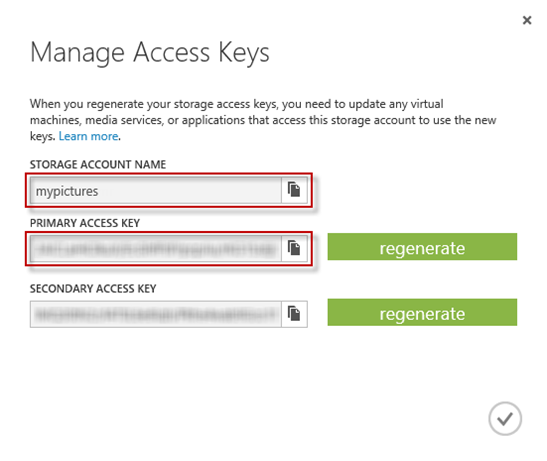
# How to reduce the times of requests sent to Azure Storage Service in Windows Store apps

## Introduction

When you develop an app that needs connecting to Azure storage service, it will take a long time to send requests and get responses in the client application.

This sample demonstrates how reduce the times of connection between client app and Azure storage service. It also shows how to handle general exception when you get the response.

## Building the Sample

1. Create a new **Storage Account** from the Windows Azure Management Portal.  
     
   To do this, follow the instructions in [How To Create a Storage Account](http://www.windowsazure.com/en-us/manage/services/storage/how-to-create-a-storage-account/).  
     
   Get the **Storage Account Keys**. Browse to your storage account dashboard and click **Manage Access Keys** on the bottom bar.  
     
     
     
   Copy the **Storage Account Name** and **Primary Access Key** values.  
     
   
2. Open this project, go to the **Solution->Project->DataSource->TableDataSource.cs** file, replace account name and account key you copied in the step 1.
3. Build the project, download the class library from nugget.
4. Press **F5** to start the app.

## Running the Sample

1. This app only has one page view: main page view.



2. You can Click the **Delete** button first, then use the **Regenerate the Sample data** button to create the sample data.

## Using the Code

The code sample provides the following reusable functions to handle Azure storage transient fault error.

### How to handle conflict error?

|  |
| --- |
| -Code block start-  --C# code snippet start--  public class ConflictRetryPolicy:IRetryPolicy  {  int maxRetryAttemps = 10;  TimeSpan defaultRetryInterval = TimeSpan.FromSeconds(5);    public ConflictRetryPolicy(TimeSpan deltaBackoff, int retryAttempts)  {  maxRetryAttemps = retryAttempts;  defaultRetryInterval = deltaBackoff;  }    public IRetryPolicy CreateInstance()  {  return new ConflictRetryPolicy(TimeSpan.FromSeconds(5), 10);  }    public bool ShouldRetry(int currentRetryCount, int statusCode, Exception lastException, out TimeSpan retryInterval, OperationContext operationContext)  {  retryInterval = defaultRetryInterval;  if (currentRetryCount >= maxRetryAttemps)  {  return false;  }  if (statusCode == 409)  {  return true;  }  else  {  return false;  }  }  }  --C# code snippet end--  --VB code snippet start--  Public Class ConflictRetryPolicy  Implements IRetryPolicy  Private maxRetryAttemps As Integer = 10  Private defaultRetryInterval As TimeSpan = TimeSpan.FromSeconds(5)  Public Sub New(deltaBackoff As TimeSpan, retryAttempts As Integer)  maxRetryAttemps = retryAttempts  defaultRetryInterval = deltaBackoff  End Sub  Public Function CreateInstance() As IRetryPolicy Implements IRetryPolicy.CreateInstance  Return New ConflictRetryPolicy(TimeSpan.FromSeconds(5), 10)  End Function  Public Function ShouldRetry(currentRetryCount As Integer, statusCode As Integer, lastException As Exception, ByRef retryInterval As TimeSpan, operationContext As Microsoft.WindowsAzure.Storage.OperationContext) As Boolean Implements IRetryPolicy.ShouldRetry  retryInterval = defaultRetryInterval  If currentRetryCount >= maxRetryAttemps Then  Return False  End If  If statusCode = 409 Then  Return True  Else  Return False  End If  End Function  End Class  --VB code snippet end--    -Code block end- |

### How to reduce request sending times?

|  |
| --- |
| -Code block start-  --C# code snippet start--  public static async Task<bool> DeleteEntity(DynamicTableEntity entity)  {  try  {  var table = client.GetTableReference(tableName);    TableOperation deleteOperation = TableOperation.Delete(entity);  var result = await table.ExecuteAsync(deleteOperation);    return true;  }  catch (Exception e)  {  //In windows store apps, StorageException is an internal class.  //You can't convert Exception to StorageException, so you should use  //RequestResult.TranslateFromExceptionMessage(e.Message) to get the HttpStatusCode.  var result = RequestResult.TranslateFromExceptionMessage(e.Message);    //We treat 404 as it has already been deleted.  if (result.HttpStatusCode == 404)  {  return true;  }  else  {  return false;  //throw new WebException(result.HttpStatusMessage);  }  }    }  --C# code snippet end--  --VB code snippet start--  Public Shared Async Function DeleteEntity(entity As DynamicTableEntity) As Task(Of Boolean)  Try  Dim table = client.GetTableReference(tableName)  Dim deleteOperation As TableOperation = TableOperation.Delete(entity)  Dim result = Await table.ExecuteAsync(deleteOperation)  Return True  Catch e As Exception  ' In windows store app, StorageException is an internal class.  ' You can't convert Exception to StorageException, so you should use  ' RequestResult.TranslateFromExceptionMessage(e.Message) get the HttpStatusCode.  Dim result = RequestResult.TranslateFromExceptionMessage(e.Message)  ' We treat 404 as it has already been deleted.  If result.HttpStatusCode = 404 Then  Return True  Else  ' Throw new WebException(result.HttpStatusMessage);  Return False  End If  End Try  End Function  --VB code snippet end--    -Code block end- |

[Azure table storage feature guide](http://www.windowsazure.com/en-us/documentation/articles/storage-dotnet-how-to-use-table-storage-20/) shows that it is not necessary to retrieve an entity before deletion if you have already retrieved it before. So we can delete that retrieve operation in code.

If you send the same entity delete operation twice, you should catch the 404 error and ignore it as the code above does.

## More Information

<http://www.windowsazure.com/en-us/documentation/articles/storage-dotnet-how-to-use-table-storage-20/#delete-entity>