REST API v3

Overview

This describes the resources that make up the Blog REST API v3. If you have any problems or requests, please contact the writer's support.

Current version

By default, all requests to `https://api.blog.com` receive the v3 version of the REST API. We encourage you to explicitly request this version via the Accept header.

```
Accept: application/vnd.blog.v3+json
```

Schema

All API access is over HTTPS, and accessed from https://api.blog.com. All data is sent and received as JSON.

```
curl -i https://api.blog.com/users/octocat/orgs
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 12 Oct 2012 23:33:14 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Status: 200 OK
ETag: "a00049ba79152d03380c34652f2cb612"
X-blog-Media-Type: blog.v3
X-RateLimit-Limit: 5000
X-RateLimit-Remaining: 4987
X-RateLimit-Reset: 1350085394
Content-Length: 5
Cache-Control: max-age=0, private, must-revalidate
X-Content-Type-Options: nosniff
```

Blank fields are included as null instead of being omitted.

All timestamps return in ISO 8601 format:

```
YYYY-MM-DDTHH:MM:SSZ
```

Summary representations

When you fetch a list of resources, the response includes a subset of the attributes for that resource. This is the "summary" representation of the resource. (Some attributes are computationally expensive for the API to provide. For performance reasons, the summary representation excludes those attributes. To obtain those attributes, fetch the "detailed" representation.)

Example: When you get a list of repositories, you get the summary representation of each repository. Here, we fetch the list of repositories owned by the octokit organization:

```
GET /orgs/octokit/repos
```

Detailed representations

When you fetch an individual resource, the response typically includes all attributes for that resource. This is the "detailed" representation of the resource. (Note that authorization sometimes influences the amount of detail included in the representation.)

Example: When you get an individual repository, you get the detailed representation of the repository. Here, we fetch the octokit/octokit.rb repository:

```
GET /repos/octokit/octokit.rb
```

The documentation provides an example response for each API method. The example response illustrates all attributes that are returned by that method.

Authentication

There are two ways to authenticate through blog API v3. Requests that require authentication will return `404` Not Found, instead of `403` Forbidden, in some places. This is to prevent the accidental leakage of private repositories to unauthorized users.

Basic authentication

```
curl -u "username" https://api.blog.com
```

OAuth2 token (sent in a header)

```
curl -H "Authorization: token OAUTH-TOKEN" https://api.blog.com
```

OAuth2 key/secret

```
curl 'https://api.blog.com/users/whatever?client_id=xxxx&client_secret=yyyy'
```

Using your client_id and client_secret does not authenticate as a user, it will only identify your OAuth application to increase your rate limit. Permissions are only granted to users, not applications, and you will only get back data that an unauthenticated user would see. For this reason, you should only use the OAuth2 key/secret in server-to-server scenarios. Don't leak your OAuth application's client secret to your users.

Failed login limit

Authenticating with invalid credentials will return 401 Unauthorized:

```
curl -i https://api.blog.com -u foo:bar
HTTP/1.1 401 Unauthorized
{
  "message": "Bad credentials",
  "documentation_url": "https://developer.blog.com/v3"
}
```

After detecting several requests with invalid credentials within a short period, the API will temporarily reject all authentication attempts for that user (including ones with valid credentials) with 403 Forbidden:

```
curl -i https://api.blog.com -u valid_username:valid_password
HTTP/1.1 403 Forbidden
{
  "message": "Maximum number of login attempts exceeded. Please try again later.",
  "documentation_url": "https://developer.blog.com/v3"
}
```

Parameters

Many API methods take optional parameters. For GET requests, any parameters not specified as a segment in the path can be passed as an HTTP query string parameter:

```
curl -i "https://api.blog.com/user_id/aritcle_id"
```

In this example, the server will return the blog page correspounding to the user_id and blog_id

For POST, PATCH, PUT, and DELETE requests, parameters not included in the URL should be encoded as JSON with a Content-Type of 'application/json':

```
curl -i -u username -d '{"scopes":["public_repo"]}'
https://api.blog.com/authorizations
```

Root endpoint

You can issue a GET request to the root endpoint to get all the endpoint of all blogs that the REST API v3 supports:

```
curl https://api.blog.com
```

Client errors

There are three possible types of client errors on API calls that receive request bodies:

1. Sending invalid JSON will result in a 400 Bad Request response.

```
HTTP/1.1 400 Bad Request
Content-Length: 35

{"message":"Problems parsing JSON"}
```

2. Sending the wrong type of JSON values will result in a 400 Bad Request response.

```
HTTP/1.1 400 Bad Request
Content-Length: 40

{"message":"Body should be a JSON object"}
```

3. Sending invalid fields will result in a `422 Unprocessable` Entity response.

```
HTTP/1.1 422 Unprocessable Entity
Content-Length: 149

{
  "message": "Validation Failed",
  "errors": [
    {
      "resource": "Issue",
      "field": "title",
      "code": "missing_field"
    }
  ]
}
```

HTTP redirects

API v3 uses HTTP redirection where appropriate. Clients should assume that any request may result in a redirection. Receiving an HTTP redirection is not an error and clients should follow that redirect. Redirect responses will have a Location header field which contains the URI of the resource to which the client should repeat the requests.

Status Code	Description
301	Permanent redirection. The URI you used to make the request has been superseded by the one specified in the Location header field. This and all future requests to this resource should be directed to the new URI.
302, 307	Temporary redirection. The request should be repeated verbatim to the URI specified in the Location header field but clients should continue to use the original URI for future requests.

Other redirection status codes may be used in accordance with the HTTP 1.1 spec.

HTTP verbs

Where possible, API v3 strives to use appropriate HTTP verbs for each action.

Verb	Description
`HEAD`	Can be issued against any resource to get just the HTTP header info.
`GET`	Used for retrieving resources.
`POST`	Used for creating resources.
`PATCH`	Used for updating resources with partial JSON data. For instance, an Issue resource has `title` and `body` attributes. A PATCH request may accept one or more of the attributes to update the resource. PATCH is a relatively new and uncommon HTTP verb, so resource endpoints also accept `POST` requests.
`PUT`	Used for replacing resources or collections. For `PUT` requests with no `body` attribute, be sure to set the `Content-Length` header to zero.
`DELETE`	Used for deleting resources.

Hypermedia

All resources may have one or more `*_url` properties linking to other resources. These are meant to provide explicit URLs so that proper API clients don't need to construct URLs on their own. It is highly recommended that API clients use these. Doing so will make future upgrades of the API easier for developers. All URLs are expected to be proper RFC 6570 URI templates.

You can then expand these templates using something like the uri_template gem:

```
>> tmpl = URITemplate.new('/notifications{?since,all,participating}')
>> tmpl.expand
=> "/notifications"

>> tmpl.expand :all => 1
=> "/notifications?all=1"

>> tmpl.expand :all => 1, :participating => 1
=> "/notifications?all=1&participating=1"
```

Pagination

Requests that return multiple items will be paginated to 30 items by default. You can specify further pages with the `?page` parameter. For some resources, you can also set a custom page size up to 100 with the `?per_page` parameter.

```
curl 'https://api.blog.com/user/article?page=2&per_page=100'
```

Note that page numbering is 1-based and that omitting the `?page` parameter will return the first page.

Link header

Note: It's important to form calls with Link header values instead of constructing your own URLs.

The Link header includes pagination information:

```
Link: <https://api.blog.com/user/article?page=3&per_page=100>; rel="next",
  <https://api.blog.com/user/article?page=50&per_page=100>; rel="last"
```

The example includes a line break for readability.

This `Link` response header contains one or more Hypermedia link relations, some of which may require expansion as URI templates.

The possible `rel` values are:

Name	Description
`next`	The link relation for the immediate next page of results.
`last`	The link relation for the last page of results.
`first`	The link relation for the first page of results.
`prev`	The link relation for the immediate previous page of results.

Rate limiting

For API requests using Basic Authentication or OAuth, you can make up to 5000 requests per hour. Authenticated requests are associated with the authenticated user, regardless of whether Basic Authentication or an OAuth token was used. This means that all OAuth applications authorized by a user share the same quota of 5000 requests per hour when they authenticate with different tokens owned by the same user.

For unauthenticated requests, the rate limit allows for up to 60 requests per hour. Unauthenticated requests are associated with the originating IP address, and not the user making requests.

Note that the Search API has custom rate limit rules.

The returned HTTP headers of any API request show your current rate limit status:

```
curl -i https://api.blog.com/users/octocat
HTTP/1.1 200 OK
Date: Mon, 01 Jul 2013 17:27:06 GMT
Status: 200 OK
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 56
X-RateLimit-Reset: 1372700873

```

Header Name	Description
`X-RateLimit-Limit`	The maximum number of requests you're permitted to make per hour.
`X-RateLimit-Remaining`	The number of requests remaining in the current rate limit window.
`X-RateLimit-Reset`	The time at which the current rate limit window resets in UTC epoch seconds

```
new Date(1372700873 * 1000)
// => Mon Jul 01 2013 13:47:53 GMT-0400 (EDT)
```

If you exceed the rate limit, an error response returns:

```
HTTP/1.1 403 Forbidden
Date: Tue, 20 Aug 2013 14:50:41 GMT
Status: 403 Forbidden
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 0
X-RateLimit-Reset: 1377013266
{
   "message": "API rate limit exceeded for xxx.xxx.xxx.xxx. (But here's the good news: Authenticated requests get a higher rate limit. Check out the documentation for more details.)",
   "documentation_url": "https://developer.blog.com/v3/#rate-limiting"
}
```

You can check your rate limit status without incurring an API hit.

Increasing the unauthenticated rate limit for OAuth applications

If your OAuth application needs to make unauthenticated calls with a higher rate limit, you can pass your app's client ID and secret as part of the query string.

```
curl -i 'https://api.blog.com/users/whatever?client_id=xxxx&client_secret=yyyy'
HTTP/1.1 200 OK
Date: Mon, 01 Jul 2013 17:27:06 GMT
Status: 200 OK
X-RateLimit-Limit: 5000
X-RateLimit-Remaining: 4966
X-RateLimit-Reset: 1372700873
```

Note: Never share your client secret with anyone or include it in client-side browser code. Use the method shown here only for server-to-server calls.

Staying within the rate limit If you exceed your rate limit using Basic Authentication or OAuth, you can likely fix the issue by caching API responses and using conditional requests.

Abuse rate limits

In order to provide quality service on blog, additional rate limits may apply to some actions when using the API. For example, using the API to rapidly create content, poll aggressively instead of using webhooks, make multiple concurrent requests, or repeatedly request data that is computationally expensive may result in abuse rate limiting.

Abuse rate limits are not intended to interfere with legitimate use of the API. Your normal rate limits should be the only limit you target. To ensure you're acting as a good API citizen, check out our Best Practices guidelines.

If your application triggers this rate limit, you'll receive an informative response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json; charset=utf-8
Connection: close
{
  "message": "You have triggered an abuse detection mechanism and have been temporarily blocked from content creation. Please retry your request again later.",
  "documentation_url": "https://developer.blog.com/v3/#abuse-rate-limits"
}
```

User agent required

All API requests MUST include a valid User-Agent header. Requests with no User-Agent header will be rejected. We request that you use your blog username, or the name of your application, for the User-Agent header value. This allows us to contact you if there are problems.

Here's an example:

```
User-Agent: Awesome-Octocat-App
```

cURL sends a valid User-Agent header by default. If you provide an invalid User-Agent header via cURL (or via an alternative client), you will receive a 403 Forbidden response:

```
curl -iH 'User-Agent: ' https://api.blog.com/meta
HTTP/1.0 403 Forbidden
Connection: close
Content-Type: text/html
Request forbidden by administrative rules.
Please make sure your request has a User-Agent header.
Check https://developer.blog.com for other possible causes.
```

Conditional requests

Most responses return an ETag header. Many responses also return a Last-Modified header. You can use the values of these headers to make subsequent requests to those resources using the If-None-Match and If-Modified-Since headers, respectively. If the resource has not changed, the server will return a 304 Not Modified.

```
curl -i https://api.blog.com/user
HTTP/1.1 200 OK
Cache-Control: private, max-age=60
ETag: "644b5b0155e6404a9cc4bd9d8b1ae730"
Last-Modified: Thu, 05 Jul 2012 15:31:30 GMT
```

```
Status: 200 OK
Vary: Accept, Authorization, Cookie
X-RateLimit-Limit: 5000
X-RateLimit-Remaining: 4996
X-RateLimit-Reset: 1372700873
curl -i https://api.blog.com/user -H 'If-None-Match:
"644b5b0155e6404a9cc4bd9d8b1ae730"'
HTTP/1.1 304 Not Modified
Cache-Control: private, max-age=60
ETag: "644b5b0155e6404a9cc4bd9d8b1ae730"
Last-Modified: Thu, 05 Jul 2012 15:31:30 GMT
Status: 304 Not Modified
Vary: Accept, Authorization, Cookie
X-RateLimit-Limit: 5000
X-RateLimit-Remaining: 4996
X-RateLimit-Reset: 1372700873
curl -i https://api.blog.com/user -H "If-Modified-Since: Thu, 05 Jul 2012
15:31:30 GMT"
HTTP/1.1 304 Not Modified
Cache-Control: private, max-age=60
Last-Modified: Thu, 05 Jul 2012 15:31:30 GMT
Status: 304 Not Modified
Vary: Accept, Authorization, Cookie
X-RateLimit-Limit: 5000
X-RateLimit-Remaining: 4996
X-RateLimit-Reset: 1372700873
```

Cross origin resource sharing

The API supports Cross Origin Resource Sharing (CORS) for AJAX requests from any origin. You can read the CORS W3C Recommendation, or this intro from the HTML 5 Security Guide.

Here's a sample request sent from a browser hitting http://example.com:

```
curl -i https://api.blog.com -H "Origin: http://example.com"
HTTP/1.1 302 Found
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: ETag, Link, X-blog-OTP, X-RateLimit-Limit, X-
RateLimit-Remaining, X-RateLimit-Reset, X-OAuth-Scopes, X-Accepted-OAuth-
Scopes, X-Poll-Interval
```

This is what the CORS preflight request looks like:

```
curl -i https://api.blog.com -H "Origin: http://example.com" -X OPTIONS
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Authorization, Content-Type, If-Match, If-
Modified-Since, If-None-Match, If-Unmodified-Since, X-blog-OTP, X-Requested-
With
Access-Control-Allow-Methods: GET, POST, PATCH, PUT, DELETE
Access-Control-Expose-Headers: ETag, Link, X-blog-OTP, X-RateLimit-Limit, X-
RateLimit-Remaining, X-RateLimit-Reset, X-OAuth-Scopes, X-Accepted-OAuth-
Scopes, X-Poll-Interval
Access-Control-Max-Age: 86400
```

JSON-P callbacks

You can send a ?callback parameter to any GET call to have the results wrapped in a JSON function. This is typically used when browsers want to embed blog content in web pages by getting around cross domain issues. The response includes the same data output as the regular API, plus the relevant HTTP Header information.

```
curl https://api.blog.com?callback=foo
/**/foo({
  "meta": {
    "status": 200,
    "X-RateLimit-Limit": "5000",
    "X-RateLimit-Remaining": "4966",
    "X-RateLimit-Reset": "1372700873",
    "Link": [ // pagination headers and other links
      ["https://api.blog.com?page=2", {"rel": "next"}]
    ]
  },
  "data": {
    // the data
  }
})
```

You can write a JavaScript handler to process the callback. Here's a minimal example you can try out:

```
<html>
<head>
<script type="text/javascript">
function foo(response) {
  var meta = response.meta;
  var data = response.data;
  console.log(meta);
  console.log(data);
}

var script = document.createElement('script');
script.src = 'https://api.blog.com?callback=foo';

document.getElementsByTagName('head')[0].appendChild(script);
</script>
</head>

<body>
  <p>Open up your browser's console.</p>
</body>
</html>
```

All of the headers are the same String value as the HTTP Headers with one notable exception: Link. Link headers are pre-parsed for you and come through as an array of [url, options] tuples.

A link that looks like this:

```
Link: <url1>; rel="next", <url2>; rel="foo"; bar="baz"
... will look like this in the Callback output:
```

```
{
  "Link": [
    [
      "url1",
      {
        "rel": "next"
      }
    ],
    [
      "url2",
      {
        "rel": "foo",
        "bar": "baz"
      }
    ]
  ]
}
```

Timezones

Some requests that create new data, such as creating a new commit, allow you to provide time zone information when specifying or generating timestamps. We apply the following rules, in order of priority, to determine timezone information for API calls.

- Explicitly providing an ISO 8601 timestamp with timezone information
- Using the Time-Zone header
- Using the last known timezone for the user
- Defaulting to UTC without other timezone information

Explicitly providing an ISO 8601 timestamp with timezone information

For API calls that allow for a timestamp to be specified, we use that exact timestamp. An example of this is the Commits API.

These timestamps look something like 2014-02-27T15:05:06+01:00. Also see this example for how these timestamps can be specified.

Using the Time-Zone header

It is possible to supply a Time-Zone header which defines a timezone according to the list of names from the Olson database.

```
curl -H "Time-Zone: Europe/Amsterdam" -X POST
https://api.blog.com/repos/blog/linguist/contents/new_file.md
```

This means that w e generate a timestamp for the moment your API call is made in the timezone this header defines. For example, the Contents API generates a git commit for each addition or change and uses the current time as the timestamp. This header will determine the timezone used for generating that current timestamp.

Using the last known timezone for the user

If no Time-Zone header is specified and you make an authenticated call to the API, we use the last known timezone for the authenticated user. The last known timezone is updated whenever you browse the blog website.

Defaulting to UTC without other timezone information

If the steps above don't result in any information, we use UTC as the timezone to create the git commit.