Windows Azure: interagire con il Cloud Storage da Visual Basic 2010

#### di [Alessandro Del Sole](https://mvp.support.microsoft.com/profile/Alessandro.Del%20Sole) - Microsoft MVP

* 1. 

*Maggio, 2012*

In questo articolo viene discussa la possibilità di interagire, da codice Visual Basic 2010, con lo spazio di memorizzazione on-line (Cloud Storage) messo a disposizione dalla piattaforma di Windows Azure. Non voglio dilungarmi su cosa sia Windows Azure e su come funzioni lo storage (se non qualche cenno necessario), visto che ci sono trattazioni di gran lunga più dettagliate delle mie. Se volete un'infarinatura su Azure e i suoi servizi, [potete scaricare le slide](http://www.visual-basic.it/scarica.asp?ID=1064) di Renato Marzaro presentate in occasione del nostro evento a Roma lo scorso novembre.

#### Cosa do per scontato

Chiaramente il fatto che abbiate un account su Windows Azure e che abbiate creato il vostro Cloud Storage. Se vi servono informazioni in merito, potete dare un'occhiata alla MSDN Library.

#### Scopo del gioco

Lo scopo di questo articolo è mostrare come sia possibile memorizzare e leggere i propri file nella parte del Cloud Storage chiamata **Blob Storage**. Lo faremo utilizzando le nuove API che troviamo a disposizione nel [Windows Azure SDK](http://www.windowsazure.com/it-it/develop/net/), che quindi è necessario scaricare ed installare. Inoltre, daremo un approccio molto "client" alla cosa. Normalmente esempi in merito si trovano (oltre che solo in C#) per dimostrare l'accesso allo storage da applicazioni Silverlight. Ma siccome non esiste solo il Web, vedremo un qualcosa di diverso.

#### Da dove partiamo e dove vogliamo arrivare

Tempo fa ho pubblicato un progetto su CodePlex, chiamato Azure Blob Studio 2011, che consiste in un client WPF ma anche in un'estensione per Visual Studio 2010 che consentono di interagire con il proprio storage per memorizzare o scaricare file, organizzati in cartelle. All'interno di quel progetto, per il quale è disponibile tutto il codice VB, ho implementato una classe che rende fruibili, in modo un pochino diverso e forse migliore (a mio avviso), la maggior parte degli oggetti esposti dalle API per lo storage di Windows Azure. L'obiettivo che quindi ci prefissiamo è ricostruire quella classe chiamata BlobService per capire come usare codice managed verso lo storage. La serie sarà costituita da tre, massimo quattro post. Se non avete tempo di aspettare o semplicemente siete curiosi o volete una linea guida, è conveniente scaricare il codice. Impareremo quindi a creare cartelle, rimuoverle, elencarle, a caricare, scaricare ed eliminare file.

#### Overview del Cloud Storage

Quando attiviamo un abbonamento a Windows Azure (incluso quello MSDN con i relativi benefit) ci viene messo a disposizione uno spazio di memorizzazione dati chiamato Cloud Storage. Tale spazio è, in sostanza, un insieme di quattro servizi:

* + **Blob**: in questo spazio possiamo archiviare dati binari, tipicamente immagini, video, documenti; file, in altre parole
  + **Table**: qui possiamo memorizzare dati in forma tabulare, secondo la ben nota logica colonne/campi. Non ci sono le relazioni in questo tipo di storage, ma la scalabilità è molto alta
  + **Queue**: questo servizio permette lo scambio di messaggi tra ruoli (Web e Worker) che compongono un'applicazione Web per Windows Azure
  + **Drive**: come il nome lascia intendere, si tratta di hard disk virtuali formattati in NTFS che permettono caricamento/lettura di informazioni da parte delle applicazioni Web che girano su Azure

Maggiori dettagli potete trovarli in questo post di Mario Fontana, Evangelist di Microsoft Italia. A noi interessa il Blob Storage per diversi motivi, il primo dei quali è che è lo spazio che ha maggiormente senso anche in applicazioni client. Perchè? L'esempio che consideriamo è semplice: ho i miei documenti Word, le mie fatture in PDF, altri file che voglio mettere al sicuro nel mio spazio sul cloud e voglio farmi un banale client in WPF, Windows Forms, Console addirittura, per gestirli senza dover per forza lavorare lato Web (anche perché magari non ho modo di deployare seriamente un'applicazione Silverlight).

#### Programmabilità

Il Cloud Storage può essere raggiunto dagli sviluppatori con due tipiche modalità:

* + approccio REST, quindi scambio di informazioni tramite Http
  + API managed, usando librerie messe a disposizione da Windows Azure SDK

Noi utilizzeremo la seconda modalità in Visual Studio 2010. Per cui scarichiamo e installiamo l'SDK. Una volta installato, localizziamo la libreria chiamata Microsoft.WindowsAzure.StorageClient.dll all'interno della cartella C:\Programmi\Windows Azure SDK\v1.x\Ref (dove ‘x’ corrisponde al numero *minor* di versione dell’SDK). Ci tornerà utile tra pochissimo.

### Iniziamo: CloudStorageAccount & CloudBlobClient

Apriamo Visual Studio 2010 e creiamo un nuovo progetto di tipo Class Library per Visual Basic 2010. Rinominiamo la classe principale in BlobService. Quando il progetto è creato, aggiungiamo un riferimento all'assembly Microsoft.WindowsAzure.StorageClient.dll sopra citato. Ora, di primo acchitto due cose ci interessano:

* + ottenere le credenziali di accesso al proprio spazio di storage
  + ottenere un riferimento managed allo storage stesso

La classe **CloudStorageAccount**, del namespace **Microsoft.WindowsAzure**, è il nostro primo contatto con lo storage e ci permette di definire le credenziali di accesso. Come sapete, le credenziali sono costituite dalla coppia Account Name e Shared Key. Il primo lo avete stabilito quando avete creato lo storage, il secondo è generato da Azure e potete recuperarlo nel portale di sviluppo, nella sezione dedicata allo storage. Il riferimento allo storage, invece, è rappresentato da una classe chiamata **CloudBlobClient**.

A livello di classe, quindi, iniziamo con l'aggiungere due dichiarazioni di questo tipo:

* 1. Visual Basic
  2. Public Class BlobService
  3. Private AccountInfo As CloudStorageAccount
  4. Private blobStorage As CloudBlobClient

La classe CloudStorageAccount fornisce diversi metodi per gestire le credenziali. In primo luogo, la classe ci permette di stabilire che vogliamo accedere allo storage locale di sviluppo, richiamandone la proprietà DevelopmentStorageAccount. Se invece vogliamo accedere allo storage online, possiamo generalmente utilizzare il metodo FromConfigurationSettings, nel caso in cui le credenziali risiedano nel file di configurazione dell'applicazione, oppure il metodo Parse (o TryParse) che invece analizza una stringa contenente le credenziali e la trasforma in un'istanza di CloudStorageAccount.

Poiché la nostra classe costituirà un modo semplificato ma univoco di accedere allo storage, possiamo fare questa distinzione offrendo due overload del costruttore, assumendo che, per ragioni di privacy, le credenziali di accesso allo storage online non siano memorizzate nel file di configurazione, bensì siano hard-coded:

* 1. Visual Basic
  2. Public Sub New(ByVal accountName As String, ByVal sharedKey As String)
  3. If String.IsNullOrEmpty(accountName) Or String.IsNullOrEmpty(sharedKey)

Then

* 1. Throw New ArgumentNullException()
  2. End If
  3. Me.AccountInfo = CloudStorageAccount.Parse(
  4. "DefaultEndpointsProtocol=http;AccountName="
  5. + accountName + ";AccountKey=" + sharedKey)
  6. blobStorage = AccountInfo.CreateCloudBlobClient()
  7. End Sub
  8. Public Sub New()
  9. 'Use the dev account
  10. Me.AccountInfo = CloudStorageAccount.DevelopmentStorageAccount
  11. blobStorage = AccountInfo.CreateCloudBlobClient()
  12. End Sub

Come vedete, la stringa contenente le credenziali non ha forma libera ma deve rispettare alcuni punti fissi come l'indicazione del protocollo di end point e i due valori AccountName/AccountKey. Qualora invece volessi utilizzare il solo storage di sviluppo locale, non faccio altro che assegnare l'istanza col valore di DevelopmentStorageAccount che è prefissato.

Una volta che abbiamo l'istanza dell'account possiamo invocare il metodo di istanza CreateCloudBlobClient, che restituisce un tipo CloudBlobClient, per ottenere un riferimento managed allo storage, indipendentemente dalla sua locazione (on-line o locale).Questo è fondamentale, perché l'oggetto CloudBlobClient così ottenuto mette a disposizione quanto necessario per accedere a container e blob. Ora la domanda che vi starete facendo è: come faccio a capire se le credenziali immesse sono errate?

La risposta è relativamente semplice: il controllo delle credenziali non è automatico sull'istanza di CloudStorageAccount, quindi in teoria si possono immettere credenziali invalide. Il trucco è invocare un qualunque metodo (come ad esempio uno di quelli che elenca i container disponibili): se l'invocazione restituisce una StorageClientException, le credenziali non sono valide.

Non lo faccio ora perché prima dovrei spiegarvi i metodi per lavorare sui container, ma successivamente implementeremo questo tipo di controllo.

### Lavorare con i Container

Il concetto di Container nel cloud storage è assimilabile al concetto di cartella su disco. Un container può contenere blob e altri container. La classe CloudBlobClient mette a disposizione diversi metodi per lavorare con i container, ma a fattor comune gli step sono:

* 1. tento di ottenere un riferimento al container tramite il metodo GetContainerReference
  2. se non esiste, ho la possibilità di creare il container e di ottenere così il riferimento
  3. se esiste, ottengo il riferimento e basta

Il riferimento al container ci serve per poter gestire successivamente i blob ed è rappresentato dalla classe CloudBlobContainer. Giusto per semplificare l'approccio, cominciamo con l'ottenere l'elenco dei container disponibili sullo storage. Scriviamo codice che invochi il metodo di istanza ListContainers della classe CloubBlobClient:

* 1. Visual Basic
  2. Public Overridable Function ListContainers() As IEnumerable(Of CloudBlobContainer)
  3. Try
  4. Dim result = blobStorage.ListContainers
  5. Return result
  6. Catch ex As Exception
  7. Throw
  8. End Try
  9. End Function

Essenzialmente questo metodo funge da wrapper, ma era necessario in una classe di "servizio" come questa. ListContainers restituisce IEnumerable(Of CloubBlobContainer) e ciascun elemento di questo insieme rappresenta un container, con le sue proprietà che vedremo meglio tra poco. Andando al contrario, vediamo come si elimina una cartella:

* 1. Visual Basic
  2. Public Sub DeleteContainer(ByVal ContainerName As String)
  3. Dim container = blobStorage.GetContainerReference(ContainerName) container.Delete()
  4. End Sub

In pratica si ottiene il suo riferimento e si invoca il metodo di istanza Delete (è bene ovviamente verificare che il riferimento restituito non sia nullo, altrimenti non c'è da cancellare nulla). Passiamo ora a vedere come sia possibile creare un nuovo container. Consideriamo il seguente metodo:

* 1. Visual Basic
  2. Public Overridable Sub CreateContainer(ByVal ContainerName As String, ByVal IsPublic As Boolean)
  3. Try
  4. Dim container = blobStorage.GetContainerReference(ContainerName)
  5. container.CreateIfNotExist()
  6. If IsPublic = True Then
  7. container.SetPermissions(New BlobContainerPermissions With {.PublicAccess = BlobContainerPublicAccessType.Container})
  8. Else
  9. container.SetPermissions(New BlobContainerPermissions With {.PublicAccess = BlobContainerPublicAccessType.Off})
  10. End If
  11. Catch ex As StorageServerException
  12. Throw New StorageServerException
  13. Catch ex As Exception
  14. Throw New Exception(ex.InnerException.Message)
  15. End Try
  16. End Sub

Il codice:

* 1. cerca di ottenere il riferimento al container
  2. se non lo trova, lo crea (CreateIfNotExists)
  3. assegna i permessi di accesso al container

Per permessi di accesso intendiamo la possibilità di rendere pubblico il contenuto del container oppure privato, che significa solo al proprietario dell'account che accede. Per farlo si invoca il metodo SetPermissions della classe CloudBlobContainer che riceve, come argomento, un'istanza della classe BlobContainerPermissions. Questa espone una proprietà chiamata PublicAccess, che va assegnata con un valore dell'enumerazione BlobContainerPublicAccess, come Off (privata) o Container (pubblica). A questo punto il container viene creato nello storage.

Se successivamente volessi ottenere un riferimento a tale container mi basterebbe scrivere:

* 1. Visual Basic
  2. Dim container = blobStorage.GetContainerReference(ContainerName)

Ora dovrebbe essere anche più chiara la modalità con cui è possibile testare la validità delle credenziali di accesso: invoco semplicemente ListContainer all'interno del costruttore, dopo l'ottenimento dell'istanza di CloudStorageAccount, e controllo che non restituisca una StorageClientException.

Riprendiamo il discorso iniziato nel post precedente sulle possibilità di interazione con il Cloud Storage di Windows Azure da codice Visual Basic 2010, nel quale avevamo iniziato a ricostruire parte del percorso che ho personalmente seguito nel creare il mio progetto Azure Blob Studio 2011 disponibile su CodePlex pertanto vi rimando alla precedente lettura se ve la siete persa.

Piccola nota: amandola molto come caratteristica, faccio largo uso della local type inference. Se non vi piace che il tipo non sia esplicito, usate il puntatore del mouse per determinare il tipo restituito dalle espressioni.

In particolare la volta scorsa abbiamo visto come utilizzare metodi per la creazione/lettura/eliminazione di container, assimilabili al concetto di folder.

Questa volta iniziamo a lavorare sui blob, assimilabili al concetto di file. Un blob è quindi un insieme di dati binari che possono essere memorizzati all'interno di container per il successivo utilizzo. Se ad esempio ho un'applicazione che riproduce dei video in streaming, tali file si possono caricare nello Storage ed essere recuperati tramite un url che viene automaticamente assegnato al caricamento nello Storage stesso.

Questa è solo una possibilità. Ad esempio potrebbe interessarmi usare lo Storage come disco virtuale nel quale vado a memorizzare semplicemente un elenco di documenti che vado poi a consumare tramite diversi tipi di client.

Un blob è rappresentato da un'istanza della classe CloudBlob. In linea di massima si ottiene un riferimento al container che contiene il blob e poi si invocano alcuni metodi di istanza della classe CloudBlobContainer. Per esempio ipotizziamo di voler ottenere l'elenco di blob che risiedono in un dato container. Questo si fa tramite il seguente codice:

* 1. Visual Basic
  2. Public Overridable Function ListBlobs(ByVal ContainerName As String) As IEnumerable(Of CloudBlob)
  3. Dim container = blobStorage.GetContainerReference(ContainerName)
  4. Dim temp = container.ListBlobs
  5. Dim lst As New List(Of CloudBlob)
  6. For Each item In temp
  7. lst.Add(CType(item, CloudBlob))
  8. Next
  9. Return lst
  10. End Function

Il metodo ListBlobs sopra esposto riceve come argomento il nome del container e si propone di restituire una IEnumerable(Of CloudBlob). La scelta del tipo IEnumerable è che questo e i suoi derivati facilitano il data-binding in molti scenari client come ad esempio WPF e Silverlight. Prima di dare altre spiegazioni, notiamo che:

•si ottiene un riferimento al container tramite il metodo CloudBlobClient.GetContainerReference

•si ottiene l'elenco di blob nel container tramite il metodo CloudBlobContainer.ListBlobs

Qui nasce un problema, seppur minimale. Questo ListBlobs restituisce una IEnumerable(Of IListBlobItem). IListBlobItem è un'interfaccia che viene implementata da CloudBlob, ma chiaramente non ne completa le caratteristiche. CloudBlob, infatti, ha molte altre proprietà; per tale ragione il metodo di cui sopra si propone di restituire una diversa IEnumerable e per tale ragione all'interno del blocco di codice viene creata una lista di CloudBlob effettivi a partire dall'elenco ottenuto. Questo diventa importante allorquando vogliamo ottenere le proprietà dei blob ottenuti.

Iniziamo ora a vedere come caricare file nel Cloud Storage. La classe CloudBlobContainer offre molti metodi per fare l'upload, come ad esempio UploadBlob o UploadFromStream. Questi, però, lavorano in modo sincrono e quindi c'è il serio rischio di bloccare il famigerato UI thread. Fortunatamente ci sono metodi che lavorano in modalità asincrona e questo consente di mantenere un'applicazione client molto più responsiva.

Consideriamo il seguente metodo:

* 1. Visual Basic
  2. Public Sub UploadBlobs(ByVal ContainerName As String, ByVal BlobList As String())
  3. Try
  4. Dim container = blobStorage.GetContainerReference(ContainerName) For Each blob In BlobList
  5. Dim blobRef = container.GetBlobReference(IO.Path.GetFileName(blob))
  6. blobRef.Properties.ContentType = getMimeType(blob)
  7. blobRef.Properties.ContentLanguage = My.Computer.Info.InstalledUICulture.EnglishName
  8. Dim fs As New FileStream(blob, FileMode.Open)
  9. blobRef.BeginUploadFromStream(fs, AddressOf UploadAsyncCallBack, blobRef)
  10. Next
  11. Catch ex As Exception
  12. Throw
  13. End Try
  14. End Sub

Questo riceve il nome del container destinatario dei blob e un elenco di file (come array) quali argomenti. Dapprima ottiene il riferimento al container quindi, per ciascun nome di blob, va ad ottenere il riferimento al blob stesso nello storage tramite GetBlobReference. Ovviamente a questo punto il blob non esiste ma è come se stessimo prenotando uno spazio per lui. Successivamente vengono impostate alcune proprietà che saranno utili per individuare il blob come ad esempio la proprietà ContentType che indica il tipo MIME del file e la proprietà ContentLanguage che può essere assegnata con una delle Culture disponibili. Il tipo MIME viene determinato da un altro metodo che illustro tra breve, qui è importante sottolineare come la proprietà Properties sia di tipo BlobProperties e consente, per l'appunto, di settare alcune proprietà per ciascun blob.

Successivamente si istanzia uno stream in lettura che punta al file fisico in locale, dopodiché si invoca un metodo chiamato BeginUploadFromStream che carica in modalità asincrona il file selezionato sullo storage. Come argomenti riceve lo stream aperto, un delegate che viene invocato al termine dell'operazione asincrona e il riferimento all'istanza del blob. Il perché di questo lo vediamo proprio nel codice del delegate:

* 1. Visual Basic
  2. Private Sub UploadAsyncCallBack(ByVal result As IAsyncResult)
  3. If result.IsCompleted Then
  4. Dim blobref = CType(result.AsyncState, CloudBlob)
  5. blobref.SetProperties()
  6. blobref.EndUploadFromStream(result)
  7. End If
  8. End Sub

Se il risultato dell'operazione è che questa è stata completata (IAsyncResult.IsCompleted), si ottiene il riferimento al blob appena caricato. Poi si invocano i metodi CloudBlob.SetProperties e CloudBlob.EndUploadFromStream che, rispettivamente, finalizzano l'assegnazione delle proprietà del blob e indicano al runtime che l'upload è terminato e di rilasciare, quindi, le risorse. Ora vediamo il metodo che determina il tipo MIME di un file:

* 1. Visual Basic
  2. Private Shared Function getMimeType(ByVal fileName As String) As String
  3. 'in case the extension is not registered, then return a default value
  4. Dim mimeType As String = "application/unknown"
  5. Dim fileExtension = System.IO.Path.GetExtension(fileName).ToLower()
  6. Dim registryKey = Registry.ClassesRoot.OpenSubKey(fileExtension)
  7. If registryKey IsNot Nothing And registryKey.GetValue("Content Type") IsNot Nothing Then
  8. mimeType = registryKey.GetValue("Content Type").ToString
  9. End If
  10. Return mimeType
  11. End Function

Niente si speciale, il codice cerca nel registro l'estensione del file e ne determina il tipo MIME oppure restituisce un valore di default se questa non è presente.

Chiaramente è anche possibile scaricare blob dallo storage sul proprio computer. Per quelle che erano le finalità nel mio client, ho scritto due overload di un metodo che ho chiamato DownloadBlobs. Entrambi usano il metodo CloudBlob.BeginDownloadToStream che si occupa del download asincrono di file dal cloud storage. Ecco i due overload:

* 1. Visual Basic
  2. Public Overridable Function DownloadBlobs(ByVal containerName As String, ByVal targetDirectory As String) As IEnumerable(Of String)
  3. Try
  4. Dim container = blobStorage.GetContainerReference(containerName)
  5. Dim blobNames As New List(Of String)
  6. Dim tempStream As IO.FileStream
  7. For Each blob In container.ListBlobs.ToCloudBlobCollection
  8. Dim tempPath = targetDirectory + "\" +
  9. IO.Path.GetFileName(blob.Uri.AbsolutePath)
  10. tempStream = New IO.FileStream(tempPath, IO.FileMode.Create)
  11. blobNames.Add(tempPath)
  12. blob.BeginDownloadToStream(tempStream, AddressOf DownloadAsyncCallBack,
  13. New KeyValuePair(Of CloudBlob, FileStream)(blob, tempStream))
  14. Next
  15. Return blobNames.AsEnumerable
  16. Catch ex As Exception
  17. Throw
  18. End Try
  19. End Function
  20. Public Overridable Sub DownloadBlobs(ByVal containerName As String,
  21. ByVal targetDirectory As String,
  22. ByVal blobsToDownload As IEnumerable(Of CloudBlob))
  23. Dim container = blobStorage.GetContainerReference(containerName)
  24. Dim tempStream As IO.FileStream
  25. For Each blob In blobsToDownload
  26. Dim tempPath = targetDirectory + "\" + IO.Path.GetFileName(blob.Uri.AbsolutePath)
  27. tempStream = New IO.FileStream(tempPath, IO.FileMode.Create) blob.BeginDownloadToStream(tempStream, AddressOf DownloadAsyncCallBack,
  28. New KeyValuePair(Of CloudBlob, FileStream)(blob, tempStream))
  29. Next
  30. End Sub

Entrambi ricevono come argomenti sia il nome del container dal quale scaricare i file sia la directory di destinazione. In particolare il primo overload:

* + scarica tutti i file presenti nel container specificato dallo storage al pc
  + restituisce l'elenco di file sotto forma di insieme di stringhe, nel caso possa servire per l'analisi dei file scaricati
  + cicla l'insieme e inserisce ciascun blob in una lista generica. Per farlo il codice si avvale di un metodo extension chiamato ToCloudBlobCollection, che vi spiego tra poco, che permette di convertire ciascun IListBlobItem in CloudBlob e questo serve per determinare la posizione del blob attraverso la sua proprietà Uri, diversamente non disponibile



Il secondo overload ha più o meno le stesse caratteristiche, solo che scarica solo i blob specificati attraverso un terzo argomento di tipo IEnumerable(Of CloudBlob) e, per tale ragione, non restituisce alcun elenco.

Entrambi i metodi scaricano i blob in modo asincrono sfruttando il metodo BeginDownloadToStream (don't block the UI thread....). Qui ho dovuto usare un piccolo trucco, in barba alla documentazione. Come vedete nell'invocare questo metodo passo l'istanza dello stream, un delegate che viene invocato al termine dell'operazione e un oggetto KeyValuePair(Of CloudBlob, Stream). Il fatto è che, a differenza dell'upload, nel download è necessario chiudere esplicitamente lo stream altrimenti il file scaricato è inutilizzabile finché bloccato.

Al momento in cui questo articolo è stato scritto, la documentazione non tiene conto di questo aspetto (ma non escludiamo aggiornamenti futuri). Infatti il delegate si presenta così:

* 1. Visual Basic
  2. Private Sub DownloadAsyncCallBack(ByVal result As IAsyncResult)
  3. Dim currentResult = CType(result.AsyncState, KeyValuePair(Of CloudBlob, FileStream))
  4. Dim blob = currentResult.Key
  5. blob.EndDownloadToStream(result)
  6. currentResult.Value.Close()
  7. End Sub

In sostanza prendiamo la chiave della coppia chiave/valore, che corrisponde al blob e ne confermiamo la chiusura tramite EndDownloadToStream. Poi riprendiamo il valore nella coppia, che corrisponde allo stream, e lo chiudiamo esplicitamente. Ecco, poi, il metodo extension ToCloudBlobCollection:

* 1. Visual Basic
  2. <Extension()> Public Function ToCloudBlobCollection(ByVal blobItemList \_ As IEnumerable(Of IListBlobItem)) As IList(Of CloudBlob)
  3. Dim c As New List(Of CloudBlob)
  4. For Each element In blobItemList
  5. c.Add(CType(element, CloudBlob))
  6. Next
  7. Return c
  8. End Function

In questo modo convertiamo un insieme di IListBlobItem in un qualcosa di più specifico per i blob. Da ultimo possiamo scrivere un metodo che elimini i blob. Ci sono anche qui diverse possibilità, sincrone (CloudBlob.DeleteIfExists) o asincrone (CloudBlob.BeginDeleteIfExists). Scegliamo ancora la via asincrona, ma siccome siamo dei fighi :-) e ci piace sguinzagliare Visual Basic 2010 sfruttiamo anche una statement lambda al posto del delegate, perché in realtà siamo anche pigri e non ci va di scrivere un metodo a parte :-) Eccolo:

* 1. Visual Basic
  2. Public Sub DeleteBlobs(ByVal Blobs As IEnumerable(Of CloudBlob))
  3. For Each blob In Blobs
  4. Dim blobRef = Me.blobStorage.GetBlobReference(blob.Uri.ToString)
  5. blobRef.BeginDeleteIfExists(Sub(result As IAsyncResult)
  6. CType(result, CloudBlob).EndDeleteIfExists(result)
  7. End Sub, blobRef.Uri)
  8. Next
  9. End Sub

Anche qui andiamo ad eliminare l'elenco di blob passato attraverso una IEnumerable(Of CloudBlob) anche se questa è solo una possibile alternativa.

Abbiamo così visto come sia possibile elencare, caricare, scaricare ed eliminare blob sul proprio storage su Windows Azure. In realtà ci sono ancora due cose da fare:

* + ragionare per eventi, affinché eventuali client siano notificati dell'inizio/fine delle operazioni
  + predisporre un esempietto pratico



Apportiamo ora qualche miglioramento al codice, continuando a ripercorrere la strada che ho fatto nello sviluppare l'applicazione [Azure Blob Studio 2011](http://azureblobstudio.codeplex.com).

Il primo miglioramento che facciamo è l'aggiunta di un metodo che ci permetta di determinare se lo Storage Emulator sia in esecuzione sul nostro sistema; questo può esserci utile per evitare degli errori nel momento in cui tentiamo di accedere allo storage locale con l'account Developer. Il metodo in questione verifica che il processo relativo all'emulator (DSService.exe) sia in esecuzione; in caso negativo, determina il suo percorso leggendolo dal registro di sistema e quindi lo lancia:

* + 1. Visual Basic
    2. Public Shared Function CheckIfDevelopmentStorageIsRunning() As Boolean  
        'If the development storage is not running, then launches it  
        Dim IsDevelopmentStorageRunning = Process.GetProcessesByName("DSService")  
          
          
        If IsDevelopmentStorageRunning.Length = 0 Then  
        Dim sdkPath As String = CStr(My.Computer.  
        Registry.GetValue("HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Microsoft SDKs\ServiceHosting\v1.3",  
        "InstallPath", Nothing))  
        Try  
        Process.Start(sdkPath + "bin\devstore\DSService.exe")  
        System.Threading.Thread.Sleep(4000)  
          
        IsDevelopmentStorageRunning = Nothing  
        Return True  
        Catch ex As Exception  
        Return False  
        End Try  
        'Waiting for the Development Storage to be ready.  
        Else  
        Return True  
        End If  
        End Function

Ho messo, per pura brevità, un ritardo nel thread di 4 secondi durante i quali possiamo star tranquilli che il servizio di storage si attivi. Ci sono altre modalità, come ad esempio metodi e proprietà di istanza della classe Process per accertarci che il processo sia attivo. Qualunque sia il modo prescelto per farlo, è bene accertarsene.

Fatto questo, passiamo a migliorare il codice precedentemente implementato con degli eventi. Una cosa fatta come si deve dovrebbe notificare ai client che l'upload di un blob sta iniziando e che magari è stato completato con successo; stesso dicasi per altre attività. Creiamo innanzitutto una classe chiamata BlobServiceEventArgs che verrà passata agli eventi e che conterrà informazioni sul blob corrente:

* + 1. Visual Basic
    2. Imports Microsoft.WindowsAzure.StorageClient  
          
       Public Class BlobServiceEventArgs  
        Inherits EventArgs  
          
        Private \_blobUri As String  
        Private \_blob As CloudBlob  
          
        Public ReadOnly Property BlobUri As String  
        Get  
        Return Me.\_blobUri  
        End Get  
        End Property  
          
        Public ReadOnly Property Blob As CloudBlob  
        Get  
        Return \_blob  
        End Get  
        End Property  
          
        Public Sub New(ByVal blob As CloudBlob)  
        Me.\_blobUri = blob.Uri.ToString  
        Me.\_blob = blob  
        End Sub  
       End Class

Così facendo siamo in grado di esporre il riferimento al blob e al suo indirizzo ai client che intercettano l'evento. Ora torniamo alla classe di servizio ed aggiungiamo i seguenti eventi:

* + 1. Visual Basic
    2. Public Event UploadBlobStarting(ByVal sender As Object, ByVal e As BlobServiceEventArgs)  
        Public Event UploadBlobCompleted(ByVal sender As Object, ByVal e As BlobServiceEventArgs)  
        Public Event DeleteBlobStarting(ByVal sender As Object, ByVal e As BlobServiceEventArgs)  
        Public Event DeleteBlobCompleted(ByVal sender As Object, ByVal e As BlobServiceEventArgs)  
        Public Event DownloadBlobStarting(ByVal sender As Object, ByVal e As BlobServiceEventArgs)  
        Public Event DownloadBlobCompleted(ByVal sender As Object, ByVal e As BlobServiceEventArgs)

Gli eventi rappresentano l'inizio e la fine di attività inerenti il caricamento, il download e l'eliminazione di blob dallo storage. Dove andiamo a generare gli eventi? Il codice seguente mostra come modificare il metodo UploadBlobs indicando che la generazione dell'evento UploadBlobStarting deve avvenire appena prima dell'inizio dell'upload:

* + 1. Visual Basic
    2. Public Sub UploadBlobs(ByVal ContainerName As String, ByVal BlobList As String())  
        Try  
        Dim container = blobStorage.GetContainerReference(ContainerName)  
          
        For Each blob In BlobList  
        Dim blobRef = container.GetBlobReference(IO.Path.GetFileName(blob))  
        blobRef.Properties.ContentType = getMimeType(blob)  
        blobRef.Properties.ContentLanguage = My.Computer.Info.InstalledUICulture.EnglishName  
          
        RaiseEvent UploadBlobStarting(Me, New BlobServiceEventArgs(blobRef))  
        Dim fs As New FileStream(blob, FileMode.Open)  
        blobRef.BeginUploadFromStream(fs, AddressOf UploadAsyncCallBack, blobRef)  
        Next  
        Catch ex As Exception  
        Throw  
        End Try  
          
        End Sub

Parimenti, nel delegate di callback possiamo generare l'evento UploadBlobCompleted:

* + 1. Visual Basic
    2. Private Sub UploadAsyncCallBack(ByVal result As IAsyncResult)  
        If result.IsCompleted Then  
        Dim blobref = CType(result.AsyncState, CloudBlob)  
        blobref.SetProperties()  
        blobref.EndUploadFromStream(result)  
        RaiseEvent UploadBlobCompleted(Me, New BlobServiceEventArgs(blobRef)) End If  
        End Sub

L'operazione può essere ripetuta in modo analogo per i metodi di download ed eliminazione blob, inoltre si potrebbe prevedere un evento da scatenare nel caso in cui l'operazione fallisca. La cosa interessante da notare è che quando un evento viene scatenato viene passata l'istanza del blob le cui proprietà possono essere analizzate dai client chiamanti (che, ad esempio, possono così sapere quale blob è collegato all'evento).

Sebbene possano esserci ampi margini di miglioramento, l'impostazione data alla classe di servizio per l'accesso al Cloud Storage può dirsi completata.

Pertanto, a questo punto ci proponiamo di illustrare un esempio di client WPF che sfrutti la libreria. L'esempio sarà piuttosto semplice/semplificato. Per un esempio molto più completo, e complesso, potete scaricare il mio progetto [Azure Blob Studio 2011](http://azureblobstudio.codeplex.com) da CodePlex.

Creiamo quindi un progetto WPF in VB 2010 ed aggiungiamo un riferimento alla libreria precedentemente creata (in Azure Blob Studio si chiama DelSole.BlobService.dll) e uno all'assembly Microsoft.WindowsAzure.StorageClient.dll.

Poiché quest'ultimo ha un riferimento all'assembly System.Web, dobbiamo assicurarci che la versione target del Framework sia la full e non la Client Profile (per cambiarla: My Project -> Compile -> Advanced compile options).

L'applicazione sostanzialmente si propone di elencare i container disponibili, di elencare i blob disponibili per container e di caricare/scaricare/rimuovere alcuni blob. I metodi della classe BlobService restituiscono IEnumerable(Of T), dove T sarà CloudBlobContainer per le folder e CloudBlob per i file. Restituendo questo particolare tipo, diventano i candidati ideali per alcuni ItemControl di WPF, uno fra tutti la DataGrid. Ecco quindi come lo XAML definisca due griglie, una che mostra i container e una che mostra i blob:

* 1. XAML
  2. <Grid>  
      <DataGrid AutoGenerateColumns="False" Height="168" HorizontalAlignment="Left" Margin="7,8,0,0" Name="DataGrid1" VerticalAlignment="Top" Width="486" ItemsSource="{Binding}">  
      <DataGrid.Columns>  
      <DataGridTextColumn Header="Uri" Binding="{Binding Path=Uri, Mode=OneWay}" Width="SizeToCells"/>  
      <DataGridTextColumn Header="Name" Binding="{Binding Path=Name, Mode=OneWay}" Width="SizeToCells"/>  
      </DataGrid.Columns>  
      </DataGrid>  
      <DataGrid AutoGenerateColumns="False" Height="182" HorizontalAlignment="Left" Margin="7,185,0,0" Name="DataGrid2" VerticalAlignment="Top" Width="487" ItemsSource="{Binding}" >  
      <DataGrid.Columns>  
      <DataGridHyperlinkColumn Header="Uri" Binding="{Binding Path=Uri, Mode=OneWay}" Width="SizeToCells">  
      <DataGridHyperlinkColumn.ElementStyle>  
      <Style TargetType="TextBlock">  
      <EventSetter Event="Hyperlink.Click" Handler="HyperlinkClick" />  
      </Style>  
      </DataGridHyperlinkColumn.ElementStyle>  
      </DataGridHyperlinkColumn>  
      </DataGrid.Columns>  
      </DataGrid>  
      <Button Content="Upload all pics" Height="37" HorizontalAlignment="Left" Margin="9,387,0,0" Name="UploadButton" VerticalAlignment="Top" Width="116" />  
      <Button Content="Remove all blobs" Height="39" HorizontalAlignment="Left" Margin="132,387,0,0" Name="RemoveButton" VerticalAlignment="Top" Width="118" />  
      <Button Content="Download all blobs" Height="37" HorizontalAlignment="Left" Margin="258,387,0,0" Name="DownloadButton" VerticalAlignment="Top" Width="115" />  
      </Grid>

Nella DataGrid dei blob si usa un HyperLink che punta alla proprietà Uri di ciascun elemento. Chiaramente il data-binding va a prendere solo le proprietà di interesse da mostrare nelle DataGrid, ma potete modificarlo. Giusto per scopo dimostrativo, che poi potete estendere per vostro conto, c'è un pulsante che carica tutte le immagini nella cartella My Pictures. Lato VB, dapprima definiamo l'istanza della classe di servizio e collezioni che ci servono per il binding:

* 1. Visual Basic
  2. Imports Microsoft.WindowsAzure.StorageClient  
     Imports System.Collections.ObjectModel  
        
     Class MainWindow  
        
      Private containerCollection As ObservableCollection(Of CloudBlobContainer)  
      Dim bService As New DelSole.BlobService.BlobService  
      Dim blobCollection As ObservableCollection(Of CloudBlob)

L'istanza della classe di servizio creata senza passare argomenti al costruttore fa riferimento al development storage locale. Fatto questo ci assicuriamo, al caricamento della finestra principale, che l'emulatore locale sia in esecuzione e poi carichiamo dallo storage i vari container, associati alla prima DataGrid:

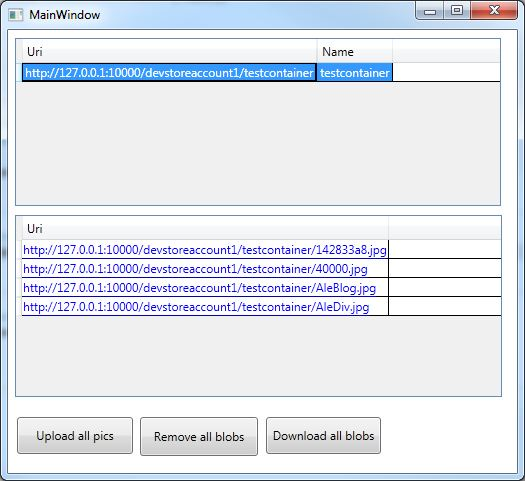
* 1. Visual Basic
  2. Private Sub MainWindow\_Loaded(ByVal sender As Object, ByVal e As System.Windows.RoutedEventArgs) \_  
      Handles Me.Loaded  
      DelSole.BlobService.BlobService.CheckIfDevelopmentStorageIsRunning()  
        
      Me.containerCollection = New ObservableCollection(Of CloudBlobContainer)(bService.ListContainers)  
      Me.DataGrid1.ItemsSource = bService.ListContainers.ToList  
      End Sub

Gestiamo poi il caricamento della lista di blob per container, che si verifica quando viene selezionato un container nell'apposita DataGrid:

* 1. Visual Basic
  2. Private Sub DataGrid1\_SelectionChanged(ByVal sender As Object,  
      ByVal e As System.Windows.Controls.SelectionChangedEventArgs) \_  
      Handles DataGrid1.SelectionChanged  
      Me.blobCollection = New ObservableCollection(Of CloudBlob)(bService.ListBlobs(CType(Me.DataGrid1.SelectedItem, CloudBlobContainer).Name))  
      Me.DataGrid2.ItemsSource = Me.blobCollection  
      End Sub
  3. Private Sub HyperlinkClick()  
      Process.Start(CType(Me.DataGrid2.SelectedItem, CloudBlob).Uri.ToString)  
      End Sub

Il metodo HyperlinkClick sfrutta i relaxed delegate e va ad aprire il contenuto del blob selezionato, a seguito di un cast da Object a CloudBlob. Infine i gestori per i tre pulsanti; il primo carica il contenuto della cartella My Pictures (questo è fatto per brevità, potreste sostituire con una OpenFileDialog e scegliere ciò che preferite), il secondo cancella i blob, il terzo li scarica in una cartella predeterminata:

La seguente figura mostra un esempio dell'applicazione in esecuzione:

* 1. 

Sfruttando quindi l'interazione client con lo storage di Azure è possibile memorizzare e gestire i propri dati in maniera totalmente managed.

#### di [Alessandro Del Sole](https://mvp.support.microsoft.com/profile/Alessandro.Del%20Sole)

* 1. [*Altri articoli di Alessandro Del Sole nella Libreria*](http://sxp.microsoft.com/feeds/3.0/msdntn/TA_MSDN_ITA?contenttype=Article&author=Alessandro%20Del%20Sole) [](http://sxp.microsoft.com/feeds/3.0/msdntn/TA_MSDN_ITA?contenttype=Article&author=Sergio%20Govoni)