Azure Photo Mosaics

Version: 1.1
Date published: 1/7/2012
Author: *Jim O'Neil (jim.oneil@microsoft.com)*

Azure Photo Mosaics

This sample application includes a Windows Azure cloud application and a Windows Forms client application to convert images uploaded from your local machine to photographic mosaics.

It exercises many of the core concepts of the Windows Azure Platform including:

- Web Roles
- Worker Roles
- Windows Azure Storage (queues, blobs, and tables)
- Windows Azure Service Bus
- Windows Azure Caching
- Windows Azure Diagnostics

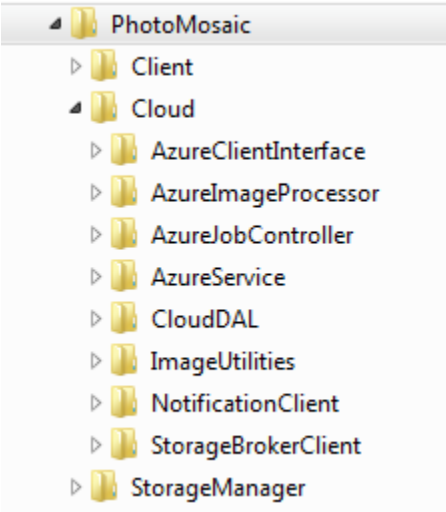
This document provides a high level overview of the application structure and requirements to run the sample either locally or in Windows Azure. A complementary blog series at http://blogs.msdn.com/b/jimoneil/p/series_mosaics.aspx explores the implementation in much greater detail.

Application Files

The application can be downloaded from github at git://github.com/jimoneil/Azure-Photo-Mosaics.git (or you can download the files in a .zip archive)

The bits may be updated from time to time, but this document will enumerate the changes made.

There are three Visual Studio 2010 solutions included in the source; each requires the .NET 4 Framework.



Client – a Visual Basic Windows Forms client application which prompts for a local image to convert and uploads the image to Windows Azure to perform the conversion.

Cloud – a Windows Azure cloud application (two Worker Roles and one Web Role) and some ancillary assemblies, coded in a combination of Visual Basic and C#.

Storage Manager – a C# Windows Forms application to create (and remove) the required Windows Azure Storage constructs. Use this application prior to running the cloud application locally or in Windows Azure.

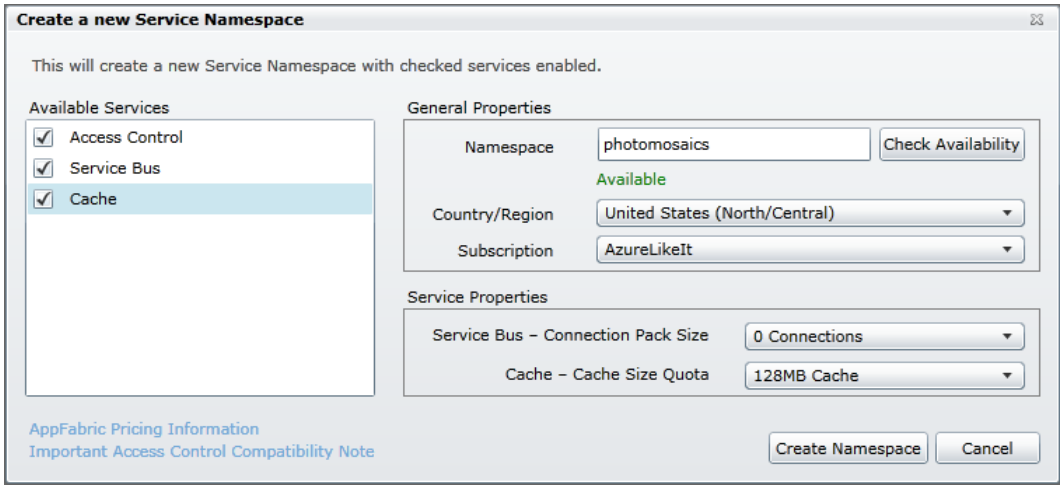
Windows Azure Service Requirements

To fully exercise the application, you'll need the following services and capabilities in Windows Azure:

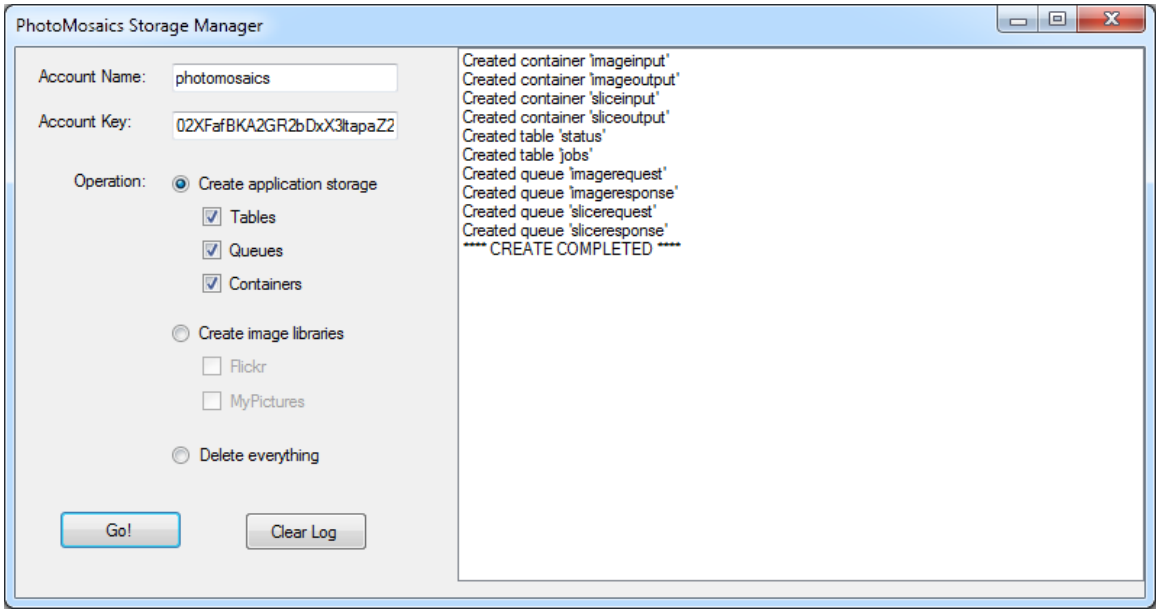
- **Windows Azure Storage Account** – one account is sufficient, but the application supports multi-tenancy so you can experiment with multiple accounts. For latency and to avoid bandwidth charges, you should make sure your Storage Account, Hosted Service, and ServiceBus and Caching namespace (covered next) are collocated in the same sub-region (e.g., North Central US).

- **Windows Azure Hosted Service** – the cloud application requires a minimum of three role instances. Small or extra-small CPU size is sufficient; some accounts provide a complimentary allotment of compute services . Consult http://www.microsoft.com/windowsazure/offers/ for details on the level of services available for your account.

- **Windows Azure Service Bus and Caching** – the application does include Service Bus and Caching capabilities, although neither is absolutely necessary. A 128 MB cache is sufficient.



Storage Manager Application Overview



Storage Manager is a simple utility program to create the queues, tables, and blob containers (along with metadata) required by the Photo Mosaics cloud application.

As distributed, the application will not compile due the intentional omission of the FlickrNET.dll, which is distributed under an open source license and available at http://flickrnet.codeplex.com/. After you download the binary release, add a Reference to FlickrNet.dll, and the application will build.

You will require your own Flickr API Key, which you can obtain via http://www.flickr.com/services/api/, and add to the *app.config* file described below.

Flickr integration is not required; however, it's useful for populating library of image tiles to generate the mosaic images.

app.config

In the *app.config* file, modify the three user settings shown below, providing the credentials for your Windows Azure Storage account and optionally a Flickr API key.

```
<userSettings>
  <StorageManager.Properties.Settings>
    <setting name="AccountName" serializeAs="String">
      <value>[Azure Storage Account]</value>
    </setting>
    <setting name="AccountKey" serializeAs="String">
      <value>[Storage Account Key]</value>
    </setting>
    <setting name="FlickrAPIKey" serializeAs="String">
      <value>[Optional Flickr API Key]</value>
```

```
    </setting>
  </StorageManager.Properties.Settings>
</userSettings>
```

Create Application Storage option

Selecting this option and pressing Go! will create the following elements in the Windows Azure Storage account. The status of each operation is shown in the ListBox in the right half of the application window.

Table	Queue	Blob Container
status	imagerequest	imageinput
jobs	imageresponse	imageoutput
	slicerequest	sliceinput
	sliceresponse	sliceoutput

Create Image Libraries option

Selecting this option will create a blob container (for each sub option selected) and populate it with images from the selected source: Flickr or your My Pictures directory.

Two constant values defined in *StorageManager.cs* govern the number and size of images that will be included, and can be modified in code if desired.

```
const Int32 IMAGES_MAXPERLIBRARY = 300;     // max number of images
const Int32 IMAGES_MAXSIZE = 1024 * 250;    // max image size (bytes)
```

The Flickr option accesses the Interestingness feed and selects the square thumbnail size option (70 x 70 pixels). You will need to provide your own Flickr API key (in the *app.config*), and download the FlickNet.dll binary from http://flickrnet.codeplex.com/ to use this feature.

The MyPictures option accesses the well-known directory on your local machine `Environment.SpecialFolder.MyPictures` and its subdirectories, selects images with a size no greater than *IMAGES_MAXSIZE* and uploads them to the blob container MyPictures.

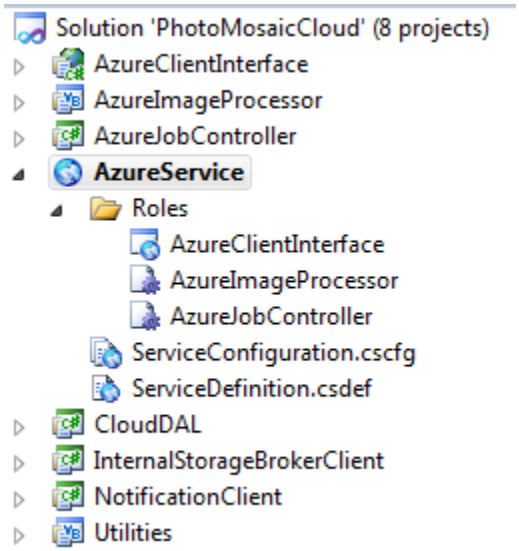
You can create your own blob containers as well and upload images from any source for use by the Photo Mosaics program; however, such containers must:

1. Allow public access to the blobs within the containers, and
2. Define a metadata element named *ImageLibraryDescription*, with an arbitrary value presumably identifying the source of the images in that library.

Delete Everything option

This option permanently and irrevocably deletes all of the Windows Azure storage elements shown in the previous table from the given Windows Azure Storage Account. Use with caution!

Cloud Application Overview



The Windows Azure cloud application solution contains eight projects:

1. **AzureClientInterface** – an Web Role housing three WCF services (C#)
2. **AzureImageProcessor** – a Worker Role that does the mosaic image creation (VB)
3. **AzureJobController** – a Worker Role that coordinates the image creation tasks (C#)
4. **AzureService** – the Windows Azure service configuration
5. **CloudDAL** – a data access library providing some abstraction over Windows Azure Storage (C#)
6. **InternalStorageBrokerClient** – WCF client assembly used to communicate with a WCF service hosted on an internal port by *AzureClientInterface* (C#)
7. **NotificationClient** – Windows Azure AppFabric Service Bus client assembly used to provide updates to the Windows Forms client application (C#)
8. **Utilities** – miscellaneous image processing utility methods (VB)

By default, the three roles specify a small VM size, but Extra-Small should work fine and may be a better fit depending on the type of account you are using. The roles are configured with a single instance, which does not guarantee an SLA, but it enough for learning purposes. The primary mechanism for modifying application scale is by increasing the number of instances of the *AzureImageProcessor*.

Configuration Requirements

Most of the configuration parameters are owned by the *AzureClientInterface* Web Role and exposed to the other roles via an internal WCF service. The notable exceptions are the *ApplicationStorageConnectionString* and the Windows Azure diagnostics connection string, which are required to be defined for each role.

AzureClientInterface

- **ApplicationStorageConnectionString:** the full connection string for the Windows Azure Storage Account under which the application will be run. The four queues used by the application must exist in this account.

- **AppFabricNamespace:** the namespace for Service Bus and Caching
- **ServiceBusSecret:** The Default Key associated with your Service Bus namespace
- **ServiceBusIssue:** The Default Issuer associated with your Service Bus namespace (typically the string *owner*)
- **UserStorageAccountName:** account name that owns the Windows Azure tables and blob containers used by the application. This is an entry point to demonstrating multi-tenancy options, but is not required; you can reuse the *ApplicationStorageConnectionString* account name here.
- **UserStorageAccountKey:** account key associated with *UserStorageAccountName*
- **Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString:** full connection string for the Windows Azure Diagnostics tables and blob containers; this should be set to the same value as *ApplicationStorageConnectionString.*

AzureImageProcessor

- **ApplicationStorageConnectionString:** as above
- **CachingMechanism:** a integral value from 0 to 3 indicating the type of caching mechanism to use for access to the ImageLibrary tiles:
  - **0:** no caching, directly access the ImageLibrary container and resize the images to the requisite tile size on each request
  - **1:** access is via a secondary blob container containing all of the ImageLibrary images, but resize for immediate use
  - **2:** use Windows Azure Caching
  - **3:** cache the images in memory in each role instance

  If you enable the caching capability, you must also include the host and authentication token for your AppFabric Caching namespace in the *app.config* file for this role (excerpt below). You can obtain these items from the Windows Azure portal.

```
<dataCacheClients>
  <dataCacheClient name="default">
    <hosts>
      <host name="<namespace>.cache.windows.net" cachePort="22233" />
    </hosts>
    <securityProperties mode="Message">
      <messageSecurity authorizationInfo="<authentication token>”>
      </messageSecurity>
    </securityProperties>

   <localCache isEnabled="true" ttlValue="21600" />

    <tracing sinkType="DiagnosticSink" traceLevel="Verbose"/>
  </dataCacheClient>
</dataCacheClients>
```

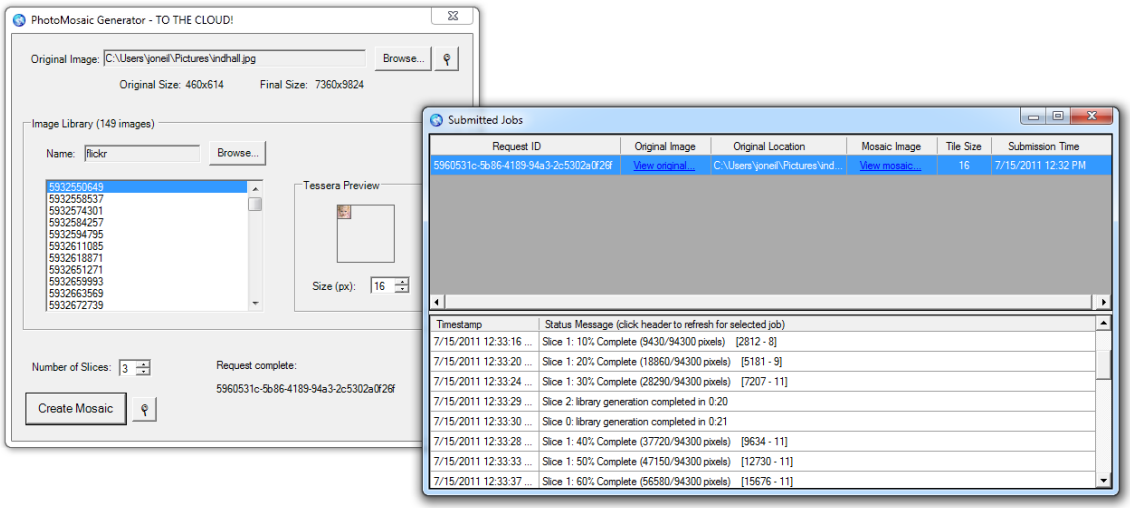
- **Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString:** as above

AzureJobController

- **ApplicationStorageConnectionString:** as above
- **Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString:** as above

You should be able to use Windows Azure development storage when you run the application within the Windows Azure emulator (previously known as the Development Fabric); however, most of the development occurred directly against storage accounts in the cloud. Utilization of Caching while running the application in the Windows Azure emulator is definitely not recommended, since there is no local counterpart for that feature as part of the emulator.

Client Application Overview



The client is a Visual Basic Windows Forms application that access the cloud application through Web Services implemented in the *AzureClientInterface* Web Role (namely, the *AzureJobBroker* and *AzureStorageBroker* services).

The only configuration requirement is to modify the *app.config* file to include the endpoint addresses of the two Web Services:

```
<client>
  <endpoint address="http://127.0.0.1:81/StorageBroker.svc"
      binding="basicHttpBinding"
      contract="AzureStorageBroker.IStorageBroker"
    />
  <endpoint address="http://127.0.0.1:81/JobBroker.svc"
      binding="basicHttpBinding"
      contract="AzureJobBroker.IJobBroker"
      bindingConfiguration="JobBroker_BindingConfig"
    />
</client>
```

The above configuration targets the application running locally under the Windows Azure Cloud Emulator (Development Fabric). If you deploy the Cloud application to a Hosted Service in Windows Azure, replace the host with the appropriate endpoint, e.g., http://mymosaicapp.cloudapp.net/.

Note there is currently no access control mechanism implemented for the cloud application, so anyone with the client application and your endpoint address can access the service!

> You may want to consider deploying to a staging account to limit exposure and unauthorized usage. Incorporation of the Windows Azure Access Control Service in this sample is planned.

HELP!

Send a note to jim.oneil@microsoft.com if you are having trouble getting this sample to run or have questions about its implementation. Be sure to check out the blog series at http://blogs.msdn.com/b/jimoneil/p/series_mosaics.aspx which dives deeper into the implementation.

Change Log

July 15, 2011	Initial release
January 7, 2011	Transition to github repository