3.5 - Windows Azure Storage Emulator

Это скорее всего означает, что указанные порты "слушают" другие приложения. К примеру, как отмечалось в ряде источников и было проверено нами, клиент mtorrent, запущенный до старта эмулятора конфликтует с Blob - сервисом, из-за чего к последнему не удается получить доступ.

**Подключение к хранилищу разработки**

Для подключения к эмулятору хранилища при разработке приложения (в среде Visual Studio в нашем случае) после создания проекта необходимо перейти к свойствам роли

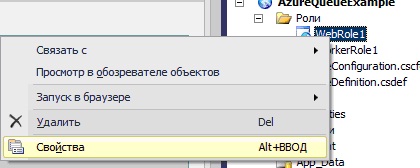


Рисунок 3.6 - Свойства ролей

А затем во вкладке "Параметры" добавить параметр строки подключения и указать значение "UseDevelopmentStorage=true"

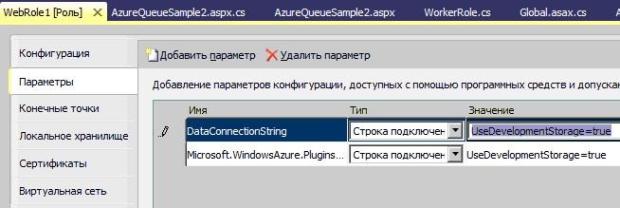


Рисунок 3.7 - Вкладка "Параметры"

При работе непосредственно с хранилищем Windows Azure, в той же вкладке необходимо, при создании строки подключения, нажать кнопку "..."

[http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/14/files/3_8sm.jpg](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/14/files/3_8.jpg)

Рисунок 3.8 - Хранилище Windows Azure

И в появившемся окне задать параметры подключения и используемой учетной записи Windows Azure.

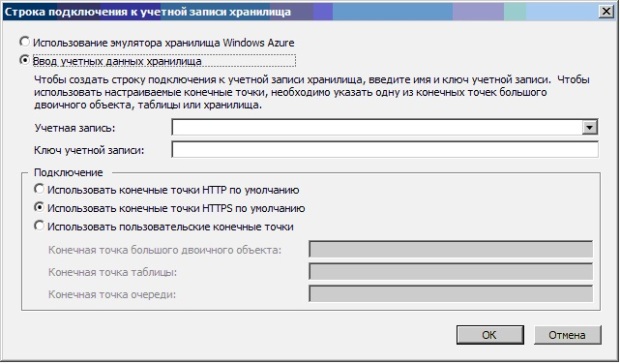


Рисунок 3.9 - Строка подключения к учетной записи хранилища

Запустив Обозреватель серверов (Меню "Вид" - Обозреватель серверов), увидим появившееся хранилище Windows Azure и хранилище "Разработка" - являющееся отображением эмулятора.

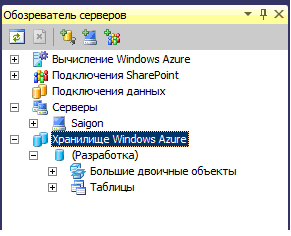


Рисунок 3.10 - Обозреватель серверов

При помощи обозревателя серверов можно просматривать содержимое конкретных таблиц или контейнеров бинарных объектов.

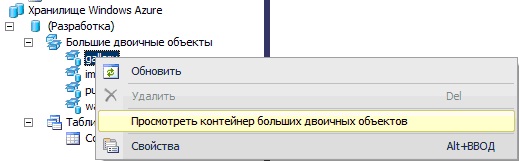


Рисунок 3.11 - Обозреватель серверов

Также при помощи обозревателя серверов можно подключиться к хранилищу Windows Azure, добавив данные учетной записи.

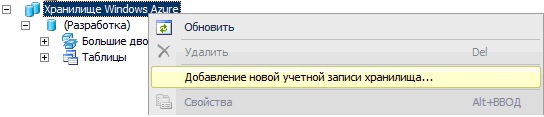


Рисунок 3.12 - Хранилище Windows Azure

Для того, чтобы продемонстрировать подключение к хранилищу данных, рассмотри небольшой пример, по созданию хранилища с простой структурой - это будет таблица - список юридических лиц.

**Задание. Создание хранилища с простой структурой данных.**

1. Создадим проект облачной службы. SimpleDataStructure (Меню "Файл" - Создать - Проект). Добавим решению рабочую роль.

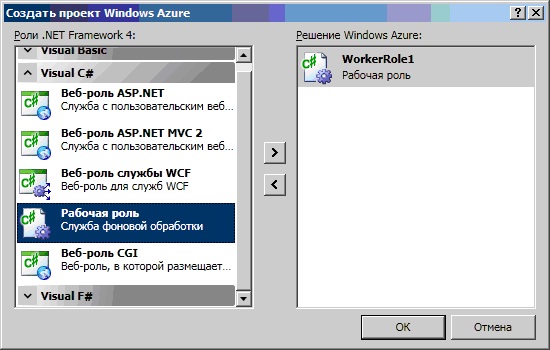


Рисунок 3.13 - Создание проекта

Наше приложение будет подключаться к эмулятору хранилища, создавать таблицу Firm, если ее не существует и добавлять туда одну произвольную запись.

В свойствах рабочей роли определим строку подключения к эмулятору хранилища Azure.

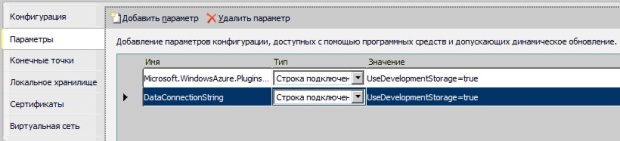
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/14/files/3_14.jpg)

Рисунок 3.14 - Параметры конфигурации

Нам необходим класс, который будет описывать структуру сущности для нашей таблицы. Класс должен быть наследником Microsoft.WindowsAzure.StorageClient.TableServiceEntity

class Address : TableServiceEntity

{

public String address { get; set; }

public String firm { get; set; }

public String telephone { get; set; }

}

Для создания таблиц необходимо определить класс - контекст, при чем:

класс должен быть наследником TableServiceContext

Для каждой таблицы необходимо определить свойство типа IQueryable, где значение параметра DataItemType - тип сущностей, хранимых в таблице, в нашем случае - Address.

class AddressConext: TableServiceContext

{

public IQueryable<Address> ContactData

{

get

{

return this.CreateQuery<Address>("Address");

}

}

public AddressConext(Uri baseAddress, StorageCredentials credentials) : base(baseAddress.AbsoluteUri, credentials) { }

}

Теперь обратим внимание на методы Run и OnStart. Первый содержит код выполняемый в ходе работы роли, второй - при ее запуске.

Для того, чтобы добавить данные в таблицу нам необходимо в методе Run:

* создать экземпляр класса - учетной записи
* создать экземпляр класс - контекста
* создать экземпляр класса - сущности и задать его параметры
* создать таблицу Address, если она не существует
* добавить сущность в таблицу

Добавим следующий код:

CloudStorageAccount.SetConfigurationSettingPublisher(

(configName, configSettingPublisher) =>

{

var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);

configSettingPublisher(connectionString);

}

);

//определение учетной записи

CloudStorageAccount account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

//создание таблицы Windows Azure Table

CloudTableClient \_tc = null;

\_tc = account.CreateCloudTableClient();

\_tc.CreateTableIfNotExist("Address");

/\*определение сущности, в том числе свойств ключ строки и ключ секции, унаследованных от родительского TableServiceEntity\*/

Address adrs = new Address();

adrs.PartitionKey = "Firm";

adrs.RowKey = "Test entity";

adrs.telephone = "xxx-xx-xx";

adrs.address = "Evergreen Terrace 247";

adrs.firm = "My new firm";

//определение контекста

AddressConext context = new AddressConext(account.TableEndpoint, account.Credentials);

//добавление сущности таблице Address

context.AddObject("Address", adrs);

//сохранение изменений

context.SaveChanges();

Запустите приложение, убедитесь в том, что оно выполнилось без ошибок и остановите его.

В диспетчере серверов, во вкладке "Хранилище Windows Azure" обновите вкладку "Таблицы", вы увидите созданную нашим приложением таблицу Address.

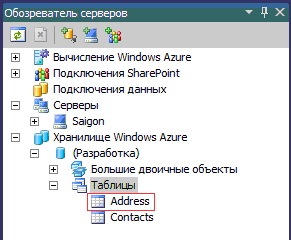


Рисунок 3.15 - Обозреватель серверов

Щелкните на таблице правой кнопкой мыши и выберите "Просмотреть данные". Вы увидите, что определенная нами сущность добавлена в таблицу.

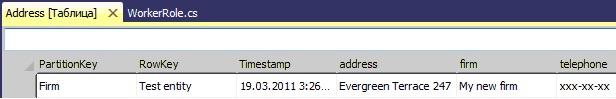


Рисунок 3.16 - Добавление таблиц

**Список вспомогательных материалов**

1. <http://www.michaelfcollins3.me/2010/07/creating-table-storage-entities-for-windows-azure/>

**Приложение WorkerRole1.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Net;

using System.Threading;

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.Diagnostics;

using Microsoft.WindowsAzure.ServiceRuntime;

using Microsoft.WindowsAzure.StorageClient;

using System.Data.Services.Client;

namespace WorkerRole1

{

public class WorkerRole : RoleEntryPoint

{

public override void Run()

{

CloudStorageAccount.SetConfigurationSettingPublisher(

(configName, configSettingPublisher) =>

{

var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);

configSettingPublisher(connectionString);

}

);

CloudStorageAccount account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

//создание таблицы Windows Azure Table

CloudTableClient \_tc = null;

\_tc = account.CreateCloudTableClient();

\_tc.DeleteTableIfExist("Address");

\_tc.CreateTableIfNotExist("Address");

Address adrs = new Address();

adrs.PartitionKey = "Firm";

adrs.RowKey = "Test entity";

adrs.telephone = "xxx-xx-xx";

adrs.address = "Evergreen Terrace 247";

adrs.firm = "My new firm";

AddressConext context = new AddressConext(account.TableEndpoint, account.Credentials);

context.AddObject("Address", adrs);

context.SaveChanges();

}

public override bool OnStart()

{

ServicePointManager.DefaultConnectionLimit = 12;

return base.OnStart();

}

class Address : TableServiceEntity

{

public String address { get; set; }

public String firm { get; set; }

public String telephone { get; set; }

}

class AddressConext: TableServiceContext

{

public IQueryable<Address> ContactData

{

get

{

return this.CreateQuery<Address>("Address");

}

}

public AddressConext(Uri baseAddress, StorageCredentials credentials) : base(baseAddress.AbsoluteUri, credentials) { }

}

}

}

**Практическая работа №4. Хранилище данных с реляционной структурой**

Хранилище Windows Azure, помимо прочего, подходит для хранения реляционных данных, для этого используются возможности Windows Azure Table. Однако, само по себе, табличное хранилище Windows Azure не хранит данные в реляционном виде. Возникает вопрос, каким образом возможно хранить реляционную структуру и как обеспечить миграцию данных?

Ответов на данные вопросы может быть множество. То как вы решите переносить существующую реляционную структуру данных в облако, зависит целиком от вас самих. Мы продемонстрируем самый простой и очевидный способ - построчное чтение данных из реляционной БД и запись их в таблицу Windows Azure, что и будет целью данной практической работы.

Во-первых, нам понадобится реляционная база данных. В нашей базе будут храниться 3 взаимосвязанные сущности - Адрес, Фирма и Сотрудник.

В данной лабораторной работе мы использовали SQL Management Studio для соединения с базой sqlexpress и выполнения запросов.

Создадим базу данных при помощи запроса 4.1

create database azureexample

Листинг 4.1.

Теперь необходимо создать таблицы, связи между ними и заполнить их тестовым набором данных. Вы можете проделать это самостоятельно, а можно просто выполнить нижеследующий зарос:

use azureexample

create table Address(

AddresID int identity(1,1) primary key,

Country nvarchar(max),

City nvarchar(max),

Street nvarchar(max),

House int

)

create table Firm(

FirmID int identity(1,1) primary key,

NameOf nvarchar(max),

Telephone nvarchar(10),

Email nvarchar(max),

AddressKey int not null

)

create table Employee(

EmployeeID int identity(1,1) primary key,

FirstName nvarchar(max),

LastName nvarchar(max),

Telephone nvarchar(10),

FirmKey int

)

Alter table dbo.Firm add constraint

FK\_Firm\_Address foreign key

(

AddressKey

) references dbo.Address

(

AddresID

) on update no action

On delete no action

alter table dbo.Employee add constraint

FK\_Employee\_Firm foreign key

(

FirmKey

)

references dbo.Firm

(

FirmID

)

on update no action

on delete no action

insert into Address

values ('RF', 'Tomsk', 'Evergreen Terrace', '247')

insert into Firm

values ('Firm1', 'xxx-xx-xx', 'firm1@testmail.com', '1')

insert into Firm

values ('Firm2', 'xxx-xx-xx', 'firm2@testmail.com', '1')

insert into Employee

values ('Ivan', 'Ivanov', 'x-xxx-xx','1')

insert into Employee

values ('Victor', 'Romanov', 'x-xxx-xx','1')

insert into Employee

values ('Alex', 'Petrov', 'x-xxx-xx','2')

Таким образом мы получили три таблицы. Диаграмма данных созданной базы представлена ниже:

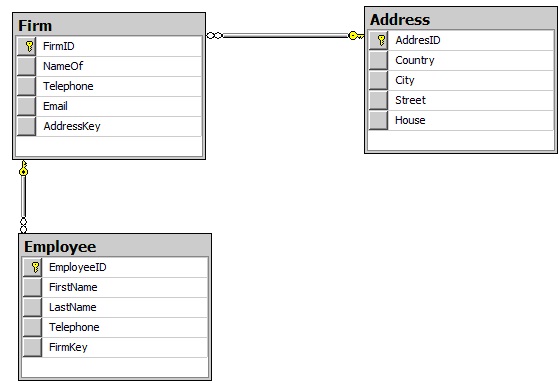


Рисунок 4.1 - Диаграмма данных

Теперь необходимо решить, каким образом данная реляционная структура может быть перенесена в таблицу. Как вам известно из лекции, таблицу Windows Azure можно соотнести с коллекцией взаимосвязанных сущностей, если проводить аналогии с реляционными базами данных. В то время, как строки таблиц Windows Azure, по сути являются экземплярами конкретных сущностей.

Мы предлагаем следующее очевидное решение:

Создать Windows Azure таблицу "Relational", которая будет содержать в себе сущности Address, Firm и Employee

Ключ секции будет указывать на соответствие строки таблицы Azure экземпляру сущности реляционной БД, т.е. для всех сотрудников ключ секции будет - "Employee"

Значения атрибутов - ключей в реляционных таблицах будут являться значениями RowKey для таблицы "Relational", такими образом пара ключ секции - ключ строки будет являться уникальным идентификатором однозначно указывающим на принадлежность Azure - строки конкретной сущности исходной БД и определяющим экземпляр сущности

Значения остальных атрибутов будут перенесены без изменений.

Мы не видим необходимости в создании нового VS - проекта, поэтому просто изменим WorkerRole.cs проекта, созданного в рамках предыдущей практической работы. А именно, метод **Run**, чей листинг представлен ниже:

CloudStorageAccount.SetConfigurationSettingPublisher(

(configName, configSettingPublisher) =>

{

var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);

configSettingPublisher(connectionString);

}

);

CloudStorageAccount account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

//создание таблицы Windows Azure Table

CloudTableClient \_tc = null;

\_tc = account.CreateCloudTableClient();

\_tc.CreateTableIfNotExist("Relational");

Context context = new Context(account.TableEndpoint, account.Credentials);

//определение параметров подключения к БД, измените строку подключения

//соответствующим образом, для соединения с вашим sql - сервером

SqlConnection conn = new SqlConnection("Data Source=saigon\\sqlexpress;

Initial Catalog=azureexample; Integrated Security=true;");

//импорт данных из таблицы Address

//указываем команду для чтения данных из базы

SqlCommand cmd = new SqlCommand("SELECT \* FROM Address", conn);

conn.Open();

SqlDataReader adr = cmd.ExecuteReader();

while (adr.Read())

{

//в параметрах метода AddObject указывает имя таблицы и определяем новую сущность класс //Address

context.AddObject("Relational", new Address

{

PartitionKey = "Address",

RowKey = adr["AddresID"].ToString(),

country = adr["Country"].ToString(),

city = adr["City"].ToString(),

street = adr["Street"].ToString(),

house = Convert.ToInt32(adr["House"].ToString())

});

context.SaveChanges();

}

adr.Close();

//импорт данных из таблицы Firm

cmd.CommandText = "SELECT \* FROM Firm";

SqlDataReader frm = cmd.ExecuteReader();

while (frm.Read())

{

context.AddObject("Relational", new Firm

{

PartitionKey = "Firm",

RowKey = frm["FirmID"].ToString(),

nameof = frm["NameOf"].ToString(),

telephone = frm["Telephone"].ToString(),

email = frm["Email"].ToString(),

adresskey = Convert.ToInt32(frm["AddressKey"].ToString())

});

context.SaveChanges();

}

frm.Close();

//импорт данных из таблицы Employee

cmd.CommandText = "SELECT \* FROM Employee";

SqlDataReader emp = cmd.ExecuteReader();

while (emp.Read())

{

context.AddObject("Relational", new Employee

{

PartitionKey = "Employee",

RowKey = emp["EmployeeID"].ToString(),

firstname = emp["FirstName"].ToString(),

lastname = emp["LastName"].ToString(),

telephone = emp["Telephone"].ToString(),

firmkey = Convert.ToInt32(emp["FirmKey"].ToString())

});

context.SaveChanges();

}

emp.Close();

conn.Close();

Кроме того, нам необходимо создать классы - соответствующие сущностям реляционной базы данных, а также класс - контекст.

**Класс Address.cs**

class Address: TableServiceEntity

{

public String country { get; set; }

public String city { get; set; }

public String street { get; set; }

public int house { get; set; }

}

**Класс Firm.cs**

class Firm: TableServiceEntity

{

public String nameof { get; set; }

public String telephone { get; set; }

public String email { get; set; }

public int adresskey { get; set; }

}

**Класс Employee.cs**

class Employee: TableServiceEntity

{

public String firstname { get; set; }

public String lastname { get; set; }

public String telephone { get; set;}

public int firmkey { get; set; }

}

**Класс Context.cs**

class Context: TableServiceContext

{

public IQueryable<Address> ContactData

{

get

{

return this.CreateQuery<Address>("Address");

}

}

public Context(Uri baseAddress, StorageCredentials credentials) :

base(baseAddress.AbsoluteUri, credentials) { }

}

Запустите приложение и дождитесь конца его выполнения. В обозревателе серверов раскройте вкладку "Хранилище Windows Azure", затем обновите вкладку "Таблицы" и раскройте ее. Как вы можете видеть, появилась таблица "Relational".

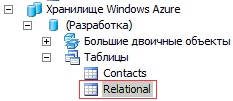


Рисунок 4.2 - Таблица "Relational"

Если вы решите ее просмотреть, то увидите, что данные из реляционной БД успешно перенесены в таблицу.

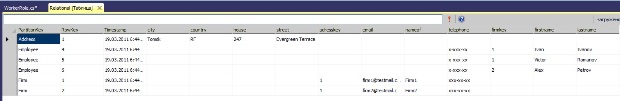
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/16/files/4_3.jpg)

Рисунок 4.3 - Таблицы базы данных

**Практическая работа №5. Работа с Windows Azure Table**

В качестве примера рассмотрим простое приложение - справочник контактов, с поддержкой функции добавления, изменения и удаления.

Прежде всего создадим проект "AzureTableExample", необходимый для данной лабораторной работы.

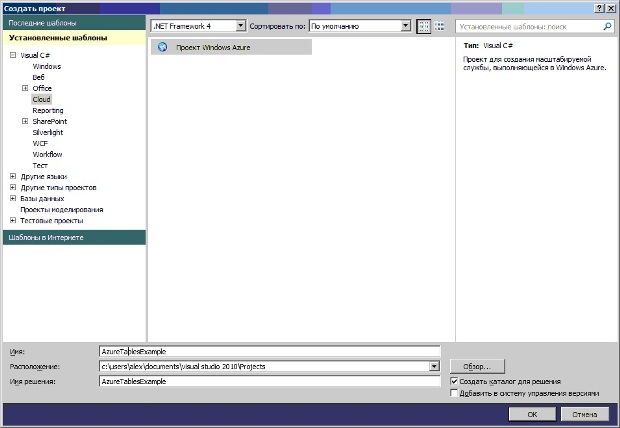
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/17/files/5_1.jpg)

Рисунок 5.1 - Создание проекта

Для демонстраци возможностей приложения, использующего в качестве хранилища данных WindowsAzure Table нам понадобится только веб - роль, добавьте ее решению.

В свойствах веб - роли определим строку подключения к эмулятору хранилища Azure.

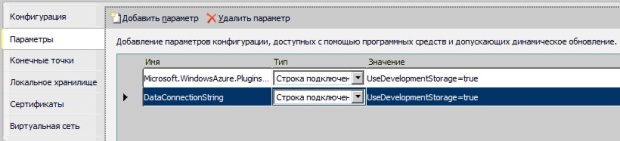
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/17/files/5_2.jpg)

Рисунок 5.2 - Параметры конфигурации

**Создание классов будущего приложения**

**Класс - Контакт.**

Добавим следующие ссылки:

using Microsoft.WindowsAzure.StorageClient;

Контакт будет содержать следующую информацию:

* Имя
* Фамилию
* Контактный телефон
* Адрес электронной почты

Добавим C# класс нашей рабочей роли - Contact.cs и зададим необходимые свойства, унаследовав его от Microsoft.WindowsAzure.StorageClient.TableServiceEntity :

class Contact : Microsoft.WindowsAzure.StorageClient.TableServiceEntity

{

}

Определим требуемые свойства для класса контактов. И в конечном итоге, наш класс будет выглядеть следующим образом:

class Contact : Microsoft.WindowsAzure.StorageClient.TableServiceEntity

{

public String FirstName { get; set; }

public String LastName { get; set; }

public String TelNumber { get; set; }

public String Email { get; set; }

}

**Класс - контекст для доступа ContactContext.cs**

Нам понадобятся следующие ссылки:

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.StorageClient;

using System.Data.Services.Client;

Создадим соответствующий C# - класс, унаследуем его от TableServiceContext и определим конструктор:

class ContactContext : TableServiceContext

{

public ContactContext (Uri baseAddress, StorageCredentials credentials): base(baseAddress.AbsoluteUri, credentials) {}

}

Добавим свойство для возвращения data service - запроса для таблицы Contacts.

public IQueryable<Contact> ContactData

{

get

{

return this.CreateQuery<Contact>("Contacts");

}

}

Последним шагом по созданию класса - контекста является создание метода для добавления строки в таблицу. В итоге, класс должен выглядеть следующим образом 5.1:

class ContactContext : TableServiceContext

{

public IQueryable<Contact> ContactData

{

get

{

return this.CreateQuery<Contact>("Contacts");

}

}

public ContactContext (Uri baseAddress, StorageCredentials credentials): base(baseAddress.AbsoluteUri, credentials) {}

public void Add(Contact cntct)

{

this.AddObject("Contacts", cntct);

this.SaveChanges();

}

}

Листинг 5.1.

**Интерфейc**

Создадим веб - форму для работы с нашим контакт - листом.

Внешний вид веб формы

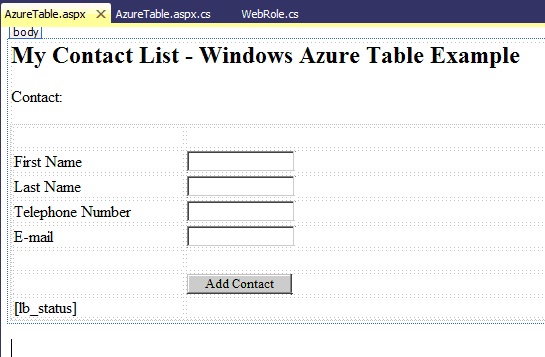


Рисунок 5.3 - Веб-форма

**Задача 1. Добавление данных в таблицу**

В обозревателе решений найдите файл Global.asax.cs. В метод Application\_Start необходимо добавить код создания таблицы, в случае если она не была создана до этого:

void Application\_Start(object sender, EventArgs e)

{

CloudStorageAccount.SetConfigurationSettingPublisher(

(configName, configSettingPublisher) =>

{

var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);

configSettingPublisher(connectionString);

}

);

var account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

//создание таблицы Windows Azure Table

CloudTableClient \_tc = null;

\_tc = account.CreateCloudTableClient();

\_tc.CreateTableIfNotExist("Contacts");

}

Далее в обработчике события нажатия кнопки "AddContact" необходимо создать аккаунт для подключения к хранилищу, используя параметр строки подключения "DataConnection String" . После этого, для непосредственного добавления данных в таблицу нам понадобится экземпляр класса - контекса. Вызвав метод "Add" данного класса [(листинг 5.1)](http://www.intuit.ru/studies/courses/677/533/lecture/6925?page=1#example.17.1) мы добавим строку в таблицу, передав в качестве параметра экземпляр класса "Contact" . Обратите внимание, в классе Contact мы не описывали необходимые параметры для добавления строки в таблицу (ключ секции и ключ строки), класс унаследовал их от родителя. В качестве значения ключа секции зададим произвольную строку, в качестве ключа строки будем использовать сочетание "Last Name + First Name".

protected void btn\_add\_Click(object sender, EventArgs e)

{

var statusMessage = String.Empty;

try

{

var account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

var context = new ContactContext(account.TableEndpoint, account.Credentials);

context.Add(new Contact { PartitionKey = "MyContacts",

RowKey = this.tb\_lastname.Text + " " + this.tb\_firstname.Text,

FirstName = tb\_firstname.Text, LastName = tb\_lastname.Text,

TelNumber = tb\_telnum.Text, Email = tb\_email.Text });

}

catch (DataServiceRequestException ex)

{

statusMessage = "Unable to connect to the table storage server. Please check that the service is running.<br>"

+ ex.Message;

}

lb\_status.Text = statusMessage;

}

Запустив отладку приложения, введем необходимые данные в появившуюся веб - форму и нажмем кнопку "Add Contact".

asp - код веб - формы данной задачи:

<%@ Page Language="C#"

AutoEventWireup="true" CodeBehind="AzureTable.aspx.cs" Inherits="WebRole1.AzureTable" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

<style type="text/css">

.style1

{

width: 169px;

}

</style>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Label ID="NameOfExample" runat="server" Font-Bold="True"

Font-Italic="False" Font-Size="Larger" Font-Strikeout="False"

Font-Underline="False" Text="My Contact List - Windows Azure Table Example"></asp:Label>

<br />

<br />

<asp:Label ID="ContactLabel" runat="server" Text="Contact:"></asp:Label>

<br />

<table style="width:100%;">

<tr>

<td class="style1">

 </td>

<td>

 </td>

</tr>

<tr>

<td class="style1">

<asp:Label ID="lb\_firstname" runat="server" Text="First Name"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_firstname" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td class="style1">

<asp:Label ID="lb\_lastname" runat="server" Text="Last Name"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_lastname" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td class="style1">

<asp:Label ID="lb\_telnum" runat="server" Text="Telephone Number"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_telnum" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td class="style1">

<asp:Label ID="lb\_email" runat="server" Text="E-mail"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_email" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td class="style1">

 </td>

<td>

 </td>

</tr>

<tr>

<td class="style1">

 </td>

<td>

<asp:Button ID="btn\_add" runat="server" Height="20px" Text="Add Contact" />

</td>

</tr>

<tr>

<td class="style1">

<asp:Label ID="lb\_status" runat="server"></asp:Label>

</td>

<td>

 </td>

</tr>

</table>

</div>

</form>

</body>

</html>

**Задача 2. Просмотр данных**

Форма создана. Классы сформированы. Осталось понять добавилось ли что-либо в наше хранилище. Открыв обозреватель серверов и раскрыв последовательно вкладки "Хранилище Windows Azure", "Разработка" и "Таблицы", мы увидим созданную таблицу "Contacts".

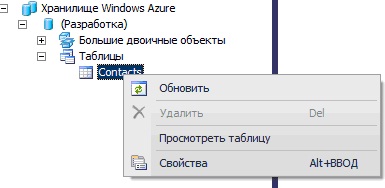


Рисунок 5.4 - Таблица "Contacts"

Щелкнув правой кнопкой мыши на нашей таблице и выбрав "Просмотреть таблицу", получив следующее:

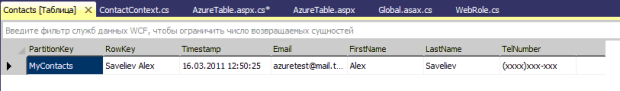
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/17/files/5_5.png)

Рисунок 5.5 - Таблица "Contacts"

Как видно, данные успешно добавлены.

Но нас интересует отображение списка контактов в рамках нашего приложения.

Для этого добавим на веб - форму элемент управления GridView с ID = contactGV , который мы будем использовать для отображения текущих данных таблицы.

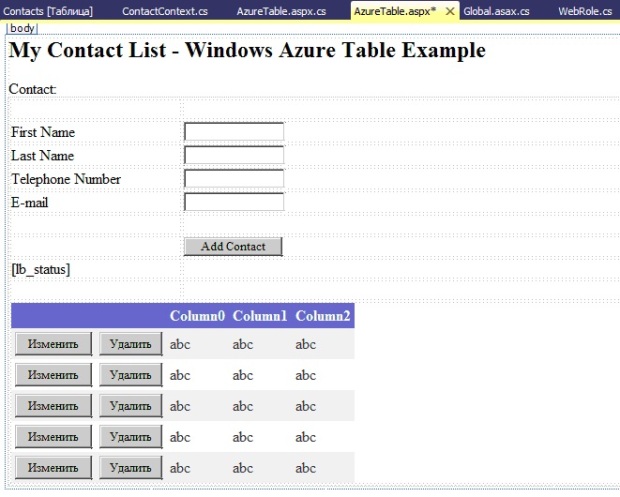
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/17/files/5_6.jpg)

Рисунок 5.6 - Работа проекта

asp - код для GridView:

<asp:GridView ID="contactGV" runat="server" CellPadding="4" ForeColor="#333333"

GridLines="None" onrowdeleting="contactGV\_RowDeleting"

onselectedindexchanged="contactGV\_SelectedIndexChanged">

<AlternatingRowStyle BackColor="White" />

<Columns>

<asp:CommandField ButtonType="Button" SelectText="Изменить"

ShowSelectButton="True" />

<asp:CommandField ButtonType="Button" ShowDeleteButton="True" />

</Columns>

<EditRowStyle BackColor="#2461BF" />

<FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />

<HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />

<PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />

<RowStyle BackColor="#EFF3FB" />

<SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor="#333333" />

<SortedAscendingCellStyle BackColor="#F5F7FB" />

<SortedAscendingHeaderStyle BackColor="#6D95E1" />

<SortedDescendingCellStyle BackColor="#E9EBEF" />

<SortedDescendingHeaderStyle BackColor="#4870BE" />

</asp:GridView>

Определим переменные учетной записи и контекста, до метода **Page\_Load**:

private CloudStorageAccount account = null;

private ContactContext context = null;

В метод **Page\_Load** нашей страницы добавим следующий код, для привязки GridView к источнику данных:

account= CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

context = new ContactContext(account.TableEndpoint, account.Credentials);

contactGV.DataSource = context.ContactData;

contactGV.DataBind();

Листинг 5.2.

Также в методе btn\_add\_Click добавим следующую строку:

contactGV.DataBind();

Запустив приложение, получим следующее:

[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/17/files/5_7.jpg)

Рисунок 5.7 - Запуск приложения

Как видно, при загрузке страницы мы сразу получаем и текущий список контактов. Если добавить новый контакт, то [(см. 5.2)](http://www.intuit.ru/studies/courses/677/533/lecture/6925?page=3#example.17.2) список дополнится еще одной записью.

[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/17/files/5_8.jpg)

Рисунок 5.8 - Запуск страницы приложения

**Задача 3. Редактирование и удаление сущностей**

По своей сути задачи редактирование и удаления сущностей конкретной таблицы весьма проста. Достаточно вызвать соответствующие методы класса - контекста. Однако, в качестве параметров методов выступают изменяемые сущности. Таким образом, все сводится к выделению конкретной сущности из таблицы по ее ключам секции и строки, которые, напомним, являются частями уникального ключа сущности.

Поскольку нашей целью является демонстрация работы с Windows Azure Table, мы пойдем по самому простому пути.

Для начала добавим на форму кнопку btn\_change, невидимую при загрузке страницы.

Данные между методами формы будем передавать при помощи сессий.

Первая и главная задача сводится к тому, чтобы определить ключи строки и секции редактируемой сущности. Учитывая, что структура таблицы и названия параметров могут быть произвольны, за одним исключением - как раз параметров Partition Key и Row Key , добавим в метод **Page\_Load** следующее:

int i = 0;

foreach (TableCell cell in contactGV.HeaderRow.Cells)

{

if (cell.Text == "PartitionKey") { Session["pkindex"] = i; }

if (cell.Text == "RowKey") { Session["rkindex"] = i; }

i++;

}

Теперь, вне зависимости от структуры таблицы, сессии pkindex и rkindex будут содержать номера столбцов contactGV , в которых находятся параметры ключей секции и строки.

Полностью метод **Page\_Load** для данного задания должен быть следующим:

protected void Page\_Load(object sender, EventArgs e)

{

btn\_change.Visible = false;

account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

context = new ContactContext(account.TableEndpoint, account.Credentials);

contactGV.DataSource = context.ContactData;

contactGV.DataBind();

int i = 0;

foreach (TableCell cell in contactGV.HeaderRow.Cells)

{

if (cell.Text == "PartitionKey") { Session["pkindex"] = i; }

if (cell.Text == "RowKey") { Session["rkindex"] = i; }

i++;

}

}

Также добавим в класс ContactContext методы для обновления и удаления сущностей - Update и Delete соответственно:

public void Delete(Contact cnt)

{

this.DeleteObject(cnt);

this.SaveChanges();

}

public void Update(Contact cnt)

{

this.UpdateObject(cnt);

this.SaveChanges();

}

**Удаление строки**

Добавим метод, обрабатывающий нажатие кнопки "Удалить" элемента contactGV.

protected void contactGV\_RowDeleting(object sender, GridViewDeleteEventArgs e)

{

GridView g = (GridView)sender;

try

{

Contact c = (from contact in context.CreateQuery<Contact>("Contacts")

where contact.PartitionKey == g.Rows[e.RowIndex].

ells[Convert.ToInt32(Session["pkindex"].ToString())].Text

&& contact.RowKey == g.Rows[e.RowIndex].Cells[Convert.ToInt32

(Session["rkindex"].ToString())].Text

select contact).FirstOrDefault();

context.Delete(c);

}

catch(DataServiceRequestException ex)

{

lb\_status.Text = ex.Message;

}

g.DataBind();

}

Обратим ваше внимание на то, что сущность для удаления мы получаем при помощи linq - запроса, указывая значения параметров PartitionKey и RowKey. Значения же параметров мы получаем из contactGV, указывая значения соответствующих ячеек.

Реализация удаления сущности на этом закончена.

**Редактирование сущности**

Для начала напишем обработчик события изменения индекса выбранной строки contactGV , инициируемое нажатием кнопки "Изменить" . При нажатии этой кнопки должен становиться видимым элемент управления btn\_change, а соответствующие текстовые поля заполняться значениями параметров редактируемой строки.

protected void contactGV\_SelectedIndexChanged(object sender, EventArgs e)

{

GridView g = (GridView)sender;

int index = g.SelectedIndex;

Contact c = (from contact in context.CreateQuery<Contact>("Contacts")

where contact.PartitionKey == g.Rows[index].

Cells[Convert.ToInt32(Session["pkindex"].ToString())].Text

&& contact.RowKey == g.Rows[index].

Cells[Convert.ToInt32(Session["rkindex"].ToString())].Text

select contact).FirstOrDefault();

tb\_firstname.Text = c.FirstName;

tb\_lastname.Text = c.LastName;

tb\_email.Text = c.Email;

tb\_telnum.Text = c.TelNumber;

Session["index"] = index;

btn\_change.Visible = true;

}

Здесь стоит обратить внимание разве что на формирование еще одной сессии, в которой мы будем хранить номер редактируемой строки в contactGV.

Вот, что должно получиться при нажатии кнопки "Изменить" напротив второй строки списка контактов:

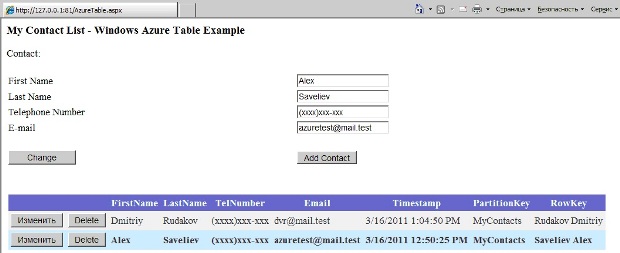
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/17/files/5_9.jpg)

Рисунок 5.9 - Список контактов

Как видим, значения текстовых полей стали соответствовать значениям параметров редактируемой строки.

Осталось только написать метод обрабатывающий событие нажатия кнопки btn\_change.

Он будет выглядеть следующим образом:

protected void btn\_change\_Click(object sender, EventArgs e)

{

int index = Convert.ToInt32(Session["index"].ToString());

try

{

Contact c = (from contact in context.CreateQuery<Contact>("Contacts")

where contact.PartitionKey == contactGV.Rows[index]. Cells[Convert.ToInt32(Session["pkindex"].ToString())].Text

&& contact.RowKey == contactGV.Rows[index].

Cells[Convert.ToInt32(Session["rkindex"].ToString())].Text

select contact).FirstOrDefault();

c.FirstName = tb\_firstname.Text;

c.LastName = tb\_lastname.Text;

c.Email = tb\_email.Text;

c.TelNumber = tb\_telnum.Text;

context.Update(c);

}

catch (DataServiceRequestException a)

{

lb\_status.Text = a.Message;

}

contactGV.DataBind();

}

Номер изменяемой строки мы получаем из сессии "index" , остальное уже не должно вызывать вопросов.

Запустите приложение еще раз и измените произвольным образом любой из параметров, либо насколько из них какой - либо строки. Мы изменили Email на tableentitychange@mail.test . Нажмите кнопку "Change".

Мы получили следующее:

[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/17/files/5_10.jpg)

Рисунок 5.10 - Работа с проектом

Еще раз обратим ваше внимание на то, что демонстрируемое приложение является не более, чем примером способов работы с Windows Azure Storage, поэтому мы пренебрегли обработкой исключительных ситуаций и проверкой правильности и целостности введенных данных. Оставляем это на ваше усмотрение.

В случае, если выполнение задания вызвало сложности и затруднения, в приложениях к данной практической работе вы найдете итоговый программный код в том виде, в котором он необходим для Задания№3.

**Список вспомогательных материалов**

**Работа с Windows Azure Table**

1. <http://msdn.microsoft.com/ru-ru/magazine/gg309178.aspx>
2. <http://msdn.microsoft.com/en-us/wazplatformtrainingcourse_exploringwindowsazurestoragevs2010_topic2#_Toc282791662>
3. <http://blogs.msdn.com/b/morebits/archive/2010/12/26/building-windows-azure-service-part3-table-storage.aspx>

**Приложение А. ASP - код веб - формы (AzureTable.aspx)**

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="AzureTable.aspx.cs" Inherits="WebRole1.AzureTable" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

<style type="text/css">

.style1

{

}

.style2

{

width: 149px;

}

</style>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Label ID="NameOfExample" runat="server" Font-Bold="True"

Font-Italic="False" Font-Size="Larger" Font-Strikeout="False"

Font-Underline="False" Text="My Contact List - Windows Azure Table Example"></asp:Label>

<br />

<br />

<asp:Label ID="ContactLabel" runat="server" Text="Contact:"></asp:Label>

<br />

<table style="width:100%;">

<tr>

<td class="style2">

 </td>

<td>

 </td>

</tr>

<tr>

<td class="style2">

<asp:Label ID="lb\_firstname" runat="server" Text="First Name"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_firstname" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td class="style2">

<asp:Label ID="lb\_lastname" runat="server" Text="Last Name"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_lastname" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td class="style2">

<asp:Label ID="lb\_telnum" runat="server" Text="Telephone Number"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_telnum" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td class="style2">

<asp:Label ID="lb\_email" runat="server" Text="E-mail"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_email" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td class="style2">

 </td>

<td>

 </td>

</tr>

<tr>

<td class="style2">

<asp:Button ID="btn\_change" runat="server" onclick="btn\_change\_Click"

Text="Change" Width="105px" />

</td>

<td>

<asp:Button ID="btn\_add" runat="server" Height="20px" onclick="btn\_add\_Click"

Text="Add Contact" />

</td>

</tr>

<tr>

<td class="style2">

<asp:Label ID="lb\_status" runat="server"></asp:Label>

</td>

<td>

 </td>

</tr>

<tr>

<td class="style2">

 </td>

<td>

 </td>

</tr>

<tr>

<td class="style1" colspan="2">

<asp:GridView ID="contactGV" runat="server" CellPadding="4" ForeColor="#333333"

GridLines="None" onrowdeleting="contactGV\_RowDeleting"

onselectedindexchanged="contactGV\_SelectedIndexChanged">

<AlternatingRowStyle BackColor="White" />

<Columns>

<asp:CommandField ButtonType="Button" SelectText="Изменить"

ShowSelectButton="True" />

<asp:CommandField ButtonType="Button" ShowDeleteButton="True" />

</Columns>

<EditRowStyle BackColor="#2461BF" />

<FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />

<HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />

<PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />

<RowStyle BackColor="#EFF3FB" />

<SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor="#333333" />

<SortedAscendingCellStyle BackColor="#F5F7FB" />

<SortedAscendingHeaderStyle BackColor="#6D95E1" />

<SortedDescendingCellStyle BackColor="#E9EBEF" />

<SortedDescendingHeaderStyle BackColor="#4870BE" />

</asp:GridView></td>

</tr>

</table>

</div>

</form>

</body>

</html>

**Приложение Б. Класс Contact.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace WebRole1

{

class Contact : Microsoft.WindowsAzure.StorageClient.TableServiceEntity

{

public String FirstName { get; set; }

public String LastName { get; set; }

public String TelNumber { get; set; }

public String Email { get; set; }

}

}

**Приложение В. Класс ContactContext.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.StorageClient;

using System.Data.Services.Client;

namespace WebRole1

{

class ContactContext : TableServiceContext

{

public IQueryable<Contact> ContactData

{

get

{

return this.CreateQuery<Contact>("Contacts");

}

}

public ContactContext(Uri baseAddress, StorageCredentials credentials) : base(baseAddress.AbsoluteUri, credentials) { }

public void Add(Contact cnt)

{

this.AddObject("Contacts", cnt);

this.SaveChanges();

}

public void Delete(Contact cnt)

{

this.DeleteObject(cnt);

this.SaveChanges();

}

public void Update(Contact cnt)

{

this.UpdateObject(cnt);

this.SaveChanges();

}

}

}

**Приложение Г. Global.asax.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Security;

using System.Web.SessionState;

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.StorageClient;

using System.Data.Services.Client;

using Microsoft.WindowsAzure.ServiceRuntime;

namespace WebRole1

{

public class Global : System.Web.HttpApplication

{

void Application\_Start(object sender, EventArgs e)

{

CloudStorageAccount.SetConfigurationSettingPublisher(

(configName, configSettingPublisher) =>

{

var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);

configSettingPublisher(connectionString);

}

);

var account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

CloudTableClient \_tc = null;

\_tc = account.CreateCloudTableClient();

\_tc.CreateTableIfNotExist("Contacts");

}

void Application\_End(object sender, EventArgs e)

{

}

void Application\_Error(object sender, EventArgs e)

{

}

void Session\_Start(object sender, EventArgs e)

{

}

void Session\_End(object sender, EventArgs e)

{

}

}

}

**Приложение Д. AzureTable.aspx.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Data.Services.Client;

using Microsoft.WindowsAzure;

namespace WebRole1

{

public partial class AzureTable : System.Web.UI.Page

{

private CloudStorageAccount account = null;

private ContactContext context = null;

protected void Page\_Load(object sender, EventArgs e)

{

btn\_change.Visible = false;

account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

context = new ContactContext(account.TableEndpoint, account.Credentials);

contactGV.DataSource = context.ContactData;

contactGV.DataBind();

int i = 0;

foreach (TableCell cell in contactGV.HeaderRow.Cells)

{

if (cell.Text == "PartitionKey") { Session["pkindex"] = i; }

if (cell.Text == "RowKey") { Session["rkindex"] = i; }

i++;

}

}

protected void btn\_add\_Click(object sender, EventArgs e)

{

var statusMessage = String.Empty;

try

{

context.Add(new Contact { PartitionKey = "MyContacts",

RowKey = this.tb\_lastname.Text + " " + this.tb\_firstname.Text,

FirstName = tb\_firstname.Text, LastName = tb\_lastname.Text,

TelNumber = tb\_telnum.Text, Email = tb\_email.Text });

}

catch (DataServiceRequestException ex)

{

statusMessage = "Unable to connect to the table storage server. Please check that the service is running.<br>"

+ ex.Message;

}

lb\_status.Text = statusMessage;

contactGV.DataBind();

}

protected void contactGV\_RowDeleting(object sender, GridViewDeleteEventArgs e)

{

GridView g = (GridView)sender;

try

{

Contact c = (from contact in context.CreateQuery<Contact>("Contacts")

where contact.PartitionKey == g.Rows[e.RowIndex]. Cells[Convert.ToInt32(Session["pkindex"].ToString())].Text

&& contact.RowKey == g.Rows[e.RowIndex]. .Cells[Convert.ToInt32(Session["rkindex"].ToString())].Text

select contact).FirstOrDefault();

context.Delete(c);

}

catch(DataServiceRequestException ex)

{

lb\_status.Text = ex.Message;

}

g.DataBind();

}

protected void contactGV\_SelectedIndexChanged(object sender, EventArgs e)

{

GridView g = (GridView)sender;

int index = g.SelectedIndex;

Contact c = (from contact in context.CreateQuery<Contact>("Contacts")

where contact.PartitionKey == g.Rows[index].Cells[Convert.ToInt32(Session["pkindex"].ToString())].Text

&& contact.RowKey == g.Rows[index].Cells[Convert.ToInt32(Session["rkindex"].ToString())].Text

select contact).FirstOrDefault();

tb\_firstname.Text = c.FirstName;

tb\_lastname.Text = c.LastName;

tb\_email.Text = c.Email;

tb\_telnum.Text = c.TelNumber;

Session["index"] = index;

btn\_change.Visible = true;

}

protected void btn\_change\_Click(object sender, EventArgs e)

{

int index = Convert.ToInt32(Session["index"].ToString());

try

{

Contact c = (from contact in context.CreateQuery<Contact>("Contacts")

where contact.PartitionKey == contactGV.Rows[index].

Cells[Convert.ToInt32(Session["pkindex"].ToString())].Text

&& contact.RowKey == contactGV.Rows[index].

Cells[Convert.ToInt32(Session["rkindex"].ToString())].Text

select contact).FirstOrDefault();

c.FirstName = tb\_firstname.Text;

c.LastName = tb\_lastname.Text;

c.Email = tb\_email.Text;

c.TelNumber = tb\_telnum.Text;

context.Update(c);

}

catch (DataServiceRequestException a)

{

lb\_status.Text = a.Message;

}

contactGV.DataBind();

}

}

}

**Практическая работа №6. Работа с Windows Azure Blob**

В качестве примера рассмотрим работу простого веб - приложения для загрузки изображений в хранилище Windows Azure Blob:

* подготовка приложения;
* загрузка и отображение изображений;
* удаление сущностей;
* копирование сущностей.

Создайте проект облачной службы и добавьте веб - роль в решение.

В свойствах веб - роли определим строку подключения к эмулятору хранилища Azure.

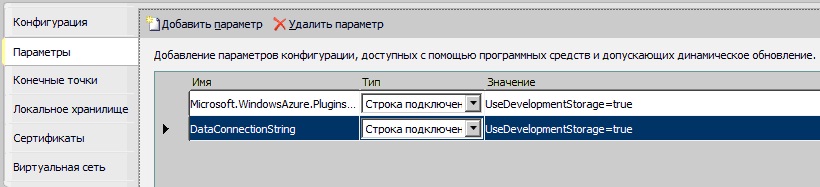


Рисунок 6.1 - Хранилище Azure

Также нам понадобится строковый параметр ContainerName, определяющий имя Blob - контейнера для изображений.

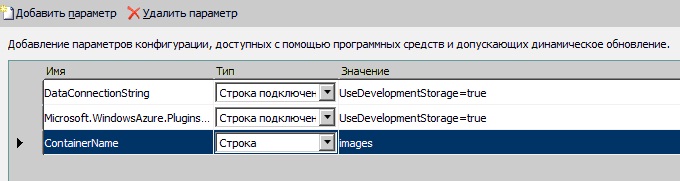


Рисунок 6.2 - Параметры контейнера

**Задание 1. Подготовка приложени**

Создадим веб - форму AzureBlobSamle.aspx и добавим ссылки в C# класс данной страницы

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.ServiceRuntime;

using Microsoft.WindowsAzure.StorageClient;

using System.Configuration;

Далее, поскольку Blob - контейнер должен быть доступным для анонимных пользователей необходимо создать метод, подтверждающий наличие контейнера:

private void EnsureContainerExists()

{

var container = GetContainer();

container.CreateIfNotExist();

var permissions = container.GetPermissions();

permissions.PublicAccess = BlobContainerPublicAccessType.Container;

container.SetPermissions(permissions);

}

Как видно, в данном методе пРисунокутствует ссылка на несуществующий пока GetConteiner(), который должен возвращать Blob-контейнер для осуществления операций с Blob-объектами.

private CloudBlobContainer GetContainer()

{

CloudStorageAccount.SetConfigurationSettingPublisher(

(configName, configSettingPublisher) =>

{

var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);

configSettingPublisher(connectionString);

}

);

var account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

var client = account.CreateCloudBlobClient();

return client.GetContainerReference(RoleEnvironment.GetConfigurationSettingValue("ContainerName"));

}

**Интерфейс**

Теперь необходимо заняться самой формой. На форме буду пРисунокутствовать текстовые поля, задающие параметры изображения и диалог загрузки файла.

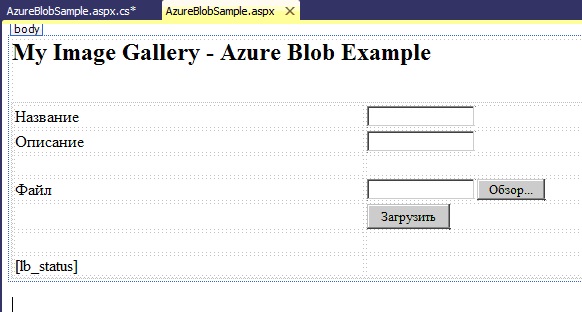


Рисунок 6.3 - Загрузка файла

asp - код веб формы:

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="AzureBlobSample.aspx.cs"

Inherits="WebRole1.AzureBlobSample" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="X-Large"

Text="My Image Gallery - Azure Blob Example"></asp:Label>

<br />

<br />

<table style="width:100%;">

<tr>

<td>

<asp:Label ID="lb\_name" runat="server" Text="Название"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_label" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td>

<asp:Label ID="lb\_desc" runat="server" Text="Описание"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_desc" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td>

 </td>

<td>

 </td>

</tr>

<tr>

<td>

<asp:Label ID="lb\_file" runat="server" Text="Файл"></asp:Label>

</td>

<td>

<asp:FileUpload ID="fu\_upload" runat="server" />

</td>

</tr>

<tr>

<td>

 </td>

<td>

<asp:Button ID="btn\_upload" runat="server" onclick="btn\_upload\_Click"

Text="Загрузить" />

</td>

</tr>

<tr>

<td>

 </td>

<td>

 </td>

</tr>

<tr>

<td>

<asp:Label ID="lb\_status" runat="server"></asp:Label>

</td>

<td>

<asp:ListView ID="lv\_images" runat="server">

<LayoutTemplate>

<asp:PlaceHolder ID="itemPlaceholder" runat="server" />

</LayoutTemplate>

<EmptyDataTemplate>

<h2>No Data Available</h2>

</EmptyDataTemplate>

<ItemTemplate>

<div class="item">

<ul style="width:40em;float:left;clear:left" >

<asp:Repeater ID="blobMetadata" runat="server">

<ItemTemplate>

<li><%# Eval("Name") %><span><%# Eval("Value") %></span></li>

</ItemTemplate>

</asp:Repeater>

<li>

</li>

</ul>

<img src="<%# Eval("Uri") %>" alt="<%# Eval("Uri") %>" style="float:left"/>

</div>

</ItemTemplate>

</asp:ListView>

</td>

</tr>

<tr>

<td>

<br />

</td>

<td>

 </td>

</tr>

</table>

</div>

</form>

</body>

</html>

Отметим, что элемент управления, при помощи которого мы будем отображать имеющиеся изображения - это ListView с идентификатором lv\_images. Для того чтобы сделать возможным повторени вышеуказанного шаблона для каждого элемента списка мы использовали Repeater (подробнее см. в списке вспомогательных материалов)

Теперь в метод Page\_Load добавим код, который при первой загрузке нашей страницы будет проверять наличие контейнера и выводить сообщение о возможных ошибках и привязывать lv\_images к источнику данных, вызовом метода RefreshGallery:

try

{

if (!IsPostBack)

{

this.EnsureContainerExists();

}

this.RefreshGallery();

}

catch (System.Net.WebException we)

{

lb\_status.Text = "Network error: " + we.Message;

if (we.Status == System.Net.WebExceptionStatus.ConnectFailure)

{

lb\_status.Text += "<br />Please check if the blob service is running at " +

ConfigurationManager.AppSettings["storageEndpoint"];

}

}

catch (StorageException se)

{

Console.WriteLine("Storage service error: " + se.Message);

}

Метод RefreshGallery - привязывает lv\_images к источнику данных(контейнеру бинарных объектов):

private void RefreshGallery()

{

lv\_images.DataSource =

this.GetContainer().ListBlobs(new BlobRequestOptions()

{

UseFlatBlobListing = true,

BlobListingDetails = BlobListingDetails.All

});

lv\_images.DataBind();

}

Запустим приложение и убедимся, что оно выполняется верно.

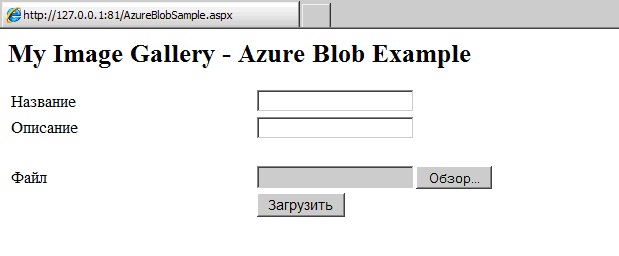


Рисунок 6.4 - Запуск приложения

**Задание 2. Загрузка изображений**

Необходимо добавить следующую ссылку в AzureBlobSample.cs:

using System.Collections.Specialized;

Создадим метод, сохраняющий бинарный объект и задающий его метаданные:

private void SaveImage(string id, string name, string description, string tags,

string fileName, string contentType, byte[] data)

{

// Создание BLob - объекта в контейнере

var blob = this.GetContainer().GetBlobReference(name);

blob.Properties.ContentType = contentType;

// определение метаданных, добавление метаданных blob-объекту и загрузка данных

try

{

blob.UploadFromStream(fu\_upload.FileContent);

blob.Metadata["Id"] = id;

blob.Metadata["Filename"] = fileName;

blob.Metadata["ImageName"] = String.IsNullOrEmpty(name) ? "unknown" : name;

blob.Metadata["Description"] = String.IsNullOrEmpty(description) ? "unknown" : description;

blob.SetMetadata();

lv\_images.DataBind();

}

catch (Exception ex)

{ lb\_status.Text = ex.Message; }

}

}

Остается только написать метод - обрабатывающий событие нажатия кнопки btn\_upload:

protected void btn\_upload\_Click(object sender, EventArgs e)

{

if (fu\_upload.HasFile)

{

lb\_status.Text = "Inserted [" + fu\_upload.FileName + "] -

Content Type [" + fu\_upload.PostedFile.ContentType + "] -

Length [" + fu\_upload.PostedFile.ContentLength + "]";

this.SaveImage(Guid.NewGuid().ToString(),

tb\_name.Text,

tb\_desc.Text,

fu\_upload.FileName,

fu\_upload.PostedFile.ContentType,

fu\_upload.FileBytes

);

RefreshGallery();

}

else

lb\_status.Text = "No image file";

}

Теперь можно проверить функцию загрузки изображения, выбрав его через диалог загрузки файла.

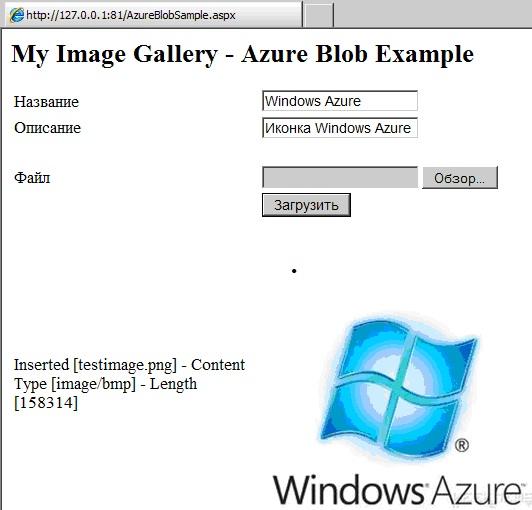


Рисунок 6.5 - Функция загрузки изображения

Надо сказать, что в таком виде информация об изображении малоинформативная. Blob, как мы знаем, может хранить привязанные к нему метаданные. Добавим в наше решение функционал, позволяющий получать и отображать метаданные, ассоциированные с конкретным изображением.

Для начала, необходимо создать метод обрабатывающий событие ItemDataBound элемента управления lv\_images:

protected void lv\_images\_ItemDataBound(object sender, ListViewItemEventArgs e)

{

if (e.Item.ItemType == ListViewItemType.DataItem)

{

var metadataRepeater = e.Item.FindControl("blobMetadata") as Repeater;

var blob = ((ListViewDataItem)(e.Item)).DataItem as CloudBlob;

if (blob != null)

{

if (blob.SnapshotTime.HasValue)

{

var delBtn = e.Item.FindControl("deleteBlob") as LinkButton;

if (delBtn != null) delBtn.Text = "Delete Snapshot";

var snapshotBtn = e.Item.FindControl("SnapshotBlob") as LinkButton;

if (snapshotBtn != null) snapshotBtn.Visible = false;

}

if (metadataRepeater != null)

{

metadataRepeater.DataSource = from key in blob.Metadata.AllKeys

select new

{

Name = key,

Value = blob.Metadata[key]

};

metadataRepeater.DataBind();

}

}

}

}

Теперь запустим наше приложение и получим следующее:

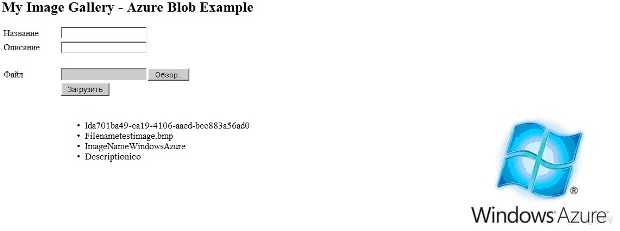
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/20/files/6_6.jpg)

Рисунок 6.6 - Запуск приложения

Слева от изображения появился список метаданных.

**Задание 3 Удаление сущностей**

Для начала необходимо добавить следующий asp - код для lv\_images:

...

div class="item">

<ul style="width:40em;float:left;clear:left" >

<asp:Repeater ID="blobMetadata" runat="server">

<ItemTemplate>

<li><%# Eval("Name") %><span><%# Eval("Value") %></span></li>

</ItemTemplate>

</asp:Repeater>

<li>

<asp:LinkButton ID="deleteBlob"

OnClientClick="return confirm('Delete image?');"

CommandName="Delete"

CommandArgument='<%# Eval("Uri")%>'

runat="server" Text="Удалить" oncommand="OnDeleteImage" />

...

Это необходимо для формировании кнопки "Delete" в рамках нашего ListView. Кроме того, этот кол содержит ссылку на метод, выполняющийся при нажатии кнопки.

Метод OnDeleteImage:

protected void OnDeleteImage(object sender, CommandEventArgs e)

{

try

{

if (e.CommandName == "Delete")

{

var blobUri = (string)e.CommandArgument;

var blob = this.GetContainer().GetBlobReference(blobUri);

blob.DeleteIfExists();

}

}

catch (StorageClientException se)

{

lb\_status.Text = "Storage client error: " + se.Message;

}

catch (Exception) { }

RefreshGallery();

}

Данный метод определяет бинарный объект и удаляет его.

Запустите приложение и протестируйте функцию удаления.

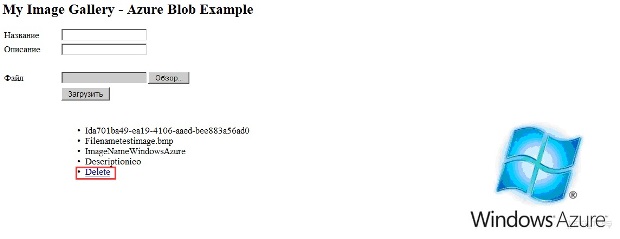
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/20/files/6_7.jpg)

Рисунок 6.7 - Запуск приложения

**Задание 4 Копирование сущностей**

Копирование бинарных объектов также поддерживается Windows Azure. В рамках данного задания мы рассмотрим этот механизм.

Для формирования кнопки "Копировать" в списке имеющихся изображений, добавим следующий asp - код для lv\_images :

...

<div class="item">

<ul style="width:40em;float:left;clear:left" >

<asp:Repeater ID="blobMetadata" runat="server">

<ItemTemplate>

<li><%# Eval("Name") %><span><%# Eval("Value") %></span></li>

</ItemTemplate>

</asp:Repeater>

<li>

...

<asp:LinkButton ID="CopyBlob"

OnClientClick="return confirm('Copy image?');"

CommandName="Copy"

CommandArgument='<%# Eval("Uri")%>'

runat="server" Text="Copy" oncommand="OnCopyImage" />

...

Как и в предыдущем задании, создадим метод OnCopyImage:

protected void OnCopyImage(object sender, CommandEventArgs e)

{

if (e.CommandName == "Copy")

{

// Prepare an Id for the copied blob

var newId = Guid.NewGuid();

// получение исходного объекта

var blobUri = (string)e.CommandArgument;

var srcBlob = this.GetContainer().GetBlobReference(blobUri);

// создание нового бинарного объекта

var newBlob = this.GetContainer().GetBlobReference(newId.ToString());

// копирование содержимого исходного объекта

newBlob.CopyFromBlob(srcBlob);

// получаем метаданные для нового объекта

newBlob.FetchAttributes(new BlobRequestOptions { BlobListingDetails = BlobListingDetails.Metadata });

// изменение метаданных нового объекта, чтобы показать, что это копия

newBlob.Metadata["ImageName"] = "Copy of \"" + newBlob.Metadata["ImageName"]

+ "\"";

newBlob.Metadata["Id"] = newId.ToString();

newBlob.SetMetadata();

RefreshGallery();

}

}

Проверим функционал:

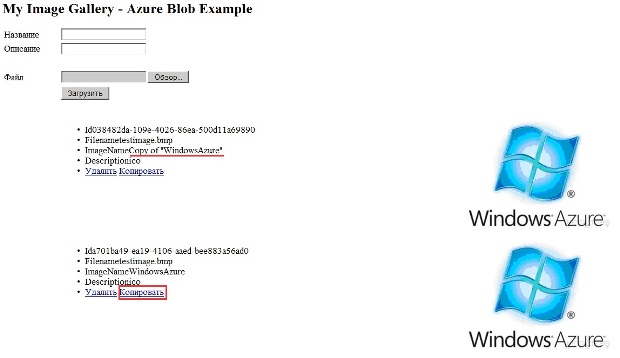
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/20/files/6_8.jpg)

Рисунок 6.8 - Запуск приложения

В случае, если выполнение задания вызвало сложности и затруднения, в приложениях к данной практической работе вы найдете итоговый программный код в том виде, в котором он необходим для последнего задания.

**Список вспомогательных материалов**

**Работа с Windows Azure Blob**

1. <http://msdn.microsoft.com/en-us/wazplatformtrainingcourse_exploringwindowsazurestoragevs2010_topic3>
2. <http://blogs.msdn.com/b/jnak/archive/2010/01/11/walkthrough-windows-azure-blob-storage-nov-2009-and-later.aspx>
3. <http://blogs.msdn.com/b/jnak/archive/2008/10/29/walkthrough-simple-blob-storage-sample.aspx>
4. <http://wotudo.net/blogs/wotudo/archive/2010/02/16/copying-files-to-windows-azure-blob-storage.aspx>

**Repeater**

1. <http://msdn.microsoft.com/ru-ru/library/system.web.ui.webcontrols.repeater.aspx>
2. <http://www.w3schools.com/ASPNET/aspnet_repeater.asp>
3. <http://articles.sitepoint.com/article/asp-net-repeater-control>

**Приложение А asp - код страницы AzureBlobSample.aspx**

<%@ Page Language="C#" AutoEventWireup="true"

CodeBehind="AzureBlobSample.aspx.cs" Inherits="WebRole1.AzureBlobSample" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

<style type="text/css">

.style1

{

height: 25px;

}

</style>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="X-Large"

Text="My Image Gallery - Azure Blob Example"></asp:Label>

<br />

<br />

<table style="width:100%;">

<tr>

<td>

<asp:Label ID="lb\_name" runat="server" Text="Название"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_name" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td>

<asp:Label ID="lb\_desc" runat="server" Text="Описание"></asp:Label>

</td>

<td>

<asp:TextBox ID="tb\_desc" runat="server"></asp:TextBox>

</td>

</tr>

<tr>

<td>

 </td>

<td>

 </td>

</tr>

<tr>

<td class="style1">

<asp:Label ID="lb\_file" runat="server" Text="Файл"></asp:Label>

</td>

<td class="style1">

<asp:FileUpload ID="fu\_upload" runat="server" />

</td>

</tr>

<tr>

<td>

 </td>

<td>

<asp:Button ID="btn\_upload" runat="server" onclick="btn\_upload\_Click"

Text="Загрузить" />

</td>

</tr>

<tr>

<td>

<asp:Label ID="lb\_status" runat="server"></asp:Label>

</td>

<td>

 </td>

</tr>

<tr>

<td>

 </td>

<td>

<asp:ListView ID="lv\_images" runat="server"

onitemdatabound="lv\_images\_ItemDataBound">

<LayoutTemplate>

<asp:PlaceHolder ID="itemPlaceholder" runat="server" />

</LayoutTemplate>

<EmptyDataTemplate>

<h2>No Data Available</h2>

</EmptyDataTemplate>

<ItemTemplate>

<div class="item">

<ul style="width:40em;float:left;clear:left" >

<asp:Repeater ID="blobMetadata" runat="server">

<ItemTemplate>

<li><%# Eval("Name") %><span><%# Eval("Value") %></span></li>

</ItemTemplate>

</asp:Repeater>

<li>

<asp:LinkButton ID="deleteBlob"

OnClientClick="return confirm('Delete image?');"

CommandName="Delete"

CommandArgument='<%# Eval("Uri")%>'

runat="server" Text="Удалить" oncommand="OnDeleteImage" />

<asp:LinkButton ID="CopyBlob"

OnClientClick="return confirm('Copy image?');"

CommandName="Copy"

CommandArgument='<%# Eval("Uri")%>'

runat="server" Text="Копировать" oncommand="OnCopyImage" />

</li>

</ul>

<img src="<%# Eval("Uri") %>" alt="<%# Eval("Uri") %>" style="float:left"/>

</div>

</ItemTemplate>

</asp:ListView>

</td>

</tr>

<tr>

<td>

<br />

</td>

<td>

 </td>

</tr>

</table>

</div>

</form>

</body>

</html>

**Приложение Б AzureBlobSample.aspx.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.ServiceRuntime;

using Microsoft.WindowsAzure.StorageClient;

using System.Configuration;

using System.Collections.Specialized;

namespace WebRole1

{

public partial class AzureBlobSample : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

try

{

if (!IsPostBack)

{

this.EnsureContainerExists();

}

this.RefreshGallery();

}

catch (System.Net.WebException we)

{

lb\_status.Text = "Network error: " + we.Message;

if (we.Status == System.Net.WebExceptionStatus.ConnectFailure)

{

lb\_status.Text += "<br />Please check if the blob service is running at " +

ConfigurationManager.AppSettings["storageEndpoint"];

}

}

catch (StorageException se)

{

Console.WriteLine("Storage service error: " + se.Message);

}

}

private void RefreshGallery()

{

lv\_images.DataSource =

this.GetContainer().ListBlobs(new BlobRequestOptions()

{

UseFlatBlobListing = true,

BlobListingDetails = BlobListingDetails.All

});

lv\_images.DataBind();

}

private CloudBlobContainer GetContainer()

{

CloudStorageAccount.SetConfigurationSettingPublisher(

(configName, configSettingPublisher) =>

{

var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);

configSettingPublisher(connectionString);

}

);

var account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

var client = account.CreateCloudBlobClient();

return client.GetContainerReference(RoleEnvironment.GetConfigurationSettingValue("ContainerName"));

}

private void EnsureContainerExists()

{

var container = GetContainer();

container.CreateIfNotExist();

var permissions = container.GetPermissions();

permissions.PublicAccess = BlobContainerPublicAccessType.Container;

container.SetPermissions(permissions);

}

private void SaveImage(string id, string name, string description, string fileName, string contentType, byte[] data)

{

var blob = this.GetContainer().GetBlobReference(name);

blob.Properties.ContentType = contentType;

try

{

blob.UploadFromStream(fu\_upload.FileContent);

blob.Metadata["Id"] = id;

blob.Metadata["Filename"] = fileName;

blob.Metadata["ImageName"] = String.IsNullOrEmpty(name) ? "unknown" : name;

blob.Metadata["Description"] = String.IsNullOrEmpty(description) ? "unknown" : description;

blob.SetMetadata();

lv\_images.DataBind();

}

catch (Exception ex)

{ lb\_status.Text = ex.Message; }

}

protected void btn\_upload\_Click(object sender, EventArgs e)

{

if (fu\_upload.HasFile)

{

lb\_status.Text = "Inserted [" + fu\_upload.FileName + "] - Content Type [" +

fu\_upload.PostedFile.ContentType + "] - Length [" + fu\_upload.PostedFile.ContentLength + "]";

this.SaveImage(Guid.NewGuid().ToString(),

tb\_name.Text,

tb\_desc.Text,

fu\_upload.FileName,

fu\_upload.PostedFile.ContentType,

fu\_upload.FileBytes

);

RefreshGallery();

}

else

lb\_status.Text = "No image file";

}

protected void lv\_images\_ItemDataBound(object sender, ListViewItemEventArgs e)

{

if (e.Item.ItemType == ListViewItemType.DataItem)

{

var metadataRepeater = e.Item.FindControl("blobMetadata") as Repeater;

var blob = ((ListViewDataItem)(e.Item)).DataItem as CloudBlob;

if (blob != null)

{

if (blob.SnapshotTime.HasValue)

{

var delBtn = e.Item.FindControl("deleteBlob") as LinkButton;

if (delBtn != null) delBtn.Text = "Delete Snapshot";

var snapshotBtn = e.Item.FindControl("SnapshotBlob") as LinkButton;

if (snapshotBtn != null) snapshotBtn.Visible = false;

}

if (metadataRepeater != null)

{

metadataRepeater.DataSource = from key in blob.Metadata.AllKeys

select new

{

Name = key,

Value = blob.Metadata[key]

};

metadataRepeater.DataBind();

}

}

}

}

protected void OnDeleteImage(object sender, CommandEventArgs e)

{

try

{

if (e.CommandName == "Delete")

{

var blobUri = (string)e.CommandArgument;

var blob = this.GetContainer().GetBlobReference(blobUri);

blob.DeleteIfExists();

}

}

catch (StorageClientException se)

{

lb\_status.Text = "Storage client error: " + se.Message;

}

catch (Exception) { }

RefreshGallery();

}

protected void OnCopyImage(object sender, CommandEventArgs e)

{

if (e.CommandName == "Copy")

{

// Prepare an Id for the copied blob

var newId = Guid.NewGuid();

// получение исходного объекта

var blobUri = (string)e.CommandArgument;

var srcBlob = this.GetContainer().GetBlobReference(blobUri);

// создание нового бинарного объекта

var newBlob = this.GetContainer().GetBlobReference(newId.ToString());

// копирование содержимого исходного объекта

newBlob.CopyFromBlob(srcBlob);

// получаем метаданные для нового объекта

newBlob.FetchAttributes(new BlobRequestOptions { BlobListingDetails = BlobListingDetails.Metadata });

// изменение метаданных нового объекта, чтобы показать, что это копия

newBlob.Metadata["ImageName"] = "Copy of \"" +

newBlob.Metadata["ImageName"] + "\"";

newBlob.Metadata["Id"] = newId.ToString();

newBlob.SetMetadata();

RefreshGallery();

}

}

}

}

**Практическая работ №7. Работа с Windows AzureQueue**

Создайте проект облачной службы и добавьте веб - роль и рабочую роль в решение.

В свойствах ролей определим строку подключения к эмулятору хранилища Azure.

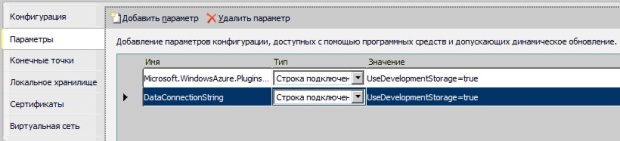
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/23/files/7_1.jpg)

Рисунок 7.1 - Хранилище Azure

**Задание 1**

**Отправка сообщения в очередь**

Создадим веб - форму AzureQueueSample со элементами управления - текстовое поле и кнопка:

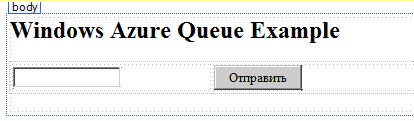


Рисунок 7.2 - Хранилище Azure

asp - код веб формы:

<%@ Page Language="C#" AutoEventWireup="true"

CodeBehind="AzureQueueSample.aspx.cs" Inherits="WebRole1.AzureQueueSample" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

<style type="text/css">

.style1

{

width: 197px;

}

</style>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="X-Large"

Text="Windows Azure Queue Example"></asp:Label>

<br />

<br />

<table style="width:100%;">

<tr>

<td class="style1">

<asp:TextBox ID="tb\_message" runat="server"></asp:TextBox>

</td>

<td>

<asp:Button ID="btn\_send" runat="server" onclick="btn\_send\_Click"

Text="Отправить" />

</td>

</tr>

</table>

<br />

</div>

</form>

</body>

</html>

Добавим следующие ссылки в AzureQueueSample.aspx.cs:

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.StorageClient;

using Microsoft.WindowsAzure.ServiceRuntime;

Создадим метод, обрабатывающие событие нажатия на кнопку btn\_send :

protected void btn\_send\_Click(object sender, EventArgs e)

{

//определяем параметры учетной записи и клиента для создания очереди и сообщения

var storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

var queueClient = storageAccount.CreateCloudQueueClient();

var queue = queueClient.GetQueueReference("messagequeue");

//создаем очередь

queue.CreateIfNotExist();

//создаем сообщение, источник - текстовое поле tb\_message

var msg = new CloudQueueMessage(tb\_message.Text);

//добавляем сообщение в очередь

queue.AddMessage(msg);

tb\_message.Text = string.Empty;

}

На этом работа с формой закончена.

**Извлечение сообщения из очереди**

Перейдите с помощью обозревателя решений к файлу Global.asax.cs и добать следующий код в метод ApplicationStart :

void Application\_Start(object sender, EventArgs e)

{

CloudStorageAccount.SetConfigurationSettingPublisher((configName, configSetter) =>

{ configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));

});

}

Затем перейдите к файлу WorkerRole.cs нашей рабочей роли. Добавьте следующие ссылки:

using System.Diagnostics;

using System.Threading;

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.Diagnostics;

using Microsoft.WindowsAzure.ServiceRuntime;

using Microsoft.WindowsAzure.StorageClient

Измените метод OnStart в соответствие со следующим кодом:

public override bool OnStart()

{

ServicePointManager.DefaultConnectionLimit = 12;

CloudStorageAccount.SetConfigurationSettingPublisher((configName, configSetter) =>

{ configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));

});

return base.OnStart();

}

Затем приведите метод Run в соответствие с:

public override void Run()

{

var storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

// получение ссылок на сообщения очереди

var queueClient = storageAccount.CreateCloudQueueClient();

var queue = queueClient.GetQueueReference("messagequeue");

// получение сообщение и передача его в лог эмулятора Compute Emulator

while (true)

{

Thread.Sleep(10000);

if (queue.Exists())

{

var msg = queue.GetMessage();

if (msg != null)

{

Trace.TraceInformation(string.Format("Сообщение '{0}' обработано.", msg.AsString));

queue.DeleteMessage(msg);

}

}

}

}

Запустите приложение, заполните текстовое поле и нажмите кнопку "Отправить":



Рисунок 7.3 - Запуск приложения

Затем, откройте интерфейс Compute Emulator:

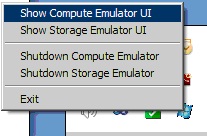


Рисунок 7.4 - Интерфейс Compute Emulator

И откройте лог рабочей роли, чтобы убедиться в работоспособности приложения:

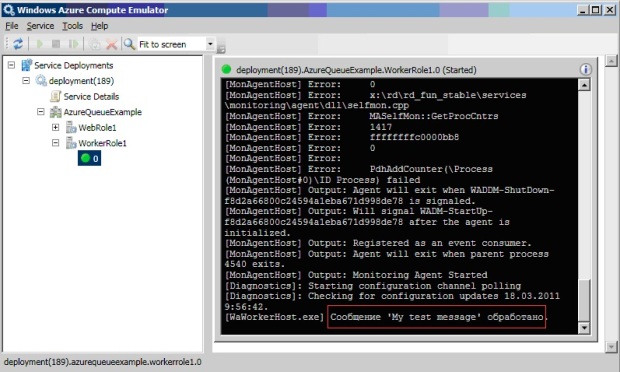
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/23/files/7_5.jpg)

Рисунок 7.5 - Работа приложения

**Задание 2**

Для следующего задания можно "усовершенствовать" предыдущую веб - форму, но мы предпочли создать новую - AzureQueueSample2.aspx:

[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/23/files/7_6.jpg)

Рисунок 7.6 - Работа приложения

asp - код веб формы для второго задания:

<%@ Page Language="C#" AutoEventWireup="true"

CodeBehind="AzureQueueSample2.aspx.cs" Inherits="WebRole1.AzureQueueSample2" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

<style type="text/css">

.style1

{

width: 268px;

}

.style2

{

width: 247px;

}

</style>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Label ID="Label4" runat="server" Font-Bold="True" Font-Size="X-Large"

Text="Windows Azure Queue - Second Example"></asp:Label>

<br />

<table style="width:100%;">

<tr>

<td class="style1">

<asp:Label id="lb\_inert" Text="Добавить в очередь сообщение:" runat="server" />

</td>

<td class="style2">

<asp:TextBox id="tb\_insert" runat="server" />

</td>

<td>

<asp:Button ID="bnt\_addmessage" Text="Добавить сообщение" runat="server"

onclick="bnt\_addmessage\_Click" Width="231px" />

</td>

</tr>

<tr>

<td class="style1">

<asp:Label id="lb\_binarydata" Text="Добавить, как двойчный объект:" runat="server" />

</td>

<td class="style2">

<asp:FileUpload ID="fu\_insertbinary" runat="server" />

</td>

<td>

<asp:Button ID="btn\_addbinary" Text="Добавить двоичные данные" runat="server"

onclick="btn\_addbinary\_Click" />

</td>

</tr>

<tr>

<td class="style1">

<asp:Button ID="btn\_getmessage" Text="Получить сообщение" runat="server"

onclick="btn\_getmessage\_Click" />

</td>

<td class="style2">

<asp:Label id="lb\_retrievedmessage" runat="server" />

</td>

<td>

 </td>

</tr>

</table>

<br />

<asp:Repeater id="Repeater1" runat="server">

<HeaderTemplate>

<table border="1">

<tr><td><b>Queue Messages</b></td></tr>

</HeaderTemplate>

<ItemTemplate>

<tr>

<td> <asp:Label ID="Label1" runat="server"

Text="<%# Container.DataItem %>" /> </td>

</tr>

</ItemTemplate>

<FooterTemplate>

</table>

</FooterTemplate>

</asp:Repeater>

</div>

</form>

</body>

</html>

Наша задача состоит в следующем:

1. Реализовать функционал создания очереди
2. Добавить сообщение в очередь
3. Добавить в очередь файл, в виде двойного объекта
4. Отобразить сообщения очереди
5. Получить сообщение из очереди

Добавьте следующие ссылки в AzureQueueSample2.aspx.cs :

using System.IO;

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.StorageClient;

using Microsoft.WindowsAzure.ServiceRuntime;

Также определим переменные учетной записи, очереди и клиента до метода PageLoad:

CloudStorageAccount storageAccount = null;

CloudQueue cloudQueue = null;

CloudQueueClient queueClient = null;

Для выполнения первого пункта, добавим следующий код в метод PageLoad :

// определение контекста

this.storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

this.queueClient = storageAccount.CreateCloudQueueClient();

// создание очереди

this.cloudQueue = this.queueClient.GetQueueReference("myqueue");

this.cloudQueue.CreateIfNotExist();

this.DisplayMesssages();

Для добавления сообщения в очередь, создадим метод, обрабатывающий событие нажатия кнопки btn\_addmessage :

protected void bnt\_addmessage\_Click(object sender, EventArgs e)

{

CloudQueueMessage msg = new CloudQueueMessage(this.tb\_insert.Text);

this.cloudQueue.AddMessage(msg);

this.DisplayMesssages();

}

Для отображения содержимого очереди, создадим метод DisplayMessages:

private void DisplayMesssages()

{

// получение первых пяти сообщений, без удаления их из очереди

var msgs = this.cloudQueue.PeekMessages(5);

var cloudList = new List<string>();

foreach (var msg in msgs)

{

cloudList.Add("Message ID: " + msg.Id + "; Message: " + msg.AsString +

"; Message insertion time: " + msg.InsertionTime);

}

// привязка к источнику данных

this.Repeater1.DataSource = cloudList;

this.Repeater1.DataBind();

}

Для добавления очередь файла, создадим метод обрабатывающий нажатие кнопки btn\_addbinary:

protected void btn\_addbinary\_Click(object sender, EventArgs e)

{

// файл не должен превышать размер - лимит сообщения - 8Кб

CloudQueueMessage msg = new CloudQueueMessage(File.ReadAllBytes(fu\_insertbinary.FileName));

this.cloudQueue.AddMessage(msg);

this.DisplayMesssages();

}

Осталось только написать метод, для получения сообщения из очереди, обратите внимание, что при этом сообщение из очереди удаляется, а получить можно только первое сообщение из очереди:

protected void Page\_Load(object sender, EventArgs e)

{

this.storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

this.queueClient = storageAccount.CreateCloudQueueClient();

this.cloudQueue = this.queueClient.GetQueueReference("myqueue");

this.cloudQueue.CreateIfNotExist();

this.DisplayMesssages();

}

Теперь запустим приложение и протестируем функционал.

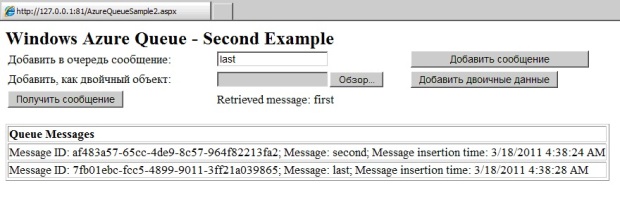
[](http://www.intuit.ru/EDI/08_07_13_1/1373232057-31070/tutorial/992/objects/23/files/7_7.jpg)

Рисунок 7.7 - Работа приложения

Обратим ваше внимание на то, что файл, который можно добавить в очередь должен соответствовать следующим условиям:

* размер файла не должен превышать 8Кб
* файл должен находиться: [путь к проекту AzureQueueExample]\bin\Debug\CloudService1.csx\roles\WebRole1

В случае, если выполнение задания вызвало сложности и затруднения, в приложениях к данной практической работе вы найдете итоговый программный код в том виде, в котором он необходим для последнего задания.

**Список вспомогательных материалов**

**Работа с Windows Azure Queue**

1. <http://msdn.microsoft.com/en-us/wazplatformtrainingcourse_exploringwindowsazurestoragevs2010_topic4>
2. <http://dotnetbyexample.blogspot.com/2009/07/storing-objects-as-compressed-messages.html>
3. <http://soumya.wordpress.com/2010/05/20/azure-simplified-part-4-using-azure-queue-storage/>
4. <http://www.dotnetconsult.co.uk/weblog2/PermaLink,guid,c24f8177-f6bf-479b-bd5c-d532e0e40272.aspx>

**Приложение А AzureQueueSample.aspx.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.StorageClient;

using Microsoft.WindowsAzure.ServiceRuntime;

namespace WebRole1

{

public partial class AzureQueueSample : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

}

protected void btn\_send\_Click(object sender, EventArgs e)

{

var storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

var queueClient = storageAccount.CreateCloudQueueClient();

var queue = queueClient.GetQueueReference("messagequeue");

queue.CreateIfNotExist();

var msg = new CloudQueueMessage(tb\_message.Text);

queue.AddMessage(msg);

tb\_message.Text = string.Empty;

}

}

}

**Приложение Б Global.asax.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Security;

using System.Web.SessionState;

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.ServiceRuntime;

namespace WebRole1

{

public class Global : System.Web.HttpApplication

{

void Application\_Start(object sender, EventArgs e)

{

CloudStorageAccount.SetConfigurationSettingPublisher((configName, configSetter) =>

{ configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));

});

}

void Application\_End(object sender, EventArgs e)

{

}

void Application\_Error(object sender, EventArgs e)

{

}

void Session\_Start(object sender, EventArgs e)

{

}

void Session\_End(object sender, EventArgs e)

{

}

}

}

**Приложение В Worker.cs**

using System.Diagnostics;

using System.Linq;

using System.Net;

using System.Threading;

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.Diagnostics;

using Microsoft.WindowsAzure.ServiceRuntime;

using Microsoft.WindowsAzure.StorageClient;

namespace WorkerRole1

{

public class WorkerRole : RoleEntryPoint

{

public override void Run()

{

var storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

var queueClient = storageAccount.CreateCloudQueueClient();

var queue = queueClient.GetQueueReference("messagequeue");

while (true)

{

Thread.Sleep(10000);

if (queue.Exists())

{

var msg = queue.GetMessage();

if (msg != null)

{

Trace.TraceInformation(string.Format("Сообщение '{0}' обработано.", msg.AsString));

queue.DeleteMessage(msg);

}

}

}

}

public override bool OnStart()

{

ServicePointManager.DefaultConnectionLimit = 12;

CloudStorageAccount.SetConfigurationSettingPublisher((configName, configSetter) =>

{ configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));

});

return base.OnStart();

}

}

}

**Приложение Г AzureQueueSample2.aspx.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.IO;

using Microsoft.WindowsAzure;

using Microsoft.WindowsAzure.StorageClient;

using Microsoft.WindowsAzure.ServiceRuntime;

namespace WebRole1

{

public partial class AzureQueueSample2 : System.Web.UI.Page

{

CloudStorageAccount storageAccount = null;

CloudQueue cloudQueue = null;

CloudQueueClient queueClient = null;

protected void Page\_Load(object sender, EventArgs e)

{

this.storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

this.queueClient = storageAccount.CreateCloudQueueClient();

this.cloudQueue = this.queueClient.GetQueueReference("myqueue");

this.cloudQueue.CreateIfNotExist();

this.DisplayMesssages();

}

private void DisplayMesssages()

{

var msgs = this.cloudQueue.PeekMessages(5);

var cloudList = new List<string>();

foreach (var msg in msgs)

{

cloudList.Add("Message ID: " + msg.Id + "; Message: " + msg.AsString +

"; Message insertion time: " + msg.InsertionTime);

}

this.Repeater1.DataSource = cloudList;

this.Repeater1.DataBind();

}

protected void bnt\_addmessage\_Click(object sender, EventArgs e)

{

CloudQueueMessage msg = new CloudQueueMessage(this.tb\_insert.Text);

this.cloudQueue.AddMessage(msg);

this.DisplayMesssages();

}

protected void btn\_addbinary\_Click(object sender, EventArgs e)

{

CloudQueueMessage msg = new CloudQueueMessage(File.ReadAllBytes(fu\_insertbinary.FileName));

this.cloudQueue.AddMessage(msg);

this.DisplayMesssages();

}

protected void btn\_getmessage\_Click(object sender, EventArgs e)

{

CloudQueueMessage msg = this.cloudQueue.GetMessage();

this.lb\_retrievedmessage.Text = "Retrieved message: " + msg.AsString;

this.cloudQueue.DeleteMessage(msg);

this.DisplayMesssages();

}

}

}

**Практическая работа №8. Mmicrosoft .Net Service Bus: обзор, обмен сообщениями, управление доступом**

**Обзоp**

Основной задачей .Net Service Bus является обеспечение коммуникаций между приложениями. Использование данных служб решает проблемы:

* передачи запросов через брандмауэр;
* определения конечных точек (endpoints) сервисов.

На сегодняшний день, наиболее популярным способом решения вышеобозначенных проблем являются веб - службы, базирующиеся на SOAP - протоколах. Клиентские приложения используют WSDL для генерации прокси - классов, для определения конечных точек и получения доступа к сервисам через firewall.

Вторым способом решения данных проблем является Windows Communication Foundation (WCF), использующий основанные веб-протоколы передачи данных, в т.ч. SOAP.

Предприятия, как правило, используют два подхода для решения данных проблем. Первый заключается в том, чтобы выборочно позволять приложениям открывать входящие порты на локальных и сетевых брандмауэрах. Второй - заключается в использовании промежуточных служб, находящихся между брандмауэрами и клиентскими приложениями, и являющихся, по сути, "мостом", обеспечивающим обмен сообщениями.

Первый подход реализуем только в сравнительно небольших корпоративных сетях, ограничением является эффективное обеспечение безопасности. Проблема второго заключается в трудности реализации, организация маршрутизации между тысячами и миллионами соединений потребует значительных издержек.

Рассмотрим более подробно .Net Service Bus, чтобы понять, как данный сервис справляется с обеспечением эффективного обмена сообщениями.

**Концепция .Net Service Bus**

Первоначально для того, чтобы воспользоваться возможностями .Net Service Bus, приложения должны зарегистрироваться в соответствующем реестре. Когда приложение обращается к службе, находящейся за брандмауэром, с помощью ,Net Service Bus определяется конечная точка службы и с ней устанавливается связь., т.е. непосредственно сам сервис выполняется за брандмауэром, а соединение с ним обеспечивает Service Bus. Клиенты видят только IP адрес, предоставляемый Service Bus, а не IP, предоставляемый компанией. Данный подход использует возможности .Net Access Control для обеспечения безопасности доступа.

Фактически, приложение использующее .Net Service Bus, реализует WCF функционал, но это не является обязательным. Приложение, обращающееся к сервисам за брандмауэром также может сформировать SOAP или REST - запрос.

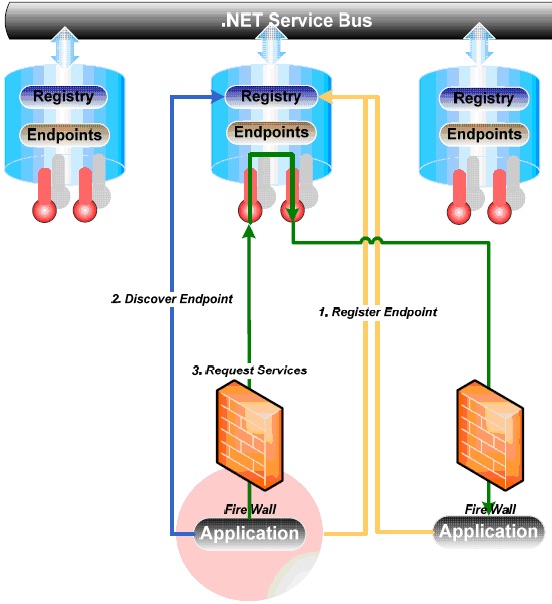


Рисунок 8.1 - (Источник -книга " Introducing Windows Azure" Henry Li)

**Шаблон Enterprise Service Bus (ESB)**

Enterprise Service Bus, или сервисная шина предприятия — подход к построению распределённых корпоративных информационных систем. Обычно включает в себя промежуточное ПО, которое обеспечивает взаимосвязь между различными приложениями по различным протоколам взаимодействия.

Шаблон ESB требует:

* интегрированных механизмов идентификации и управления доступом
* механизма пРисунокваивания имен сервисам
* общей среды обмена сообщениями

Шаблон ESB помогает преодолеть различия между сервисами с точки зрения управления идентификацией, соглашений о пРисунокваивании имен, форматов сообщений и протоколов связи. Как только сервис попадает в шину, все остальные сущности, входящие в нее, могут связываться с ним, даже несмотря на то, что в обычных условиях взаимодействие между ними напрямую было бы невозможным.

К продуктам и технологиям реализации ESB можно отнести:

* Active Directory
* UDDI
* BizTalk Server
* MSMQ
* WCF

**Сервисная шина Интернета (Internet Service Bus – ISB)**

Концепция сервисной шиной Интернета - разрабатываемый подход, при котором возможности шаблона ESB используются в рамках Интернета. Основное отличие от обычного ESB подхода заключается в том, что компоненты ESB должны быть спроектированы и реализованы для работы в вычислительном "облаке".

ISB сделала бы возможным интегрировать вашу локальную ESB с вашими сервисами, выполняющимися в "облаке", с различными сервисами сторонних производителей, RIA и веб - приложениями.

Проблемы реализации двусторонней Интернет связи:

* нехватка IPv4-адресов;
* безопасность - как правило, локальное программное обеспечение практически полностью отгорожено от внешнего мира множеством уровней межсетевых экранов и другими защитными сетевыми устройствами.

Тем не менее на сегодняшний день можно говорить о наличии двунаправленных приложений, таких как:

сервисы мгновенного обмена сообщениями (ICQ, MSN)

сетевые игры

приложения совместного использования файлов (torrent - клиенты)

**Обмен сообщениями**

Центральной частью .Net Services Bus является сервис ретрансляции, обеспечивающий возможность обмена сообщениями.

Сервис ретрансляции поддерживает:

* однонаправленный обмен сообщениями;
* синхронный обмен сообщениями;
* обмен сообщениями между равноправными участниками сети.

В этом случае вашим сервисам больше нет необходимости создавать локальных слушателей транспортного уровня; они полностью полагаются на сервис ретрансляции в вопросах обработки указанных транспортных аспектов связи. Сервис ретрансляции просто пересылает входящие сообщения на ваш локальный сервис.

**Список источников для самостоятельного изучения**

**.Net Service Bus**

1. <http://msdn.microsoft.com/ru-ru/library/ee872418.aspx>
2. <http://www.microsoft.com/windowsazure/AppFabric/Overview/default.aspx#top>
3. <http://itechthoughts.wordpress.com/2009/04/12/windows-azure-service-bus-publishsubscribe-example/>

**Работа с .Net Service Bus**

1. <http://msdn.microsoft.com/en-us/magazine/dd569756.aspx>

**Сервисная шина Интернета**

1. <http://msdn.microsoft.com/en-us/library/bb906065.aspx>

**Enterprise Service Bus**

1. <http://www.ibm.com/developerworks/ru/library/ws-whyesb/>