Google Reader API

Nick Bradbury
Last Updated 11-Sept-2009

Authentication

Authentication against the GReader API requires a SID (session ID) token and a T token. The SID token is retrieved using a HTTPS POST to Google's ClientLogin service, which in FeedDemon's case looks like this:

```
POST /accounts/ClientLogin HTTP/1.0
HOST www.google.com
Content-type: application/x-www-form-urlencoded

accountType=GOOGLE&Email=nick.bradbury&Passwd=xxxxx&service=reader&source=NewsGator-FeedDemon-2.8
```

Note: Make sure to URL encode the login.

Documentation of the ClientLogin service may be found at http://code.google.com/apis/accounts/docs/AuthForInstalledApps.html. The SID token can be parsed from the response, which will be something like this:

```
HTTP/1.0 200 OK
Server: GFE/1.3
Content-Type: text/plain

SID=DQAAAGgA...7Zg8CTN
LSID=DQAAAGsA...lk8BBbG
Auth=DQAAAGgA...dk3fA5N
```

Once you have a SID token, you can request the T token with a GET request to http://www.google.com/reader/api/0/token which adds the SID token as a cookie header, like this:

```
GET /reader/api/0/token
HOST www.google.com
Cookie: SID=DQAAAGgA...7Zg8CTN
```

The entire response will be the T token.

Once you have valid SID and T tokens, you can perform any operation against the GReader API. The SID token is generated each time you login to Google and *should* be valid until the client application logs out. However, the T token can expire frequently, which means that a GRAPI call may fail due to an expired T token. You can detect a bad token by checking the response header for "`X-Reader-Google-Bad-Token: true`".

NGAPI / GRAPI Equivalents

Notes:

- The SID **must** be included in a cookie header for all calls.
- The query string for all calls **must** include the T token (`&T=`).
- Unless otherwise stated, all calls use base URL of `http://www.google.com/reader/api/0/` (ex: `http://www.google.com/reader/api/0/subscription/list/`)
- The client name can optionally be appended to each query string (`&client=`). I'm not sure what this does since the same information could be obtained using the `User-Agent` header, but I'm adding it anyway as `&client=FeedDemon`. GReader uses the client name "`scroll`" (previous version used "`lens`").
- GReader treats a feed's URL as its feed ID, and a post's Atom entry ID as its post ID
- GReader folders are really tags (or "labels" in GReader terms), which means that the same feed can exist in multiple folders
- All calls accept a `&ck=[timestamp]` parameter, which according to http://code.google.com/p/pyrfeed/wiki/GoogleReaderAPI is "the current time stamp, probably used as a quick hack to be sure that cache won't be triggered."

GetSubscriptionList (verified)

```
GET subscription/list
```

Use `output=json` or `output=xml` to the query string to specify the format. Xml response contains separate <object> nodes for each feed. Ex:

```
<object>
  <string name="id">feed/http://feeds.feedburner.com/NickBradbury</string>
  <string name="title">Nick Bradbury</string>
  <list name="categories">
    <object>
      <string name="id">user/[USER_ID]/label/Bradbury Software</string>
      <string name="label">Bradbury Software</string>
```

```
      </object>
    </list>
    <string name="sortid">C430EC8F</string>
    <number name="firstitemmsec">1203460831522</number>
  </object>
```

The name of the folder is stored in the `<list name="categories">` node as `<string name="label">`. Feeds in the root folder have an empty `<list name="categories"/>` node. The `firstitemmsec` attribute is the timestamp of the oldest (first) item in that subscription for that user.

A simpler option is to call `GET http://www.google.com/reader/subscriptions/export` to retrieve the user's subscriptions in OPML format, but this doesn't include any GReader data (such as `firstitemmsec`).

GetNews/Feed.aspx (verified)

See **Feed Parsing** section below for details.

AddSubscription (verified)

```
POST subscription/edit ac=subscribe&s=feed/[feedUrl]&a=user/-/label/[folder]&t=[feedTitle]
```

To add a subscription without specifying the title or folder, use:

```
POST subscription/quickadd quickadd=feed/[feedUrl]
```

DeleteSubscription (verified)

```
POST subscription/edit ac=unsubscribe&s=feed/[feedUrl]
```

To delete multiple feeds in one call, use:

```
POST subscription/edit?client=settings ac=unsubscribe&s=[feedUrl1]&s=[feedUrl2]...
```

MoveSubscription (verified)

```
POST subscription/edit ac=edit&s=feed/[feedUrl]&r=user/-/label/[oldfolder]&a=user/-/label/[newfolder]
```

RenameSubscription (verified)

```
POST subscription/edit ac=edit&s=feed/[feedUrl]&t=[feedTitle]
```

CreateFolder (questionable)

```
POST edit-tag ac=edit&a=user/-/label/[folder]&s=feed/[feedUrl]
```

Note: Fails unless a feed is passed (ie: can't create an empty folder – have to add a feed to it)

DeleteFolder (verified)

`POST disable-tag ac=disable-tags&s=user/-/label/[folder]&t=[folder]`

Note: This does not delete the feeds – only the folder itself is removed.

MoveFolder (no equivalent)

GReader doesn't support subfolders.

RenameFolder (verified)

No simple way to do this – in fact, GReader itself doesn't let you rename a folder. Instead, you have to move every feed to a folder with the new name, then delete the old folder name (see **MoveSubscription** and **DeleteFolder**).

GetFolders (questionable)

Don't see a way to request just the folder names, but you can send a GET request to `tag/list` to get list of all tags and then extract the folders from the list by looking for the tags which are labels. Ex:

```
<object>
  <list name="tags">
     <object>
       <string name="id">user/-/state/com.google/starred</string>
       <string name="sortid">2F2AE32B</string>
     </object>
     <object>
       <string name="id">user/-/label/Folder Name</string>
       <string name="sortid">89C444DA</string>
     </object>
  </list>
</object>
```

One problem with this: I don't see a way to determine when a label is a folder name and when it's a tag that's been used on a post.

MergeSubscriptions / ImportSubscriptions (verified)

There isn't a direct way to do this, but there is way to add multiple feeds to a single folder in one call. With some extra work, the client can use this to approximate a merge.

```
POST subscription/edit a=user/-/label/[folderName]&ac=subscribe&s=feed/[feedUrl1]&s=feed/[feedUrl2]...
```

Note: To avoid adding duplicate feeds, the client should remove feeds that already exist in the user's subscriptions before performing this operation.

Another option (untested) is to duplicate the POST used by GReader's OPML import:

```
POST /reader/subscriptions/import HTTP/1.1
Content-Type: multipart/form-data; boundary=----------------------------7d9393199e0a04

------------------------------7d9393199e0a04
Content-Disposition: form-data; name="T"

N7iWAR8BAAA.9POkV_f2QOGEFGeW47PTzA.xRD0ZrnXEs9DgfXRR4QGvg
------------------------------7d9393199e0a04
Content-Disposition: form-data; name="action"

opml-upload
------------------------------7d9393199e0a04
Content-Disposition: form-data; name="opml-file"; filename="C:\Users\nbradbury\Desktop\test.opml"
Content-Type: application/octet-stream

<opml version="1.1">
  <head>
    <title>FeedDemon Subscriptions</title>
    <dateModified>Fri, 23 Jan 2009 03:18:55 GMT</dateModified>
  </head>
  <body>
    <outline text="Video">
      <outline text="Hulu - Popular videos this month" title="Hulu - Popular videos this month" type="rss"
xmlUrl="http://rss.hulu.com/HuluPopularVideosThisMonth" htmlUrl="http://www.hulu.com/feed" description="Hulu - Popular videos
this month"/>
      <outline text="Videos uploaded by MontyPython" title="Videos uploaded by MontyPython" type="rss"
xmlUrl="http://gdata.youtube.com/feeds/base/users/MontyPython/uploads?alt=rss&amp;amp;v=2&amp;amp;client=ytapi-youtube-profile"
htmlUrl="http://www.youtube.com/profile_videos?user=MontyPython" category="http://gdata.youtube.com/schemas/2007#video"/>
    </outline>
  </body>
</opml>

------------------------------7d9393199e0a04--
```

ReplaceSubscriptions (no equivalent)

Don't see a way to do this without deleting all feeds and folders, and then adding the new feeds and folders.

MarkFeedRead (verified)

```
POST mark-all-as-read s=feed/[feedUrl]&ts=[timestamp]
```

MarkFolderRead

```
POST mark-all-as-read t=[folderName]&ts=[timestamp]
```

Mark a single post as read (verified)

`POST edit-tag ac=edit-tags&i=[EntryID]&a=user/-/state/com.google/read&async=true`

Note: GReader includes the feed URL as `&s=feed/[FeedUrl]` but this doesn't appear to be necessary.

Mark a single post as unread (questionable)

`POST edit-tag ac=edit-tags&i=[EntryID]&r=user/-/state/com.google/read&async=true`

Note: This doesn't always work, but it's the same thing that RSS Bandit does (which likewise doesn't always work).

Mark a list of posts as read (verified)

`POST edit-tag ac=edit-tags&a=user/-/state/com.google/read&async=true&i=[EntryID1]&i=[EntryID2]...`

GetSubscriptionCounts (verified)

`GET unread-count?output=xml`

The response is an object list of unread counts for all feeds and folders (labels), and for the entire reading list (total unread count). Ex:

```
<object>
    <number name="max">1000</number>
    <list name="unreadcounts">
        <object>
            <string name="id">user/-/state/com.google/reading-list</string>
            <number name="count">8</number>
            <number name="newestItemTimestampUsec">1232641608774062</number>
        </object>
        <object>
            <string name="id">feed/http://rss.cnn.com/rss/cnn_topstories.rss</string>
            <number name="count">4</number>
            <number name="newestItemTimestampUsec">1232641203555343</number>
        </object>
        <object>
            <string name="id">user/-/label/FolderName</string>
            <number name="count">8</number>
            <number name="newestItemTimestampUsec">1232641608774062</number>
        </object>
        <object>
            <string name="id">feed/http://www.engadget.com/rss.xml</string>
            <number name="count">4</number>
            <number name="newestItemTimestampUsec">1232641608774062</number>
        </object>
    </list>
</object>
```

Each feed is timestamped with the datetime of its most recent entry (time since the Unix epoch, UTC). When no unread entries exist, it returns this:

```
<object>
    <number name="max">1000</number>
    <list name="unreadcounts"/>
</object>
```

GetUpdates (verified)

No direct way to find out which feeds have updated, but it's possible to use the method described in **GetSubscriptionCounts** to get the timestamp of the most recent entry in each feed, and then compare that timestamp to the most recent entry in the client's feed cache. Note that this would only tell which feeds have new entries – it wouldn't let us know when an entry has changed.

UpdatePostMetadata (no equivalent)

GReader offers no single call which can be used to upload states changed in the client and download states changed in the reader.

ClipItems / ClipPosts / UnClipPosts

See **Sharing** section below

AddTags / RemoveTags / RemoveAllTags / Rename (rename tag) / ReplaceTags

See **Tagging** section below

Feed Parsing

Feeds are retrieved from GReader using this format:

```
GET http://www.google.com/reader/atom/feed/[feedUrl]
```

For example, my blog feed is retrieved using:

```
GET http://www.google.com/reader/atom/feed/http://feeds.feedburner.com/NickBradbury
```

By default, GReader retrieves the most recent 20 entries. This can be changed by adding `&n=[#items]`. To retrieve only unread entries, use the exclude target (xt) parameter to exclude the "read" label, like this:

```
GET http://www.google.com/reader/atom/feed/[FeedUrl]&xt=user/-/state/com.google/read
```

An interesting feature is the ability to translate a feed to the user's language (as defined in their account settings) by adding `&trans=true` to the URL.

Here's an example entry from my feed as seen by GReader:

```
<entry gr:is-read-state-locked="true" gr:crawl-timestamp-msec="1231022805003">
  <id gr:original-id="http://nick.typepad.com/blog/2009/01/the-great-un-fr.html">tag:google.com,2005:reader/item/30fd6c8e007ebcbf</id>
  <category term="user/[userid]/state/com.google/read" scheme="http://www.google.com/reader/" label="read" />
  <category term="user/[userid]/state/com.google/starred" scheme="http://www.google.com/reader/" label="starred"/>
  <category term="user/[userid]/label/[folderName]" scheme="http://www.google.com/reader/" label="[folderName]"/>
  <category term="[category from source feed]" />
  <title type="html">The Great Un-Friending of 2009</title>
  <published>2009-01-03T18:37:04Z</published>
  <updated>2009-01-03T18:37:04Z</updated>
  <link rel="alternate" href="http://nick.typepad.com/blog/2009/01/the-great-un-fr.html" type="text/html" />
  <summary xml:base="http://nick.typepad.com/" type="html">[Article summary]</summary>
  <author>
    <name>Nick Bradbury</name>
  </author>
  <source gr:stream-id="feed/http://feeds.feedburner.com/NickBradbury">
    <id>tag:google.com,2005:reader/feed/http://feeds.feedburner.com/NickBradbury</id>
    <title type="html">Nick Bradbury</title>
    <link rel="alternate" href="http://nick.typepad.com/" type="text/html" />
  </source>
</entry>
```

The `gr:crawl-timestamp-msec` attribute of the `entry` element is the Unix timestamp denoting when the article was received by GReader. An entry whose read state has been locked ("keep as unread") will have a `gr:is-read-state-locked="true"` attribute.

The ID of the entry is always set to the ID in GReader's platform, replacing the original GUID/ID from the source feed. However, the original GUID/ID may still be obtained from the ID node:

```
<id gr:original-id="http://nick.typepad.com/blog/2009/01/">tag:google.com,2005:reader/item/30fd6c8e007ebcbf</id>
```

Categories are used to reflect not only the categories assigned to that entry from the actual feed, but also the read/flagged state in GReader. If an entry has been read, it will have this category:

```
<category term="user/[userid]/state/com.google/read" scheme="http://www.google.com/reader/" label="read" />
```

If an entry has been starred, it will have this category:

```
<category term="user/[userid]/state/com.google/starred" scheme="http://www.google.com/reader/" label="starred"/>
```

Each entry also has a category for the folder the feed is in:

```
<category term="user/[userid]/label/[folderName]" scheme="http://www.google.com/reader/" label="[folderName]"/>
```

Continuation

An important part of retrieving GReader feeds is **continuation**, which provides a way to page through a feed's entries. For example, if you request the latest 20 entries in a feed, the root <feed> element may have a `gr:continuation` child element, ex:

```
<feed xmlns:gr="http://www.google.com/schemas/reader/atom/" xmlns="http://www.w3.org/2005/Atom">
  <generator uri="http://www.google.com/reader">Google Reader</generator>
  <id>tag:google.com,2005:reader/feed/http://feeds.feedburner.com/NickBradburyClippings</id>
  <title>Nick Bradbury&amp;#39;s Shared Items on NewsGator Online</title>
  <updated>2009-01-15T15:17:16Z</updated>
  <gr:continuation>CP__hN2fq5gC</gr:continuation>
```

Simply append the `gr:continuation` value to a subsequent request to get the next 20 entries in the feed, like this:

```
GET http://www.google.com/reader/atom/feed/http://feeds.feedburner.com/NickBradbury&c=CP__hN2fq5gC
```

This enables accessing a feed's history, which may be useful immediately after subscribing to a feed or when the user first chooses to enable GReader synching.

Sharing

To retrieve a user's shared items as an Atom feed, use:

```
GET http://www.google.com/reader/atom/user/-/state/com.google/broadcast&n=[count]
```

Sharing/unsharing an item uses the `edit-tag` command with the tag `broadcast`. For example, to share an item:

```
POST edit-tag a=user/-/state/com.google/broadcast&i=[entryId]&s=[streamId]&async=true
```

According to Mihai Parparita, the `streamId` identifies the feed the shared item came from, and it's important:

> *"If it's absent, we can look it up, but that will both slow down the request and also increase the chance of failure. I believe you should always have this information on-hand, it's in <source> element in our Atom feeds, as a gr:stream-id namespaced attribute."*

To unshare an item:

```
POST edit-tag r=user/-/state/com.google/broadcast&i=[entryId]&async=true
```

It's important to note that the entry ID **must** be the ID of the entry in GReader, so entries from non-GReader feeds need to be shared the same way as arbitrary URLs, like this:

```
POST item/edit
share=true&title=[title]&url=[link]&snippet=[excerpt]&srcTitle=[sourceTitle]&srcUrl=[sourceUrl]
```

The next time you retrieve the user's shared items, you'll see that GReader has assigned an entry ID to this item. The `sourceTitle` and `sourceUrl` parameters are the name and link of the page/ feed the item is from. An `annotation` parameter may also be added.

Entries in shared items that have been annotated ("Share with Note") will contain something like this:

```
<gr:annotation>
  <content type="html">Text of annotation.</content>
  <author gr:user-id="[userid]" gr:profile-id="[profileid]">
    <name>nbradbury</name>
  </author>
</gr:annotation>
```

Note that the annotation text will also be included in the entry's content.

Tagging

To get a list of all tags:

```
POST tag/list?output=xml
```

Note: Because GReader treats folders as tags, **this list will include folder names**. It's up to the client to parse out tags that are actually the names of folders.

To add a tag to an entry:

```
POST edit-tag a=user/-/label/[tag]&i=[entryId]
```

To add multiple tags to an entry:

```
POST edit-tag a=user/-/label/[tag1]&a=user/-/label/[tag2]...&i=[entryId]
```

To remove a tag from an entry:

```
POST edit-tag r=user/-/label/[tag]&i=[entryId]
```

There doesn't appear to be a direct way to remove all tags from an entry, so this requires first retrieving all tags for an entry and then deleting them.

To remove a tag from all entries, use:

```
POST disable-tag s=user/-/label/[tag]&t=[tag]
```

To rename a tag, you have to first delete the old tag, then apply the new tag to all items that had the old one.

Sharing Tags

Google Reader enables sharing tags, which means that a "public" HTML page and RSS feed can be made available for each tag.

To change a tag's public setting, use:

```
POST tag/edit?client=settings s=user/-/label/[tag]&t=[tag]&pub=[true/false]
```

The HTML page for a shared tag is:

```
http://www.google.com/reader/shared/user/[userid]/label/[tag]
```

And the RSS feed for a shared tag is:

```
http://www.google.com/reader/public/atom/user/[userid]/label/[tag]
```

Note: When a feed is unsubscribed, GReader retains tagged posts in that feed.

Miscellaneous

- Custom states can be supported by using a namespace other then `com.google` (ex: `user/-/state/com.newsgator/clipped`)
- Tags/folders cannot contain these characters: < > ? & / \ ^
- `GET http://www.google.com/reader/atom/user/-/state/com.google/starred` returns all starred (flagged) entries
- ~~`GET http://www.google.com/reader/user-info` returns a user's ID, profile ID and email address (JSON only)~~
- As of 07-Sept-2009, the above call redirects to `http://www.google.com/reader/api/0/user-info`
- **`GET friend/list` returns a list of the user's friends**
- `GET preference/list` returns the user's GReader preferences
- `GET stream/contents/user/-/state/com.google/reading-list` returns all unread items. FeedDemon uses this whenever possible because it's **much** faster than retrieving unread items on a per-feed basis.
- `http://www.google.com/reader/directory/search?q=[searchterm]` searches the GReader taxonomy (HTML results)
- `http://www.google.com/reader/view/feed/[feedUrl]` previews a feed
- `http://www.google.com/reader/subscriptions/export` is the user's OPML
- `GET search/items/ids?q=[searchterm]` returns an object list of IDs containing that term, and `POST stream/items/contents?output=atom i=[id1]&i=[id2]…` returns the contents of those items (note that this returns posts in JSON format when `output=atom` isn't specified)

Google Gears Notes

To retrieves IDs of items with specific state(s), Gears uses:

```
GET http://www.google.com/reader/api/0/stream/items/ids?n=[count]&offsync=true&s=[state1]&s=[state2]&output=xml
```

It then retrieves the contents for those items using:

```
POST http://www.google.com/reader/api/0/stream/items/contents?offsync=true i=[id1]&i=[id2]...
```

For example, before going offline, Gears retrieves the IDs of the most recent 2000 items in the user's reading list by using:

```
http://www.google.com/reader/api/0/stream/items/ids?n=2000&offsync=true&s=user%2f16940339600606336011%2fstate%2fcom.google%2freading-list&output=xml
```

Gears retrieves all unread items by using:

```
http://www.google.com/reader/api/0/stream/contents/user/16940339600606336011/state/com.google/reading-list?ot=1231506000&r=n&xt=user%2F16940339600606336011%2Fstate%2Fcom.google%2Fread&n=[count]
```

IMPORTANT: The response from `stream/contents/` is always returned as JSON!

Reference

These sites contain some details about the GReader API, but much of it is outdated. Use these only as guidance, not as gospel. For more reliable information, use an HTTP sniffer to sniff the requests from GReader itself.

- http://code.google.com/p/pyrfeed/wiki/GoogleReaderAPI (fairly accurate)
- http://www.niallkennedy.com/blog/2005/12/google-reader-api.html (outdated)
- http://markmail.org/search/?q=%22google+reader%22+api
- http://googlesystem.blogspot.com/2008/03/explore-your-interactions-with-google.html
- http://blog.gpowered.net/2007/08/google-reader-api-functions.html
- http://www.25hoursaday.com/weblog/2007/12/31/CommandLineClientForGoogleReaderInIronPython.aspx
- http://groups.google.com/group/google-reader-help/search?group=google-reader-help&q=API&qt_g=Search+this+group
- http://code.google.com/p/readair/source/browse/tags/0.3/assets/js/gra/GRA.atomentry.js
- http://cpansearch.perl.org/src/GRAY/WebService-Google-Reader-0.07/lib/WebService/Google/Reader.pm

Sync Outline

OUTDATED

- Call subscription/list to get latest GR subscriptions, then compare to local and check for changes.
- Call reader/atom/user/-/state/com.google/reading-list?ot=<timestamp>&xt=user/-/state/com.google/read to get unread posts since the previous sync, then adding them to FD if they don't already exist.

- Call stream/items/ids?n=<count>&s=user/-/state/com.google/read to get the IDs of the most recently read posts, then making sure they're marked read in FD. If any of those IDs don't exist in FD, I then download them and add them to FD as read posts.
- This usually takes care of matching read/unread posts between FD and GR, but there are always edge cases, so I then call unread-count?output=xml to get GR's unread counts, and if any feed shows a different unread count/timestamp in GR than it does in FD, I then update that feed individually.
- Get the IDs of the most recent starred posts in GR, then make sure they're starred in FD.
- Call tag/list?output=xml to get a list of tags from GR, then making separate calls to stream/items/ids?n=<count>&s=user/-/label/<tag> to get the IDs of items with each tag and matching to local tagged items.