Send batch requests

This document shows how to batch API calls together to reduce the number of HTTP connections your client has to make.

This document is specifically about making a batch request by sending an HTTP request. If, instead, you're using a Google client library to make a batch request, see the client library's documentation (https://developers.google.com/api-client-library/).

Overview

Each HTTP connection your client makes results in a certain amount of overhead. The Google Calendar API supports batching, to allow your client to put several API calls into a single HTTP request.

Examples of situations when you might want to use batching:

- You've just started using the API and you have a lot of data to upload.
- A user made changes to data while your application was offline (disconnected from the Internet), so your application needs to synchronize its local data with the server by sending a lot of updates and deletes.

In each case, instead of sending each call separately, you can group them together into a single HTTP request. All the inner requests must go to the same Google API.

You're limited to 1000 calls in a single batch request. If you must make more calls than that, use multiple batch requests.

Note: The batch system for the Google Calendar API uses the same syntax as the OData batch processing (http://www.odata.org/documentation/odata-version-3-0/batch-processing/) system, but the semantics differ.

Batch details

A batch request consists of multiple API calls combined into one HTTP request, which can be sent to the `batchPath` specified in the API discovery document (https://developers.google.com/discovery/v1/reference/apis). The default path is **`/batch/`*`api_name`*`/`** ***`api_version`***. This section describes the batch syntax in detail; later, there's an example (#example).

Note: A set of ***n*** requests batched together counts toward your usage limit as ***n*** requests, not as one request. The batch request is separated into a set of requests before processing.

Format of a batch request

A batch request is a single standard HTTP request containing multiple Google Calendar API calls, using the `multipart/mixed` content type. Within that main HTTP request, each of the parts contains a nested HTTP request.

Each part begins with its own `Content-Type: application/http` HTTP header. It can also have an optional `Content-ID` header. However, the part headers are just there to mark the beginning of the part; they're separate from the nested request. After the server unwraps the batch request into separate requests, the part headers are ignored.

The body of each part is a complete HTTP request, with its own verb, URL, headers, and body. The HTTP request must only contain the path portion of the URL; full URLs are not allowed in batch requests.

The HTTP headers for the outer batch request, except for the `Content-` headers such as `Content-Type`, apply to every request in the batch. If you specify a given HTTP header in both the outer request and an individual call, then the individual call header's value overrides the outer batch request header's value. The headers for an individual call apply only to that call.

For example, if you provide an Authorization header for a specific call, then that header applies only to that call. If you provide an Authorization header for the outer request, then that header applies to all of the individual calls unless they override it with Authorization headers of their own.

When the server receives the batched request, it applies the outer request's query parameters and headers (as appropriate) to each part, and then treats each part as if it were a separate HTTP request.

Response to a batch request

The server's response is a single standard HTTP response with a **`multipart/mixed`** content type; each part is the response to one of the requests in the batched request, in the same order as the requests.

Like the parts in the request, each response part contains a complete HTTP response, including a status code, headers, and body. And like the parts in the request, each response part is preceded by a **`Content-Type`** header that marks the beginning of the part.

If a given part of the request had a **`Content-ID`** header, then the corresponding part of the response has a matching **`Content-ID`** header, with the original value preceded by the string **`response-`**, as shown in the following example.

Note: The server might perform your calls in any order. Don't count on their being executed in the order in which you specified them. If you want to ensure that two calls occur in a given order, you can't send them in a single request; instead, send the first one by itself, then wait for the response to the first one before sending the second one.

Example

The following example shows the use of batching with a generic (fictional) demo API called the Farm API. However, the same concepts apply to the Google Calendar API.

Example batch request

```
POST /batch/farm/v1 HTTP/1.1
Authorization: Bearer your_auth_token
Host: www.googleapis.com
Content-Type: multipart/mixed; boundary=batch_foobarbaz
Content-Length: total_content_length

--batch_foobarbaz
Content-Type: application/http
Content-ID: <item1:12930812@barnyard.example.com>
```

```
GET /farm/v1/animals/pony

--batch_foobarbaz
Content-Type: application/http
Content-ID: <item2:12930812@barnyard.example.com>

PUT /farm/v1/animals/sheep
Content-Type: application/json
Content-Length: part_content_length
If-Match: "etag/sheep"

{
  "animalName": "sheep",
  "animalAge": "5"
  "peltColor": "green",
}

--batch_foobarbaz
Content-Type: application/http
Content-ID: <item3:12930812@barnyard.example.com>

GET /farm/v1/animals
If-None-Match: "etag/animals"

--batch_foobarbaz--
```

Example batch response

This is the response to the example request in the previous section.

```
HTTP/1.1 200
Content-Length: response_total_content_length
Content-Type: multipart/mixed; boundary=batch_foobarbaz

--batch_foobarbaz
Content-Type: application/http
Content-ID: <response-item1:12930812@barnyard.example.com>

HTTP/1.1 200 OK
Content-Type application/json
Content-Length: response_part_1_content_length
ETag: "etag/pony"
```

```
{
  "kind": "farm#animal",
  "etag": "etag/pony",
  "selfLink": "/farm/v1/animals/pony",
  "animalName": "pony",
  "animalAge": 34,
  "peltColor": "white"
}

--batch_foobarbaz
Content-Type: application/http
Content-ID: <response-item2:12930812@barnyard.example.com>

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: response_part_2_content_length
ETag: "etag/sheep"

{
  "kind": "farm#animal",
  "etag": "etag/sheep",
  "selfLink": "/farm/v1/animals/sheep",
  "animalName": "sheep",
  "animalAge": 5,
  "peltColor": "green"
}

--batch_foobarbaz
Content-Type: application/http
Content-ID: <response-item3:12930812@barnyard.example.com>

HTTP/1.1 304 Not Modified
ETag: "etag/animals"

--batch_foobarbaz--
```