Developer Portal Data Services Design Document

Contents

[Introduction 2](#_Toc228706032)

[Why REST? 2](#_Toc228706033)

[Why ATOM? 2](#_Toc228706034)

[Scenario 2](#_Toc228706035)

[End-to-End Code Snippet 2](#_Toc228706036)

[Data Services 6](#_Toc228706037)

[Product List 6](#_Toc228706038)

[Client API 6](#_Toc228706039)

[Products Web Page 7](#_Toc228706040)

[File List 8](#_Toc228706041)

[Client API 8](#_Toc228706042)

[Files Web Page 9](#_Toc228706043)

[Event List 9](#_Toc228706044)

[Client API 10](#_Toc228706045)

[Events Web Pages 12](#_Toc228706046)

[Event Details 13](#_Toc228706047)

[Client API 14](#_Toc228706048)

[Event Details Web Page 14](#_Toc228706049)

[Cab List 15](#_Toc228706050)

[Client API 15](#_Toc228706051)

[Cab List Web Page 16](#_Toc228706052)

[Open Items 17](#_Toc228706053)

[Appendix 17](#_Toc228706054)

[Service Schema 17](#_Toc228706055)

# Introduction

This document describes the design of the data services offering for the Developer Portal. The objective of the data services is to enable ISV’s to pull data and cabs from the Developer Portal in an automated fashion.

The data services will be implemented in [REST](http://www.ietf.org/rfc/rfc4287.txt) style and specifically in the [ATOM](http://msdn2.microsoft.com/en-us/library/ed577840(VS.80).aspx) syndication format.

## Why REST?

We wanted our web services to be easy and simple. Using REST style makes the web services simple since when you need a resource you can parse a URL and get to the resource. That resource may have other URL’s that lead to other resources. The traditional way of implementing web services is using the SOAP protocol, but that protocol is bulky and complicated and does not really serve any purpose if you are not using any of the extensions to SOAP like WS-Trust, WS-Security etc.

Links to additional REST information

* [http://www.xfront.com/REST.html](http://schemas.microsoft.com/developerportal)
* [http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html](http://www.w3.org/2005/Atom)

## Why ATOM?

ATOM is already a standard and is extensible. The Atom Publishing Protocol is also a standard and defines ways to change the data using standard HTTP protocols like GET, POST, DELETE and PUT. Since ATOM is standard for syndication type services, in the future we should be able to leverage that for notifications and subscriptions.

Links to additional ATOM information

* [http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared](http://www.w3.org/2005/Atom)
* [http://www.ietf.org/rfc/rfc4287.txt](http://en.wikipedia.org/wiki/REST) - The ATOM syndication format spec
* [http://www.ietf.org/rfc/rfc5023.txt](http://en.wikipedia.org/wiki/ATOM) - The ATOM Publishing Protocol spec

# Scenario

The primary scenario is the transfer of crash data (event data and cabs) from the developer portal to the ISV. Under the primary scenario the ISV will pull data from the developer portal on a regular basis   
(like daily) and they would only pull modified data. To satisfy this requirement all the data items will have a modified date associated with them.

## End-to-End Code Snippet

//

// get the date when the last pull was done. Assign the date to lastPullDate;

//

DateTime lastPullDate;

//

// folder path to save cabs

//

string cabFolderPath;

//

// cab file name

//

string cabFileName;

//

// get the products

//

ProductCollection products = Product.GetProducts();

//

// loop through the products

//

foreach (Product product in products)

{

//

// check the date created. if it is greater than the last

// pull date then this is a new product and hence insert

//

if (product.DateCreated > lastPullDate)

{

//

// insert new product record

//

}

else if (product.DateModified > lastPullDate)

{

//

// update the product information if product last modified

// date is greater than the last pull date

//

product.TotalResponses;

product.TotalEvents;

product.DateModified;

}

//

// get the files for the product

//

FileCollection files = product.Files;

//

// loop through the files

//

foreach (File file in files)

{

//

// check the date created. if it is greater than the last

// pull date then this is a new file and hence insert

//

if (file.DateCreated > lastPullDate)

{

//

// insert new file record

//

}

//

// get the events for the file with the start date as last pull date + 1

//

EventPageReader eventPageReader = file.GetEvents(lastPullDate.AddDays(1));

//

// loop through the event pages

//

while (eventPageReader.Read() == true)

{

//

// get the events for the page

//

EventReader events = eventPageReader.Events;

//

// loop over the events in the page

//

while (events.Read() == true)

{

//

// get the event

//

Event dpEvent = events.Event;

//

// check the date created. if it is greater than the last

// pull date then this is a new event and hence insert

//

if (dpEvent.DateCreated > lastPullDate)

{

//

// insert new event record

//

}

else if (dpEvent.DateModified > lastPullDate)

{

//

// update the event information if event modified

// date is greater than the last pull date

//

dpEvent.TotalHits;

dpEvent.DateModified;

}

//

// get the details for the event

//

EventInfoCollection infoCollection = dpEvent.Details;

//

// loop through the event info

//

foreach (EventInfo info in infoCollection)

{

//

// insert the event info if event info created

// date is greater than the last pull date

//

if (info.LastCreated > lastPullDate)

{

}

}

//

// get the cabs for the event

//

CabCollection cabs = dpEvent.Cabs;

//

// loop through the cab collection

//

foreach (Cab cab in cabs)

{

//

// get the cab if cab last modified date

// is greater than the last pull date

//

if (cab.LastModified > lastPullDate)

{

//

// get the cab stream

//

System.IO.Stream cabStream = cab.GetCabStream();

//

// OR

// save the cab to a folder

//

cab.SaveCab(cabFolderPath);

//

// OR

// save ghe cab to a folder and a file name

//

cab.SaveCab(cabFolderPath, cabFileName);

}

}

}

}

}

}

# Data Services

The following data services will be covered in the current release:

1. Product List
2. File List
3. Event List
4. Event Details
5. Cab List
6. Cab Download

Each item in the list above is described below and the information flow is from the client API to the database level.

Client API Namespace: Microsoft.WindowsErrorReporting.Services.Data

## Product List

The product list will contain the list of products mapped by the company. The following data will be included for each product in the product list:

1. Product name
2. Product version
3. Product id
4. Date created
5. Date modified
6. Total events
7. Total responses

### Client API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Product class** | | | | |
| Type | Name | Return | Parameters | Description |
| Property | ProductID | int | N/A | Unique identifier for the product |
| Property | ProductName | String | N/A | Name of the product. |
| Property | ProductVersion | String | N/A | Product version. |
| Property | DateCreated | DateTime | N/A | Date-time the product definition was mapped for the company. |
| Property | DateModified | DateTime | N/A | Date-time the product was changed – happens when either the TotalEvents or TotalResponses properties are updated |
| Property | TotalEvents | int | N/A | Total events for the product |
| Property | TotalResponses | int | N/A | Total responses for the product. |
| Property | Files | FileCollection | N/A | List of files mapped to the product. Internally calls the File.GetFiles(productID) method. |
| Static Method | GetProducts() | ProductCollection | N/A | Static method to get the list of products mapped to the company. Load the /feed/products.aspx page and create the collection. |

### Products Web Page

**Relative URL:**

/member/wer/user/feed/products.aspx

#### REST API

**Sample content:**

<?xml version="1.0" encoding="utf-8" ?>

<feed xmlns=”[http://www.w3.org/2005/Atom](mailto:mfernan@microsoft.com)” xmlns:dp=”http://winqual.microsoft.com/member/wer/user/feed”>

<title>Product List</title>

<updated>2007-11-09T22:35:00Z</updated>

<id>/member/wer/user/feed/products.aspx</id>

<entry>

<title>BigFiles\_And\_UserDocs\_fromGPFMe</title>

<link>/member/wer/user/feed/files.aspx?productid=4997</link>

<id>4997</id>

<updated>2007-11-09T22:35:00Z</updated>

<published>2007-11-09T22:35:00Z</published>

<dp:totalResponses>10</dp:totalResponses>

<dp:totalEvents>10</dp:totalEvents>

<dp:productVersion>1.1.0.0</dp:productVersion>

</entry>

<entry>

<title>ca</title>

<link>/member/wer/user/feed/events.aspx?productid=4235</link>

<id>4235</id>

<updated>2007-11-09T22:35:00Z</updated>

<published>2007-11-09T22:35:00Z</published>

<dp:totalResponses>10</dp:totalResponses>

<dp:totalEvents>10</dp:totalEvents>

<dp:productVersion>10.0.0.0</dp:productVersion>

</entry>

</feed>

**Class Property – XML Path Mapping:**

|  |  |
| --- | --- |
| **Property** | **Path** |
| ProductName | /feed/entry/title |
| ProductVersion | /feed/entry/pd:productVersion |
| ProductID | /feed/entry/id |
| DateCreated | /feed/entry/published |
| DateModified | /feed/entry/updated |
| TotalResponses | /feed/entry/dp:totalResponses |
| TotalEvents | /feed/entry/dp:totalEvents |

## File List

The file list will contain the list of files mapped to a product by the company. The following data will be included for each file in the file list:

1. File ID
2. File name
3. File version
4. File link date
5. Date created
6. Date modified

### Client API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **File class** | | | | |
| Type | Name | Return | Parameters | Description |
| Property | FileID | int | N/A | Unique identifier for the file |
| Property | FileName | String | N/A | Name of the file. |
| Property | FileVersion | Version | N/A | File version. |
| Property | FileLinkDate | DateTime | N/A | Link date for the file. |
| Property | DateCreated | DateTime | N/A | Date-time the file was mapped to the product. |
| Property | DateModified | DateTime | N/A | Date-time the product was changed – for now this will be the same as the DateCreated property. |
| Static Method | GetFiles | FileCollection | productID int | Static method to get the list of files mapped to a product. Load the /feed/files.aspx?productid page and load the collection. |
| Method | GetEvents | EventPageReader | N/A | Get all events for the file ordered by date ascending and total hits ascending. |
| Method | GetEvents | EventPageReader | startDate DateTime | Get all events for the file from the “startDate”. |
| Method | GetEvents | EventPageReader | startDate – DateTime  endDate - DateTime | Get all events for the file from the “startDate” till the “endDate”. |

#### Code snippet

EventPageReader pageReader = File.GetEvents(startDate);

while (pageReader.Read() == true)

{

EventReader eventReader = pageReader.Events;

while (eventReader.Read() == true)

{

Event event = eventReader.Event;

event.EventID;

event.EventTypeName;

}

}

### Files Web Page

**Relative URL:**

/member/wer/user/feed/files.aspx

#### REST API

**Sample content:**

<?xml version="1.0" encoding="utf-8" ?>

<feed xmlns=”[http://www.w3.org/2005/Atom](https://winqual.microsoft.com/member/wer/user)” xmlns:dp=”http://winqual.microsoft.com/member/wer/user/feed”>

<title>File List for Product ID {}</title>

<updated>2007-11-09T22:35:00Z</updated>

<id>/member/wer/user/feed/files.aspx?productid=10</id>

<entry>

<title>YServer.exe</title>

<link>/member/wer/user/feed/eventgroups.aspx?fileid=3230</link>

<id>3230</id>

<updated>2007-11-09T22:36:00Z</updated>

<published>2007-11-09T22:36:00Z</published>

<dp:fileVersion>4.5.1.0</dp:fileVersion>

<dp:fileLinkDate>2007-11-09T22:36:00Z</dp:fileLinkDate>

</entry>

<entry>

<title>YahooMessenger.exe</title>

<link>/member/wer/user/feed/eventgroups.aspx?fileid=6546</link>

<id>6546</id>

<updated>2007-11-09T22:36:00Z</updated>

<published>2007-11-09T22:36:00Z</published>

<dp:fileVersion>6.5.1.0</dp:fileVersion>

<dp:fileLinkDate>2007-11-09T22:36:00Z</dp:fileLinkDate>

</entry>

</feed>

**Class Property – XML Path Mapping:**

|  |  |
| --- | --- |
| **Element** | **Path** |
| FileID | /feed/entry/id |
| FileName | /feed/entry/title |
| FileVersion | /feed/entry/dp:fileVersion |
| FileLinkDate | /feed/entry/dp:fileLinkDate |
| DateCreated | /feed/entry/published |
| DateModified | /feed/entry/updated |

## Event List

The event list will contain the list of events for the file.

The events are grouped by their last modified date and within each group they are paged by 1000 events per page.

The event groups are ordered in last modified ascending. Within each last modified group the events are ordered by total hits ascending.

Each event group (by last modified) represents a snapshot for the last modified date.

The reason to do such a grouping is to enable us to cache these snapshots into files that can be loaded on demand. This should enable us to scale and reduce the traffic to the database. This feature will **not** be done in this release.

The following data will be included for each event in the event list:

1. Event ID
2. Event type name
3. Total hits
4. Date created
5. Date modified
6. Event signature

### Client API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Event class** | | | | |
| Type | Name | Return | Parameters | Description |
| Property | EventID | int | N/A | Unique identifier for the event. Corresponds to the bucket id in Watson. |
| Property | EventTypeName | String | N/A | Event type for the event e.g. Crash 32bit, Managed Crash etc. |
| Property | DateCreated | DateTime | N/A | Date-time the event was created. This will be the date when we pulled the event from Watson. |
| Property | DateModified | DateTime | N/A | Date-time the event was changed – happens when the TotalHits property is updated. |
| Property | TotalHits | int | N/A | Total hits for the event. |
| Property | Signature | EventSignature | N/A | Event signature |
| Property | Cabs | CabCollection | N/A | List of cabs for the event. |
| Property | Details | EventInfoCollection | N/A | Event details for the event. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EventSignature class** | | | | |
| Type | Name | Return | Parameters | Description |
| Property | Parameters | ParameterCollection | N/A | Collection containing the event signature. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter class** | | | | |
| Type | Name | Return | Parameters | Description |
| Property | Name | String | N/A | Name of the parameter e.g. AppName |
| Property | Value | String | N/A | Value of the parameter e.g. Photoshop.exe |

### Events Web Pages

The event pages are divided into the event group page and the events page.

**Event Groups Relative URL:**

/member/wer/user/feed/eventgroups.aspx?fileid={fileid}

#### REST API for Event Groups

**Sample content:**

<?xml version="1.0" encoding="utf-8" ?>

<feed xmlns=”[http://www.w3.org/2005/Atom](http://www.ietf.org/rfc/rfc5023.txt)” xmlns:dp=”http://winqual.microsoft.com/member/wer/user/feed”>

<title>Event Group List for File ID </title>

<updated>2007-11-09T22:35:00Z</updated>

<id>/member/wer/user/feed/eventgroups.aspx?fileid=3230</id>

<entry>

<title>Events for {date}</title>

<link>/member/wer/user/feed/events.aspx?fileid=3230&date={date}</link>

<id>{date}</id>

<dp:totalEvents>{total events modified for that date}</dp:totalEvents>

<updated>2007-11-09T22:36:00Z</updated>

<published>2007-11-09T22:36:00Z</published>

</entry>

<entry>

<title>Events for {date}</title>

<link>/member/wer/user/feed/events.aspx?fileid=3230&date={date}</link>

<id>{date}</id>

<dp:totalEvents>{total events modified for that date}</dp:totalEvents>

<updated>2007-11-09T22:36:00Z</updated>

<published>2007-11-09T22:36:00Z</published>

</entry>

</feed>

**Events Relative URL:**

/member/wer/user/feed/events.aspx?fileid={fileid}&date={datemodified}&page={pagenumber}

#### REST API for Events

**Sample content:**

<?xml version="1.0" encoding="utf-8" ?>

<feed xmlns=”[http://www.w3.org/2005/Atom](mailto:zubini@microsoft.com)” xmlns:dp=”http://winqual.microsoft.com/member/wer/user/feed”>

<title>Events File ID , Date , Page </title>

<updated>2007-11-09T22:35:00Z</updated>

<id>/member/wer/user/feed/events.aspx?fileid=10&date={date}&page={pagenum}</id>

<link rel=”previous” href=”/member/wer/user/feed/events.aspx?fileid=10&date={date}&page={pagenum - 1}” />

<link rel=”next” href=”/member/wer/user/feed/events.aspx?fileid=10&date={date}&page={pagenum + 1}” />

<dp:totalPages>10</dp:totalPages>

<dp:currentPage>1</dp:currentPage>

<entry>

<title>Event ID – Event Type - </title>

<link rel=”cabs” href=”member/wer/user/feed/cabs.aspx?eventid=3230&eventtype={eventtype}” />

<link rel=”details” href=”/member/wer/user/feed/eventdetails.aspx?eventid=3230&eventtype={eventtype}” />

<id>{eventid-eventtype}</id>

<updated>2007-11-09T22:36:00Z</updated>

<published>2007-11-09T22:36:00Z</published>

<dp:eventID>320309</dp:eventID>

<dp:eventType>Crash 32bit</dp:eventType>

<dp:totalHits>3203093243</dp:totalHits>

<dp:signature>

<dp:parameter name=”applicationName” value=”” />

<dp:parameter name=”applicationVersion” value=”” />

<dp:parameter name=”applicationTimeStamp” value=”” />

<dp:parameter name=”moduleName” value=”” />

<dp:parameter name=”moduleVersion” value=”” />

<dp:parameter name=”moduleTimeStamp” value=”” />

<dp:parameter name=”exceptionCode” value=”” />

<dp:parameter name=”offset” value=”” />

</dp:signature>

</entry>

</feed>

**Class Property – XML Path Mapping:**

|  |  |  |
| --- | --- | --- |
| **Element** | **Property** | **Path** |
| Feed title |  | /feed/title |
| Date Updated |  | /feed/updated |
| Feed ID |  | /feed/id |
| Link to previous page |  | /feed/link[rel=”previous”] |
| Link to next page |  | /feed/link[rel=”next”] |
| Total pages |  | /feed/dp:totalPages |
| Current page | CurrentPageNumber | /feed/dp:currentPage |
| Individual events | Event | /feed/entry |
| Entry title |  | /feed/entry/title |
| Link to cabs | Cabs | /feed/entry/link[rel=”cabs”] |
| Link to details | EventDetails | /feed/entry/link[rel=”details”] |
| Entry id |  | /feed/entry/id |
| Date updated | DateModified | /feed/entry/updated |
| Date created | DateCreated | /feed/entry/published |
| Event id | EventID | /feed/entry/dp:eventID |
| Event type | EventTypeName | /feed/entry/dp:eventType |
| Total hits | TotalHits | /feed/entry/dp:totalHits |
| Event signature | Signature | /feed/entry/dp:signature |
| Signature parameters | Signature.Parameters | /feed/entry/dp:signature/dp:parameter |
| Signature parameter attribute “name” | Parameter.Name | /feed/entry/dp:signature/dp:parameter[“name”] |
| Signature parameter attribute “value” | Parameter.Value | /feed/entry/dp:signature/dp:parameter[“value”] |

## Event Details

The event details will contain the time trend information for the event. The following data will be included for each event in the event details:

1. Event id
2. Event type name
3. Total hits
4. Date created – NOTE: Since Watson does not keep the actual date when the event was created; this date will be the date when the event was imported into Developer Portal.
5. Date modified
6. Operating system name
7. Operating system version
8. LCID – e.g. 3055
9. Locale – e.g. en-us
10. Language – e.g. English – United States

### Client API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EventInfo class** | | | | |
| Type | Name | Return | Parameters | Description |
| Property | EventID | int | N/A | Event id |
| Property | EventTypeName | String | N/A | Type of event e.g. Crash 32bit |
| Property | TotalHits | int | N/A | Total hits for the event. |
| Property | DateCreated | DateTime | N/A | Date-time the event was created. This will be the date when we pulled the event from Watson. |
| Property | DateModified | DateTime | N/A | Date-time the event was changed – happens when the TotalHits property is updated. |
| Property | OperatingSystemName | String | N/A | Name of the operating system. |
| Property | OperatingSystemVersion | String | N/A | Version of the operating system. |
| Property | LCID | int | N/A | Locale id e.g. 3055 |
| Property | Locale | String | N/A | Locale e.g. en-us |
| Property | Language | String | N/A | Language e.g. English – United States |

### Event Details Web Page

**Relative URL:**

/member/wer/user/feed/eventdetails.aspx?eventid={eventid}&eventtypename={eventtypename}

NOTE: This event type name is the friendly name of the event.

#### REST API

**Sample content:**

<?xml version="1.0" encoding="utf-8" ?>

<feed xmlns=”[http://www.w3.org/2005/Atom](http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html)” xmlns:dp=”http://winqual.microsoft.com/member/wer/user/feed”>

<title>Events Details for Event ID , Event Type </title>

<updated>2007-11-09T22:35:00Z</updated>

<id>/member/wer/user/feed/eventdetails.aspx?eventid=3434646&eventtype={eventtype}</id>

<dp:eventID>3434646</dp:eventID>

<dp:eventType>Crash 32bit</dp:eventType>

<entry>

<title>Total hits for date </title>

<id>hit date</id>

<updated>2007-11-09T22:36:00Z</updated>

<published>2007-11-09T22:36:00Z</published>

<dp:totalHits>3203093243</dp:totalHits>

<dp:operatingSystem name=”” version=”” />

<dp:language name=”” lcid=”” locale=”” />

</entry>

</feed>

**Class Property – XML Path Mapping:**

|  |  |  |
| --- | --- | --- |
| **Property** | **Path** | **Description** |
| EventID | /feed/dp:eventID | Event ID |
| EventTypeName | /feed/dp:eventType | Event type |
| DateModified | /feed/entry/updated | Hit date |
| DateCreated | /feed/entry/published | Hit date |
| TotalHits | /feed/entry/dp:totalHits | Total hits for the hit date |
| OperatingSystemName | /feed/entry/dp:operatingSystem[“name”] | Operating system name |
| OperatingSystem Version | /feed/entry/dp:operatingSystem[“version”] | Operating system version |
| Language | /feed/entry/dp:language[“name”] | Language name |
| LCID | /feed/entry/dp:language[“lcid”] | Language lcid |
| Locale | /feed/entry/dp:language[“locale”] | Language locale |

## Cab List

The cab list will contain the list of cabs available for download for the event. The following data will be included for each cab in the cab list:

1. Cab id
2. File name
3. Date created
4. Date modified
5. Size in bytes

### Client API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cab class** | | | | |
| Type | Name | Return | Parameters | Description |
| Property | CabID | int | N/A | Cab id |
| Property | FileName | String | N/A | Name of the cab file. |
| Property | DateCreated | DateTime | N/A | Date-time the event was created. This will be the date when we pulled the event from Watson. |
| Property | DateModified | DateTime | N/A | Date-time the event was changed – happens when the TotalHits property is updated. |
| Property | SizeInBytes | long | N/A | Size of the cab file in bytes. |
| Static Method | GetCabs | CabCollection | eventID – int  eventTypeName – string | Get the cab collection for the event. |
| Method | GetCabStream | System.IO.Stream | N/A | The cab stream. |
| Method | SaveCab | bool | folderPath – string | Folder path to save the cab file to. In this case the cab file name will be in the <bucketed>-<buckettableid>-<Cabid>.cab format. |
| Method | SaveCab | bool | folderPath – string  filename – string | Same as above, but the cab file name can also be specified. |

### Cab List Web Page

**Relative URL:**

/member/wer/user/feed/cabs.aspx?eventid={eventid}&eventtypename={eventtypename}

#### REST API

**Sample content:**

<?xml version="1.0" encoding="utf-8" ?>

<feed xmlns=”<http://www.w3.org/2005/Atom>” xmlns:dp=”http://winqual.microsoft.com/member/wer/user/feed”>

<title>Cab List for Event ID , Event type</title>

<updated>2007-11-09T22:35:00Z</updated>

<id>/member/wer/user/feed/cabs.aspx?eventid=10243435&eventtype={eventtype}</id>

<entry>

<title>Cab ID – {cabid}</title>

<id>{cabid}</id>

<updated>2007-11-09T22:36:00Z</updated>

<published>2007-11-09T22:36:00Z</published>

<link rel=”enclosure”

type=”application/octet-stream”

title=”{cabid}”

href=”/member/wer/user/feed/cab.aspx?cabID={cabid}&bid={eventid}&btid={eventtypeid}&size={size}”

length=”” />

</entry>

</feed>

**Class Property – XML Path Mapping:**

|  |  |  |
| --- | --- | --- |
| **Property** | **Path** | **Description** |
| Cab | /feed/entry | Individual cab entries |
| CabID | /feed/entry/id | Unique identifier for the entry, this will be the Cab ID |
| DateModified | /feed/entry/updated | Date the cab was created |
| DateCreated | /feed/entry/published | Date the cab was created |
| Cab link | /feed/entry/link | Link to the cab as defined by the href attribute. This will be of type enclosure |
| SizeInBytes | /feed/entry/link[“length”] | Size of the cab |

#### Web Page details

Use the HttpWebRequest object to request the URL from the server. Get the response and if the call is to:

**GetCabStream** – then return the data stream from the response as a System.IO.Stream object.

**SaveCab(string folderPath)** – get the stream from response and use the FileStream class to save the cab stream into a file in the specified folder. Use the following for the file name: <bucketid>-<buckettableid>-<cabid>.cab

**SaveCab(string folderPath, string filename)** – same as the SaveCab method above with the exception that we will save to the filename specified as the argument.

We will not put any limit on the size of the cab that can be downloaded. In the future we will think about adding chunking to downloading cabs to enable download of large cab files in small chunks.

# Appendix

## Service Schema

The base schema for the data services will be the ATOM syndication format schema located here: [http://www.ietf.org/rfc/rfc4287.txt](http://www.w3.org/2005/Atom)

We will have our own extensions to this schema. Our extensions will have the following namespace: [http://schemas.microsoft.com/developerportal](http://www.w3.org/2005/Atom)