Lab of Things: Using the Data Storage API

The data storage API enables HomeOS apps to save data (temperature readings, images from a camera, etc.) to a Windows Azure storage account. You can use the IStream and StreamFactory classes to write and retrieve data. In this section we'll cover how to set up your apps to exchange data with Azure storage.

See Also: Datastream API Reference.

Contents

Prerequisites

- Windows Azure account
- Azure Storage Account
- Azure Storage Explorer (optional, but very handy)

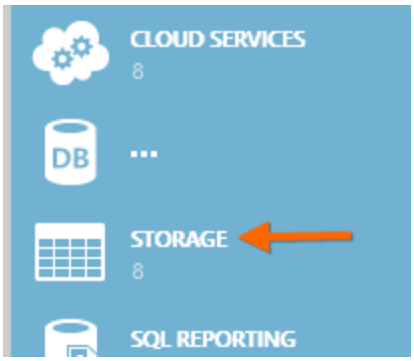
Configuring an App

Before working with data, you will need to tell your app which Azure account to use.

Use the information for "testdrive" if you are just testing (skip straight to step 6):

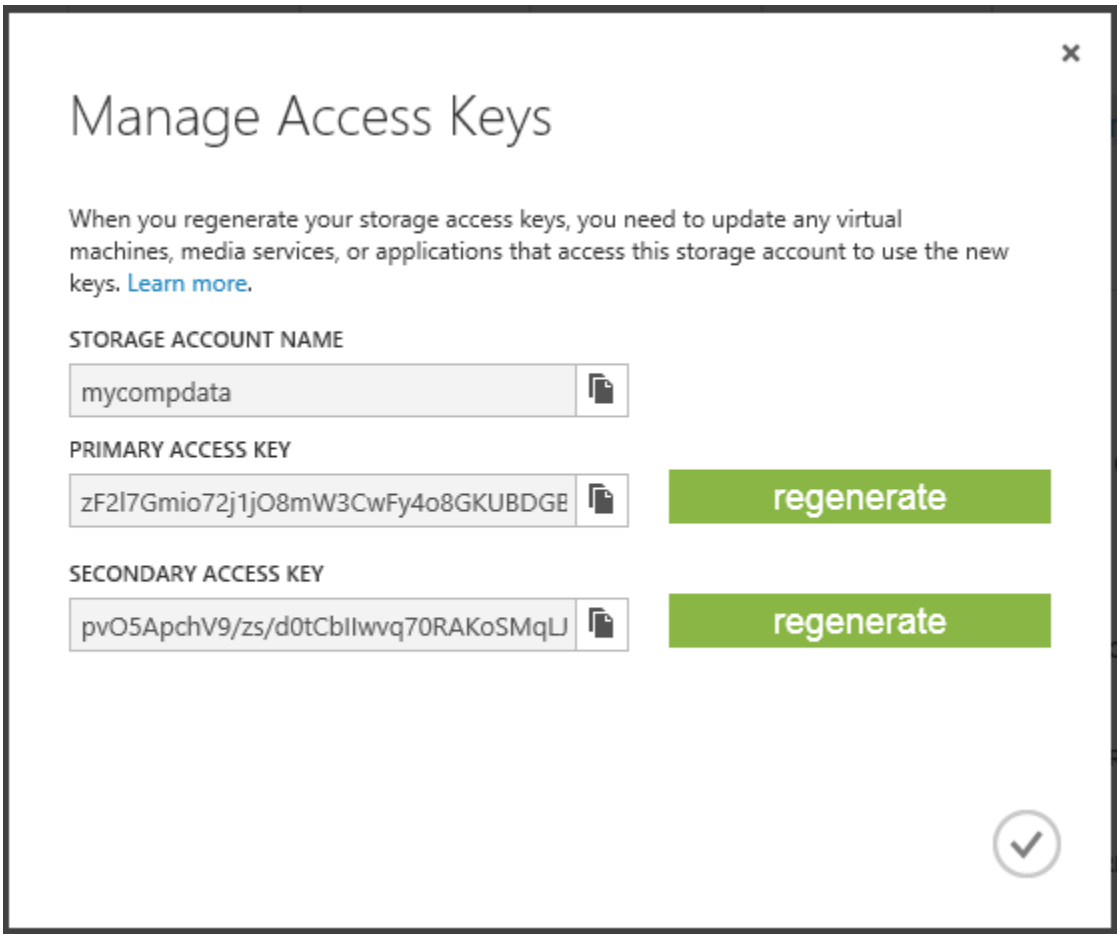
- Storage account name: `testdrive`
- Storage account key:
  `zRTT++dVryOWXJyAM7NM0TuQcu0Y23BgCQfkt7xh2f/Mm+r6c8/XtPTY0xxaF6tPSACJiuACsjotDeNIVyXM8Q==`

1. Login to your Windows Azure account.
2. In the left pane, click **Storage**.



3. Select the storage account you want to use.
4. At the bottom of the screen, click **Manage Access Keys**.

5. Copy the **Storage Account Name** and **Primary Access Key**.



6. Paste the Account Name and the Account Key of the blob account in the Settings.xml in the configuration folder you want to run (e.g. DemoConfig for testing, Config for deployment)

The entries in Settings.xml should look something like this:

```
<Param Name="DataStoreAccountName" Value="MyAccountName" />
<Param Name="DataStoreAccountKey" Value="YourKeyString" />
```

About Data Streams

All data is stored as an ordered list of elements (tuple) where each key can have one or more values. Each value also has a UTC timestamp, so that each collection of values is effectively a time series. This is useful for logging information that changes over time such as electrical current usage.

A key can be any data type that adheres to IKey. StrKey is the sole implemented type, although other types of keys can include integer and bucket (for representing ranges of values). Once a stream has been defined its data type cannot be changed. For example, once you define a string key that stores a set of integer values, you cannot pass a floating point value into that key.

There are two types of streams:

- DataFileStream: Used for small values such as temperature. Similar to entries in a log file.
- DataDirectoryStream: Used for large values such as pictures and videos. Similar to a directory with data entities.

Writing Data

There are two functions for writing data to a stream:

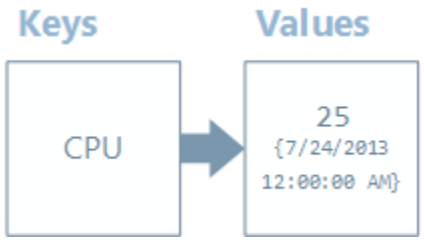
- Append: Adds a new value to the set of values for the specified key.
- Update: Modifies the newest entry in the set of values associated with the specified key.

Example 1: Append vs. Update

To demonstrate using Append and Update, let's look at a hypothetical app that monitors CPU usage and posts updates every five minutes. When the app first creates a data stream, it calls Append to add an initial value:

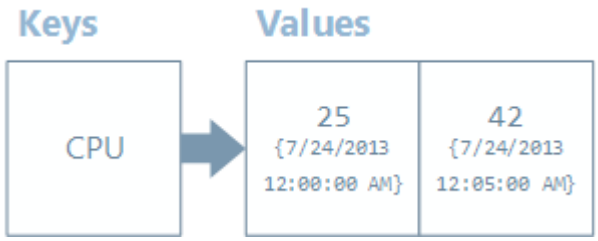
```
StrKey key = new StrKey("CPU");
StrValue val = new StrValue("25");
datastream.Append(key, val);
```

Now the key CPU has one value "25", as well as a UTC timestamp indicating the moment when the operation took place. The following diagram shows the CPU key after initial creation:



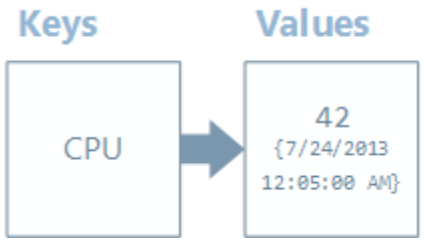
Five minutes later the app is ready to write another update. The following diagram shows the CPU key after Append is called, adding a second value of "42" (timestamp value is automatically adjusted):

```
StrKey key = new StrKey("CPU");
StrValue val = new StrValue("42");
datastream.Append(key, val);
```



If on this second call the app were to call Update instead of Append, the initial value would be overwritten and the CPU key would appear as shown here:

```
StrKey key = new StrKey("CPU");
StrValue val = new StrValue("42");
datastream.Update(key, val);
```



Example 2: Writing Data

To write data, your app must first create a stream by calling CreateFileStream. The following snippets from the Dummy example app (Hub\Apps\Dummy\Dummy.cs) show the calls in order:

1. Declare an IStream instance.

   ```
   IStream datastream;
   ```

2. Call ModuleBase.CreateFileStream, passing the stream ID and a boolean indicating whether the stream is remotable.

   ```
   datastream = base.CreateFileStream<StrKey, StrValue>("dumb", false);
   ```

3. Work with the stream. In this example, the Dummy app pings a set of ports and appends dummy key/value data.

```
public void Work()
{
    int counter = 0;
    while (true)
    {
        counter++;

        lock (this)
        {
            foreach (VPort port in accessibleDummyPorts)
            {
                SendEchoRequest(port, counter);
            }
        }

        WriteToStream();
        System.Threading.Thread.Sleep(1 * 10 * 1000);
    }
}

public void WriteToStream()
{
    StrKey key = new StrKey("DummyKey");
    datastream.Append(key, new StrValue("DummyVal"));
    logger.Log("Writing {0} to stream ", datastream.Get(key).ToString());
}
```

Reading Data

There are several functions you can use to retrieve data from a stream:

- Get: Returns the newest value for the specified key.
- GetLatest: Returns the latest tuple for the specified key.
- GetAll: Returns all tuples within the specified time range.
- GetKeys: Returns all keys in the specified key range.

Example 3: Reading Data

The following code snippet demonstrates calling `Get()`.

1. Declare an IStream instance.

   ```
   IStream datastream;
   ```

2. Call CreateFileStream, passing the stream ID and a boolean indicating whether the stream is remotable.

   ```
   datastream = base.CreateFileStream<StrKey, StrValue>("streamID", false);
   ```

3. Read the data.

   ```
   string theData = datastream.Get(key).ToString();
   ```

[]