

Twingly API

LiveFeed API



Author: Björn Milton (bjorn.milton@twingly.com)
Rev: 0.3
2010-07-02

INTRODUCTION

The LiveFeed API (<http://api.twingly.com/livefeed3.asmx>) is a web service based SOAP API used for receiving a continuous stream of blog data. The API has four different methods for retrieving data where the most common one to use is `GetDataByPostCountAndTimespan`.

To be able to retrieve data through LiveFeed an API key issued by Twingly must be used. The API key then grants access to blog data for one or more languages. The amount of data to fetch can also be reduced by only subscribing to posts from the top x blog with the highest authority values for a language. So, a subscription can for example include data for the 10 000 blogs with the highest authority written in Swedish.

INTEGRATION

The main purpose of the LiveFeed API is to provide subscribers with a continuous stream of data as it enters Twingly's indexes. The responsibility of retrieving the data in a correct manner so that nothing is missed lies entirely on the client. To facilitate this, the LiveFeed API exposes four different methods that are described below.

`GETDATA (APIKEY, TIMESTAMP)`

The `GetData` method takes two arguments:

- `apiKey` - (string) The API key used for authentication.
- `timestamp` - (dateTime) The timestamp from where to fetch data.

This method will return all posts that has entered Twingly's systems and that the API key grants access to. However, there are limitations built in so that the time window created by `timestamp` and the time when the call was made cannot be greater than 24 hours. This means that with this method only data newer than 24 hours can be retrieved.

`GETDATABYPOSTCOUNT (APIKEY, TIMESTAMP, MAXPOSTS)`

The `GetDataByPostCount` method takes three arguments:

- `apiKey` - (string) The API key used for authentication.
- `timestamp` - (dateTime) The timestamp from where to fetch data.
- `maxPosts` - (int) The max number posts to retrieve.

This method has the same semantics as `GetData`. The only difference is that it will return at most `maxPost` items. So, if an invocation of `GetDataByPostCount` will return 1000 items, and `maxPost` is set to 100, then only the 100 oldest items will be included in the results.

```
GETDATABYTIMESPAN (APIKEY, FROM, TO)
```

The `GetDataByTimespan` takes three arguments:

- `apiKey` - (string) The API key used for authentication.
- `from` - (dateTime) The timestamp from where to fetch data.
- `to` - (dateTime) The timestamp to where to fetch data.

The `GetDataByTimespan` is used to retrieve data between two timestamps. The time window created by the two timestamps is restricted to at most 24 hours. If a wider time window is given the window will be shrunk by reducing the `to` timestamp.

```
GETDATABYPOSTCOUNTANDTIMESTAMP (APIKEY, FROM, TO, MAXPOSTS)
```

The `GetDataByPostCountAndTimespan` takes four arguments:

- `apiKey` - (string) The API key used for authentication.
- `from` - (dateTime) The timestamp from where to fetch data.
- `to` - (dateTime) The timestamp to where to fetch data.
- `maxPosts` - (int) The max number posts to retrieve.

This method is the most commonly used method for fetching a continuous stream of data. This is because it lets the client control both the window of time and the max number of posts to handle at each time.

BEST PRACTICE - FULL FEED INTEGRATION

In this section one approach of getting a continuous stream of data from LiveFeed is described. The example code is given in C#.

- Use a time window of a certain size, 10 minutes for example.
- Fetch at most 1000 posts in each request.
- For the first invocation, invoke it with a `from` timestamp that has the value of the current timestamp subtracted by 60 minutes. Set the value of the `to` parameter to the value of the `from` parameter added with 10 minutes.

In code this would look like this:

```
var from = DateTime.Now.ToUniversalTime().AddMinutes(-60);  
var to = from.AddMinutes(10);
```

- When the result is returned, set the `from` parameter to the values of the `lastPost` and `lastPostMs` attributes in the result and add one millisecond. Set the `to` parameter to the value of `from` and add 30 minutes.

In C# this would look like this:

```
XmlNode data = client.GetDataByPostCountAndTimespan(
    "the api key",
    from,
    to,
    maxNoOfPosts);

var lastPostTs = DateTime.
    Parse(data.Attributes["lastPost"].InnerText).
    ToUniversalTime();

var lastPostMs = Double.
    Parse(data.Attributes["lastPostMs"].InnerText);

from = lastPostTs.AddMilliseconds(lastPostMs);
```

FULL INTEGRATION EXAMPLE

A very simple full example (without parsing of the result and error handling) follows below. Note that the LiveFeed class instantiated on the 3rd row below is the proxy object for the web service.

```
public void LivefeedExample()
{
    var client = new LiveFeed();

    var from = DateTime.Now.ToUniversalTime().AddMinutes(-60);
    var to = from.AddMinutes(10);

    var maxNoOfPosts = 1000;

    while (true)
    {
        //fetch the data
        XmlNode data = client.GetDataByPostCountAndTimespan(
            "the api key",
            from,
            to,
            maxNoOfPosts);

        //parse the result and do something with it
        //(not implemented here).

        //set the to and from paramaters to new values
        //based on the timestamp for the last (newest)
        //post in the result.
        var lastPostTs = DateTime.
            Parse(data.Attributes["lastPost"].InnerText).
            ToUniversalTime();

        var lastPostMs = Double.
            Parse(data.Attributes["lastPostMs"].InnerText);

        from = lastPostTs.AddMilliseconds(lastPostMs + 1);

        from = lastPostTs.AddMilliseconds(1);
        to = from.AddMinutes(10);

        Thread.Sleep(10 * 60 * 1000); //sleep ten minutes
    }
}
```

With this approach you will get a continuous stream of all the data that the API key grants access to. Of course, error handling must be added to handle network outages, server failure etc.

THE RESULT XML

The returned result is an XML document containing one or more posts. Below is an explanation of the various elements and attributes that can be present in the result.

Note that in older versions, the result can look different. Only the current version is documented here.

The structure of the XML document is as follows:

```
<twinglydata
  ts=""
  tsMs=""
  from=""
  fromMs=""
  to=""
  toMs=""
  type=""
  noOfPosts=""
  maxNumberOfPosts=""
  firstPost=""
  firstPostMs=""
  lastPost=""
  lastPostMs="">
  <post>
    <id></id>
    <url></url>
    <title></title>
    <summary></summary>
    <languageCode></languageCode>
    <countryCode></countryCode>
    <date></date>
    <blogName></blogName>
    <blogUrl></blogUrl>
    <blogRank></blogRank>
    <approved></approved>
  </post>
</twinglydata>
```

<TWINGLYDATA>

The `<twinglydata>` element is the root element. It has several attributes:

- `ts` and `tsMs` – These two attributes together denote the exact time when the result was built.
- `from` and `fromMs` – The `from` timestamp that was sent in to the service.
- `to` and `toMs` – The `to` timestamp that was sent in to the service. These attributes will only be present if a value for the `to` parameter was given.
- `type` – Denotes the type of the result. This attribute can only have one value at present, *stream*.
- `noOfPosts` – The actual number of posts included in the result.

- `maxNumberOfPosts` – The `maxPosts` value sent in to the service. If no value was given, this attribute will not be present.
- `firstPost` and `firstPostMs` – The timestamp for the first post (the oldest) in the result. These attributes will only be present if a value for `maxPosts` was given.

NOTE! The timestamps for `firstPost` and `lastPost` has nothing to do with the `<date>` element in the post data. The `<date>` element is the date of publication for the post. The `firstPost` and `lastPost` values and how the posts in a result are sorted are based on when the posts was indexed by Twingly.

- `lastPost` and `lastPostMs` – The timestamp for the last post (the newest) in the result. These attributes will only be present if a value for `maxPosts` was given.

Within the root element `<twinglydata>` there can be zero or more `<post>` elements.

`<POST>`

The `<post>` is the root element for a post.

`<URL>`

The `<url>` element contains the URL to the post.

`<TITLE>`

The `<title>` element contains the title of the post.

`<SUMMARY>`

The `<summary>` element contains the summary of the post. Through LiveFeed API, the full content of the summary is returned.

`<LANGUAGECODE>`

The `<languageCode>` element contains the ISO language code that represents the language that the post was written in.

`<COUNTRYCODE>`

The `<countryCode>` element contains an ISO country code that has been mapped from the language code.

`<BLOGURL>`

The `<blogUrl>` element contains the URL to the blog.

`<BLOGNAME>`

The `<blogName>` element contains the name of the blog.

<BLOGRANK>

The <blogRank> element contains the *BlogRank* value for the blog. The *BlogRank* value is a value from 1 to 10 indicating how important or influential the blog is.

<APPROVED>

The <approved> element indicates if the blog is in Twingly's spam-free index or not.