Google Data APIs Protocol Reference

This document describes the protocol used by the Google data APIs ("GData"), including information about what a query looks like, what results look like, and so on.

For more information about the Google data APIs, see the Google Data APIs Overview document and the Protocol Basics document.

Contents

Audience

This document is intended for anyone wanting to understand the details of the XML format and protocol used by the Google data APIs.

If you just want to write code that uses the GData client APIs, then you don't need to know these details; instead, you can use the language-specific client libraries.

But if you want to understand the protocol, read this document. For example, you may want to read this document to help you with any of the following tasks:

- evaluating the GData architecture
- coding using the protocol without using the provided GData libraries
- writing a client library in a new language

This document assumes that you understand the basics of XML, namespaces, syndicated feeds, and the `GET`, `POST`, `PUT`, and `DELETE` requests in HTTP, as well as HTTP's concept of a "resource." For more information about those things, see the Additional resources section of this document.

This document doesn't rely on any particular programming language; you can send and receive GData messages using any programming language that lets you issue HTTP requests and parse XML-based responses.

Protocol details

This section describes the GData document format and query syntax.

Document format

GData, Atom, and RSS 2.0 all share the same basic data model: a container that holds both some global data and any number of entries. For each protocol, the format is defined by a base schema, but it can be extended using foreign namespaces.

GData can use either the Atom syndication format (for both reads and writes) or the RSS format (for reads only).

Atom is GData's default format. To request a response in RSS format, use the `/alt=rss/` parameter; for more information, see Query requests.

When you request data in RSS format, GData supplies a feed (or other representation of the resource) in RSS format. If there's no equivalent RSS property for a given GData property, GData uses the Atom property, labeling it with an appropriate namespace to indicate that it's an extension to RSS.

Note: Most GData feeds in Atom format use the Atom namespace as the default namespace by specifying an `xmlns` attribute on the feed element; see the examples section for examples of how to do that. Thus, the examples in this document don't explicitly specify `atom:` for elements in an Atom-format feed.

The following tables show the Atom and RSS representations of the elements of the schema. All data not mentioned in these tables is treated as plain XML and shows up the same in both representations. Unless indicated otherwise, the XML elements in a given column are in the namespace corresponding to that column. This summary uses standard XPath notation: in particular, slashes show the element hierarchy, and an @ sign indicates an attribute of an element.

In each of the following tables, the highlighted items are required.

The following table shows the elements of a GData feed:

Feed Schema Item	**Atom Representation**	**RSS Representation**
Feed Title	`/feed/title`	`/rss/channel/title`
Feed ID	`/feed/id`	`/rss/channel/atom:id`
Feed HTML Link	`/feed/link[@rel="alternate"]\ [@type="text/html"]/@href`	`/rss/channel/link`
Feed Description	`/feed/subtitle`	`/rss/channel/description`
Feed Language	`/feed/@xml:lang`	`/rss/channel/language`
Feed Copyright	`/feed/rights`	`/rss/channel/copyright`

The following table shows the elements of a GData search-results feed. Note that GData exposes some of the OpenSearch 1.1 Response elements in its search-results feeds.

Search Result Feed Schema Item	Atom Representation	RSS/OpenSearch Representation
Number of Search Results	`/feed/openSearch:totalResults`	`/rss/channel/openSearch:totalResults`
Search Result Start Index	`/feed/openSearch:startIndex`	`/rss/channel/openSearch:startIndex`
Number of Search Results Per Page	`/feed/openSearch:itemsPerPage`	`/rss/channel/openSearch:itemsPerPage`

The following table shows the elements of a GData entry:

Entry Schema Item	Atom Representation	RSS Representation
Entry ID	`/feed/entry/id`	`/rss/channel/item/guid`
Entry Version ID	Optionally embedded in EditURI (see the Optimistic concurrency section of this document).	—
Entry Title	`/feed/entry/title`	`/rss/channel/item/title`

Entry Link	`/feed/entry/link`	`/rss/channel/item/link`
		`/rss/channel/item/enclosure`
		`/rss/channel/item/comments`

Entry Summary	`/feed/entry/summary` (Required in certain cases; see Atom specification.)	`/rss/channel/item/atom:summary`
Entry Content	`/feed/entry/content` (If no content element, then entry must contain at least one `<link rel="alternate">` element.)	`/rss/channel/item/description`
Entry Author	`/feed/entry/author/name` `/feed/entry/author/email`	`/rss/channel/item/author`

Queries

This section describes how to use the query system.

Query model design tenets

The query model is intentionally very simple. The basic tenets are:

- Queries are expressed as HTTP URIs, rather than as HTTP headers or as part of the payload. One benefit of this approach is that you can link to a query.
- Predicates are scoped to a single item. Thus, there's no way to send a correlation query such as "find all emails from people who sent me at least 10 emails today."
- The set of properties that queries can predicate on is very limited; most queries are simply full text search queries.
- Result ordering is up to the implementation.
- The protocol is naturally extensible. If you want to expose additional predicates or sorting in your service, you can do so easily through the introduction of new parameters.

Query requests

A client queries a GData service by issuing an HTTP `GET` request. The query URI consists of the resource's URI (called *FeedURI* in Atom) followed by query parameters. Most query parameters are represented as traditional `?name=value[&...]` URL parameters. Category parameters are handled differently; see below.

For example, if the FeedURI is `http://www.example.com/feeds/jo`, then you might send a query with the following URI:

```
http://www.example.com/feeds/jo?q=Darcy&updated-min=2005-04-19T15:30:00
```

GData services support HTTP Conditional `GET`. They set the Last-Modified response header based upon the value of the `<atom:updated>` element in the returned feed or entry. A client can send this value back as the value of the If-Modified-Since request header to avoid retrieving the content again if it hasn't changed. If the content hasn't changed since the If-Modified-Since time, then the GData service returns a 304 (Not Modified) HTTP response.

A GData service must support category queries and `alt` queries; support for other parameters is optional. Passing a standard parameter not understood by a given service results in a `403 Forbidden` response. Passing an unsupported nonstandard parameter results in a `400 Bad Request` response. For information on other status codes, see the HTTP status codes section of this document.

The standard query parameters are summarized in the following table. All parameter values need to be URL encoded.

Parameter	Meaning	Notes
`q`	Full-text query string	• When creating a query, list search terms separated by spaces, in the form `q=term1 term2 term3`. (As with all of the query parameter values, the spaces must be URL encoded.) The GData service returns all entries that match all of the search terms (like using `AND` between terms). Like Google's web search, a GData service searches on complete words (and related words with the same stem), not substrings. • To search for an exact phrase, enclose the phrase in quotation

About category queries

We decided to specify a slightly unusual format for category queries. Instead of a query like the following:

```
http://example.com/jo?category=Fritz&category=2006
```

we use:

```
http://example.com/jo/-/Fritz/2006
```

This approach identifies a resource without using query parameters, and it produces cleaner URIs. We chose this approach for categories because we think that category queries will be the most common queries.

The drawback to this approach is that we require you to use `/-/` in category queries, so that GData services can distinguish category queries from other resource URIs, such as `http://example.com/jo/MyPost/comments`.

Query responses

Queries return an Atom feed, an Atom entry, or an RSS feed, depending on the request parameters.

Query results contain the following OpenSearch elements directly under the `<feed>` element or the `<channel>` element (depending on whether results are Atom or RSS):

`openSearch:totalResults`

The total number of search results for the query (not necessarily all present in the results feed).

`openSearch:startIndex`

The 1-based index of the first result.

`openSearch:itemsPerPage`

The maximum number of items that appear on one page. This allows clients to generate direct links to any set of subsequent pages. However, for a possible pitfall in using this number, see the note regarding `start-index` in the table in the Query requests section.

The Atom response feed and entries may also include any of the following Atom and GData elements (as well as others listed in the Atom specification):

`<link rel="http://schemas.google.com/g/2005#feed" type="application/atom+xml" href="..."/>`

Specifies the URI where the complete Atom feed can be retrieved.

`<link rel="http://schemas.google.com/g/2005#post" type="application/atom+xml" href="..."/>`

Specifies the Atom feed's PostURI (where new entries can be posted).

`<link rel="self" type="..." href="..."/>`

Contains the URI of this resource. The value of the `type` attribute depends on the requested format. If no data changes in the interim, sending another GET to this URI returns the same response.

`<link rel="previous" type="application/atom+xml" href="..."/>`

Specifies the URI of the previous chunk of this query result set, if it is chunked.

`<link rel="next" type="application/atom+xml" href="..."/>`

Specifies the URI of the next chunk of this query result set, if it is chunked.

<link rel="edit" type="application/atom+xml" href="..."/>

Specifies the Atom entry's EditURI (where you send an updated entry).

Here's a sample response body, in response to a search query:

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns:atom="http://www.w3.org/2005/Atom"
        xmlns:openSearch="http://a9.com/-/spec/opensearchrss/1.0/">
  <id>http://www.example.com/feed/1234.1/posts/full</id>
  <updated>2005-09-16T00:42:06Z</updated>
  <title type="text">Books and Romance with Jo and Liz</title>
  <link rel="alternate" type="text/html" href="http://www.example.net/"/>
  <link rel="http://schemas.google.com/g/2005#feed"
    type="application/atom+xml"
    href="http://www.example.com/feed/1234.1/posts/full"/>
  <link rel="http://schemas.google.com/g/2005#post"
    type="application/atom+xml"
    href="http://www.example.com/feed/1234.1/posts/full"/>
  <link rel="self" type="application/atom+xml"
    href="http://www.example.com/feed/1234.1/posts/full"/>
  <author>
    <name>Elizabeth Bennet</name>
    <email>liz@gmail.com</email>
  </author>
  <generator version="1.0"
    uri="http://www.example.com">Example Generator Engine</generator>
  <openSearch:totalResults>2</openSearch:totalResults>
  <openSearch:startIndex>0</openSearch:startIndex>
  <entry>
    <id>http://www.example.com/feed/1234.1/posts/full/4521614025009481151</id>
    <published>2005-01-09T08:00:00Z</published>
    <updated>2005-01-09T08:00:00Z</updated>
    <category scheme="http://www.example.com/type" term="blog.post"/>
    <title type="text">This is the title of entry 1009</title>
    <content type="xhtml">
      <div
        xmlns="http://www.w3.org/1999/xhtml">This is the entry body of entry
1009</div>
    </content>
    <link rel="alternate" type="text/html"
      href="http://www.example.com/posturl"/>
    <link rel="edit" type="application/atom+xml"
      href="http://www.example.com/feed/1234.1/posts/full/4521614025009481151"/>
    <author>
      <name>Elizabeth Bennet</name>
      <email>liz@gmail.com</email>
    </author>
  </entry>
  <entry>
    <id>http://www.example.com/feed/1234.1/posts/full/3067545004648931569</id>
    <published>2005-01-07T08:00:00Z</published>
    <updated>2005-01-07T08:02:00Z</updated>
    <category scheme="http://www.example.com/type" term="blog.post"/>
```

```
    <title type="text">This is the title of entry 1007</title>
    <content type="xhtml">
      <div
        xmlns="http://www.w3.org/1999/xhtml">This is the entry body of entry
1007</div>
    </content>
    <link rel="alternate" type="text/html"
      href="http://www.example.com/posturl"/>
    <link rel="edit" type="application/atom+xml"
      href="http://www.example.com/feed/1234.1/posts/full/3067545004648931569"/>
    <author>
      <name>Elizabeth Bennet</name>
      <email>liz@gmail.com</email>
    </author>
  </entry>
</feed>
```

If the requested feed is in the Atom format, if no query parameters are specified, and if the result doesn't contain all the entries, the following element is inserted into the top-level feed: `<link rel="next" type="application/atom+xml" href="..."/>`. It points to a feed containing the next set of entries. Subsequent sets contain a corresponding `<link rel="previous" type="application/atom+xml" href="..."/>` element. By following all the *next* links, a client can retrieve all entries from a feed.

HTTP status codes

The following table describes what various HTTP status codes mean in the context of GData.

Code	Explanation
200 OK	No error.
201 CREATED	Creation of a resource was successful.
304 NOT MODIFIED	The resource hasn't changed since the time specified in the request's If-Modified-Since header.
400 BAD REQUEST	Invalid request URI or header, or unsupported nonstandard parameter.
401 UNAUTHORIZED	Authorization required.
403 FORBIDDEN	Unsupported standard parameter, or authentication or authorization failed.
404 NOT FOUND	Resource (such as a feed or entry) not found.
409 CONFLICT	Specified version number doesn't match resource's latest version number.
500 INTERNAL SERVER ERROR	Internal error. This is the default code that is used for all unrecognized errors.

Optimistic concurrency (versioning)

Sometimes it is important to ensure that multiple clients don't inadvertently overwrite one another's changes. This is most easily accomplished by ensuring that the current version of an entry that a client is modifying is the same as the version that the client is basing its modifications on. If a second client makes an update before the first client does, then the first client's update is denied, because the first client is no longer basing its modifications on the latest version.

In GData feeds that support versioning, we achieve these semantics by appending a version ID to each entry's EditURI. Note that only the EditURI is affected, not the entry ID. In this scheme, each update changes the entry's EditURI, thus guaranteeing that subsequent updates based on the original version fail. Deletes, of course, are identical to updates with respect to this feature; if you send a delete with an old version number, the delete fails.

Not all GData feeds support optimistic concurrency. In a feed that does support it, if the server detects a version conflict on PUT or DELETE, the server responds with `409 Conflict`. The body of the response contains the current state of the entry (an Atom entry document). The client is advised to resolve the conflict and resubmit the request, using the EditURI from the 409 response.

Motivation and design notes

This approach to optimistic concurrency allows us to implement the semantics we want without requiring new markup for version IDs, which makes GData's responses compatible with non-GData Atom endpoints.

Instead of specifying version IDs, we could have chosen to look at the update timestamp on each entry (`/atom:entry/atom:updated`). However, there are two problems with using the update timestamp:

- It only works for updates and not deletions.
- It forces applications to use date/time values as version IDs, which would make it harder to retrofit GData on top of many existing data stores.

Authentication

When a client tries to access a service, it may need to provide the user's credentials to the service, to demonstrate that the user has the authority to perform the action in question.

The approach that a client should use for authentication depends on the type of client:

- A desktop application should use a Google-specific authentication system called Account Authentication for Installed Applications (also known as "ClientLogin"). (Web-based clients should not use this system.)
- A web-based client, such as a third-party front end to a GData service, should use a Google-specific authentication system called Account Authentication Proxy for Web-Based Applications (also known as "AuthSub").

In the ClientLogin system, the desktop client asks the user for their credentials, and then sends those credentials to the Google authentication system.

If authentication succeeds, then the authentication system returns a token that the client subsequently uses (in an HTTP Authorization header) when it sends GData requests.

If authentication fails, then the server returns a 403 Forbidden status code, along with a WWW-Authenticate header containing a challenge applicable to the authentication.

The AuthSub system works similarly, except that instead of asking the user for their credentials, it connects the user to a Google service that requests credentials. The service then returns a token that the web application can use; the advantage of this approach is that Google (rather than the web front end) securely handles and stores the user's credentials.

For details about these authentication systems, see the GData authentication overview or the Google Account Authentication documentation.

Session state

Many business logic implementations require session stickiness—keeping track of the state of a user's session.

Google tracks session state in two ways: using cookies, and using a token that can be sent as a query parameter. Both methods achieve the same effect. We recommend that clients support one of these session-state tracking methods (either one is sufficient). If a client doesn't support either of these methods, then that client will still work with GData services, but performance may suffer compared to clients that do support these methods. Specifically, if a client doesn't support these methods, then every request results in a redirect, and therefore every request (and any associated data) is sent to the server twice, which affects the performance of both the client and the server.

The Google client libraries handle session state for you, so if you use our libraries, you don't have to do anything to get session state support.

Additional resources

You may find the following third-party documents useful:

- Comparison of Atom and RSS from intertwingly
- Overview of Atom from IBM
- Dublin Core extensions to RSS
- HTTP 1.1 method definitions; specification for `GET`, `POST`, `PUT`, and `DELETE`
- HTTP 1.1 status code definitions
- How to Create a REST Protocol
- Building Web Services the REST Way
- A Technical Introduction to XML
- XML Namespaces by Example