Stack Overflow Enterprise API v2.3

This is the documentation for the v2.3 Stack Overflow Enterprise API. All methods are available under the /2.3/ path as of the 2021.3 Release, and new properties will only be returned if the /2.3/ version of a method is called.

If you're on 2021.2 or older releases, please refer to the 2021.3 Release Notes for a list of endpoint changes since API v2.2, and continue to use the /2.2/ path when calling methods until your SOE instance is upgraded to the 2021.3 release.

Below documentation and more can be found on your Stack Overflow Enterprise instance at /api/docs which also includes an endpoint testing UI for building queries from within the browser.

General

All requests against the API require an api access key.

All API responses are JSON, we do support JSONP with the callback query parameter. Every response in the API is returned in a common "wrapper" object, for easier and more consistent parsing.

Additionally, all API responses are compressed. The Content-Encoding header is always set, but some proxies will strip this out. The proper way to decode API responses can be found here.

Developers can trim API responses down to just the fields they are interested in using custom filters. Many types have fields that are not normally returned (question bodies, for example) that can likewise be requested via a custom filter.

A number of methods in the Stack Overflow Enterprise API accept dates as parameters and return dates as properties, the format of these dates is consistent and documented. All dates in the Stack Overflow API are in unix epoch time.

Unless otherwise noted, the maximum size of any page is 100, any {ids} parameter likewise is capped at 100 elements, all indexes start at 1.

If a parameter name is plural it accepts vectorized requests, otherwise a single value may be passed.

It is possible to compose reasonably complex queries against the sites using the min, max, fromdate, todate, and sort parameters. Most, but not all, methods accept some or all of these parameters, the documentation for individual methods will highlight which do. Most methods also have a common set of paging parameters.

Per-Site Methods

These are all available routes for the Stack Overflow Enterprise API.

Routes listed as auth required require OAuth 2.0 authentication.

Answers

answers

Get all answers on the site.

answers/{ids}

Get answers identified by a set of ids.

answers/{id}/accept - auth required

Accepts an answer.

answers/{id}/accept/undo - auth required

Undoes an accept on an answer.

answers/{ids}/comments

Get comments on the answers identified by a set of ids.

answers/{id}/delete - auth required

Deletes an answer.

answers/{id}/downvote - auth required

Downvotes an answer.

answers/{id}/downvote/undo - auth required

Undoes a downvote on an answer.

answers/{id}/edit - auth required

Edits an existing answer.

answers/{id}/flags/options - auth required

Returns the different flags the user can create for the answer.

answers/{id}/flags/add - auth required

Casts a flag against the answer.

answers/{ids}/questions

Gets all questions the answers identified by ids are on.

answers/{id}/upvote - auth required

Upvotes an answer.

answers/{id}/upvote/undo - auth required

Undoes an upvote on an answer.

answers/{id}/suggested-edit/add - auth required

Creates a suggested edit on an existing answer.

Articles

articles

Get all articles on the site.

articles/{ids}

Get the articles identified by a set of ids.

articles/{id}/delete

Deletes the given article. [auth required]

articles/{id}/edit

Edits the given article. [auth required]

articles/{ids}/linked

Get the questions that are linked to the articles identified by a set of ids.

articles/add

Creates a new article. [auth required]

Badges

badges

Get all badges on the site, in alphabetical order.

badges/{ids}

Get the badges identified by ids.

badges/name

Get all non-tagged-based badges in alphabetical order.

badges/recipients

Get badges recently awarded on the site.

badges/{ids}/recipients

Get the recent recipients of the given badges.

badges/tags

Get all tagged-based badges in alphabetical order.

Comments

comments

Get all comments on the site.

comments/{ids}

Get comments identified by a set of ids.

comments/{id}/delete - auth required

Delete a comment identified by its id.

comments/{id}/edit - auth required

Edit a comment identified by its id.

comments/{id}/flags/add - auth required

Casts a flag on the given comment.

comments/{id}/flags/options - auth required

Returns valid flag options for the given comment.

comments/{id}/upvote - auth required

Casts an upvote on the given comment.

comments/{id}/upvote/undo - auth required

Undoes an upvote on the given comment.

Events

events

Get recent events that have occurred on the site. Effectively a stream of new users and content.

Info

info

Get information about the entire site.

Posts

posts

Get all posts (questions and answers) in the system.

posts/{ids}

Get all posts identified by a set of ids. Useful for when the type of post (question or answer) is not known.

posts/{ids}/comments

Get comments on the posts (question or answer) identified by a set of ids.

posts/{id}/comments/add - auth required

Create a new comment on the post identified by id.

posts/{id}/comments/render

Renders a hypothetical comment on the given post.

posts/{ids}/revisions

Get revisions on the set of posts in ids.

posts/{ids}/suggested-edits

Get suggested edits on the set of posts in ids.

Privileges

privileges

Get all the privileges available on the site.

Questions

questions

Get all questions on the site.

questions/{ids}

Get the questions identified by a set of ids.

questions/{ids}/answers

Get the answers to the questions identified by a set of ids.

questions/{id}/answers/add - auth required

Creates an answer on the given question.

questions/{id}/answers/render

Renders a hypothetical answer to a question.

questions/{id}/close/options - auth required

Returns valid flag options which are also close reasons for the question.

questions/{ids}/comments

Get the comments on the questions identified by a set of ids.

questions/{id}/delete - auth required

Deletes the given question.

questions/{id}/downvote - auth required

Casts a downvote on the given question.

questions/{id}/downvote/undo - auth required

Undoes a downvote on the given question.

questions/{id}/edit - auth required

Edits the given question.

questions/{id}/favorite - auth required

Bookmarks the given question. (Previously known as "favoriting" a question)

questions/{id}/favorite/undo - auth required

Undoes bookmarking the given question. (Previously known as "favoriting" a question)

questions/{id}/flags/add - auth required

Casts a flag on the given question.

questions/{id}/flags/options - auth required

Returns valid flag options for the given question.

questions/{ids}/linked

Get the questions that link to the questions identified by a set of ids.

questions/{ids}/related

Get the questions that are related to the questions identified by a set of ids.

questions/{id}/suggested-edit/add - auth required

Creates a suggested_edit on an existing question.

questions/{ids}/timeline

Get the timelines of the questions identified by a set of ids.

questions/{id}/upvote - auth required

Casts an upvote on the given question.

questions/{id}/upvote/undo - auth required

Undoes an upvote on the given question.

questions/add - auth required

Creates a new question.

questions/featured

Get all questions on the site with active bounties.

questions/no-answers

Get all questions on the site with no answers.

questions/render - auth required

Renders a hypothetical question.

questions/unanswered

Get all questions the site considers unanswered.

questions/unanswered/my-tags - auth required

Get questions considered unanswered within the user’s favorite or interesting tags.

Revisions

revisions/{ids}

Get all revisions identified by a set of ids.

Search

search

Search the site for questions meeting certain criteria.

search/advanced

Search the site for questions using most of the on-site search options.

similar

Search the site based on similarity to a title.

search/excerpts

Searches a site.

Sites

sites

Get information about this Stack Overflow Enterprise site.

Suggested Edits

suggested-edits

Get all the suggested edits on the site.

suggested-edits/{ids}

Get the suggested edits identified by a set of ids.

Tags

tags

Get the tags on the site.

tags/{tags}/info

Get tags on the site by their names.

tags/moderator-only

Get the tags on the site that only moderators can use.

tags/required

Get the tags on the site that fulfill required tag constraints.

tags/synonyms

Get all the tag synonyms on the site.

tags/{tags}/faq

Get frequently asked questions in a set of tags.

tags/{tags}/related

Get related tags, based on common tag pairings.

tags/{tags}/synonyms

Get the synonyms for a specific set of tags.

tags/{tag}/top-answerers/{period}

Get the top answer posters in a specific tag, either in the last month or for all time.

tags/{tag}/top-askers/{period}

Get the top question askers in a specific tag, either in the last month or for all time.

tags/{tags}/wikis

Get the wiki entries for a set of tags.

Users

All user methods that take an {ids} parameter have a /me equivalent method that takes an access_tokeninstead. These methods are provided for developer convenience, with the exception of plain /me, which is actually necessary for discovering which user authenticated to an application.

users

Get all users on the site.

users/{ids}

Get the users identified by a set of ids.

users/{ids}/answers

Get the answers posted by the users identified by a set of ids.

users/{ids}/badges

Get the badges earned by the users identified by a set of ids.

users/{ids}/comments

Get the comments posted by the users identified by a set of ids.

users/{ids}/comments/{toid}

Get the comments posted by a set of users in reply to another user.

users/{ids}/favorites

Get the questions bookmarked (previously known as "favorited") by users identified by a set of ids.

users/{ids}/mentioned

Get the comments that mention one of the users identified by a set of ids.

users/{id}/network-activity

Gets a user's activity across the Stack Exchange network.

users/{ids}/posts

Get all posts (questions and answers) owned by a set of users.

users/{id}/privileges

Get the privileges the given user has on the site.

users/{ids}/questions

Get the questions asked by the users identified by a set of ids.

users/{ids}/questions/featured

Get the questions on which a set of users, have active bounties.

users/{ids}/questions/no-answers

Get the questions asked by a set of users, which have no answers.

users/{ids}/questions/unaccepted

Get the questions asked by a set of users, which have at least one answer but no accepted answer.

users/{ids}/questions/unanswered

Get the questions asked by a set of users, which are not considered to be adequately answered.

users/{ids}/reputation

Get a subset of the reputation changes experienced by the users identified by a set of ids.

users/{ids}/reputation-history

Get a history of a user's reputation, excluding private events.

users/{ids}/reputation-history/full - auth required

Get a full history of a user’s reputation.

users/{ids}/suggested-edits

Get the suggested edits provided by users identified by a set of ids.

users/{ids}/tags

Get the tags that the users (identified by a set of ids) have been active in.

users/{id}/tags/{tags}/top-answers

Get the top answers a user has posted on questions with a set of tags.

users/{id}/tags/{tags}/top-questions

Get the top questions a user has posted with a set of tags.

users/{ids}/timeline

Get a subset of the actions of that have been taken by the users identified by a set of ids.

users/{id}/top-answer-tags

Get the top tags (by score) a single user has posted answers in.

users/{id}/top-question-tags

Get the top tags (by score) a single user has asked questions in.

users/{id}/top-tags

Get the top tags (by score) a single user has posted in.

users/moderators

Get the users who have moderation powers on the site.

users/moderators/elected

Get the users who are active moderators who have also won a moderator election.

users/{ids}/inbox - auth required

Get a user’s inbox.

users/{ids}/inbox/unread - auth required

Get the unread items in a user’s reputation.

users/{ids}/notifications

Get a user's notifications.

users/{ids}/notifications/unread

Get a user's unread notifications.

users/{ids}/tag-preferences

Get a given user's tag preferences.

users/{ids}/tag-preferences/edit

Edit a user's tag preferences

Inbox Items

inbox - auth required

Get a user's inbox, outside of the context of a site

inbox/unread - auth required

Get the unread items in a user's inbox, outside of the context of a site.

users/{id}/inbox - auth required

Get a user's inbox.

users/{id}/inbox/unread - auth required

Get the unread items in a user's inbox.

Notifications

notifications - auth required

Get a user's notifications, outside of the context of a site.

notifications/unread - auth required

Get a user's unread notifications, outside of the context of a site.

users/{id}/notifications - auth required

Get a user's notifications.

users/{id}/notifications/unread - auth required

Get a user's unread notifications.

Authentication

Overview

The Stack Overflow Enterprise API can be used as a read-only API, in which case it requires the use of an API Access Key to make successful requests against the API.

Users with Administrator permissions can also use a Write-enabled API that authenticates via OAuth.

Read-only API

Instructions for Creating a Key

Here are instructions for creating an API Access Key:

- Navigate to your user's API Access Key section.

- Type a name in the "Access Key Name" field, and click Create.
- Copy the newly created API Access Key to use in a request against the api.

Each API Access Key is linked to a user account in the Stack Overflow Enterprise site.

How to Use the Access Key in a Request

There are two different ways to use your API Access Key in a GET request:

- As a query parameter named 'key', e.g. https://your.base_url/api/2.2/questions?key=the)key)value((
- As a request header named 'X-API-Key'

The 'key' query parameter takes precedence over the 'X-API-Key' header value.

The site administrator can force the use of the request header by enabling the Api.ForceUseOfKeyInRequestHeader site setting. Enabling this site setting will disable the "Try It" interface on https://your.base_url/api/docs.

Write-enabled API

Discussion

The Stack Overflow Enterprise API offers user authentication via OAuth 2.0, specifically templated after Facebook's implementation. There are two flows, an explicit grant for server side applications and an implicit one for pure browser based ones.

The explicit OAuth 2.0 flow consists of the following steps:

1. Send a user to https://your.base_url/oauth, with these query string parameters
   - **client_id** - you can get this from your API Access Keys page.
   - **scope**
   - **redirect_uri** - must be under an apps registered domain
   - **state** - optional
2. The user approves your app
3. The user is redirected to redirect_uri, with these query string parameters
   - **code**
   - **state** - optional, only returned if provided in the first step
4. POST (application/x-www-form-urlencoded) the following parameters to https://your.base_url/oauth/access_token

- **client_id** - you can get this from your API Access Keys page.
    - **client_secret** - you can get this from your API Access Keys page after enabling write access.
    - **code** - from the previous step
    - **redirect_uri** - must be the same as the provided in the first step

This request is responded to with either an error (HTTP status code 400) or an access token of the form access_token=...&expires=1234. expires will only be set if scope does not include no_expiry, the use of which is strongly advised against unless your app truly needs perpetual access.

In order to get access_token and expires (if applicable) wrapped in a JSON object, POST to https://your.base_url/oauth/access_token/json instead.

The implicit OAuth 2.0 flow consists of the following steps:

1. Open a new window at https://your.base_url/oauth/dialog, with these query string parameters
    - **client_id** - you can get this from your API Access Keys page.
    - **scope (See below)**
    - **redirect_uri** - must be under an apps registered domain
    - **state** - optional
2. The user approves your app
3. The user is redirected to redirect_uri, with these parameters in the hash
    - **access_token**
    - **expires** - optional, only if scope doesn't contain no_expiry

The explicit flow should be used by server-side applications, with special care taken to never leak client_secret. Client side applications should use the implicit flow.

After you have authenticated a user once, regardless of flow, subsequent re-authorizations will occur without requiring user action. Naturally, should a user remove an API Access Key, then further actions will fail.

Scope

With an empty scope, authentication will only allow an application to identify a user via the /me method. In order to access other information, different scope values must be sent. Multiple values may be sent in scope by comma delimiting them.

- **read_inbox** - access a user's global inbox
- **no_expiry** - access_token's with this scope do not expire
- **write_access** - perform write operations as a user
- **private_info** - access full history of a user's private actions on the site

Desktop Applications

Desktop applications cannot participate directly in OAuth 2.0 flows, however the embeddable browser controls available in most frameworks make it possible to work around this limitation.

Desktop applications should use the implicit client-side flow, hosting the process within a browser control. For redirect_uri, a value of https://your.base_url/oauth/login_success should be used. Upon a successful authentication, access_token will be placed in the url hash as with a standard implicit authentication.

Errors

OAuth 2.0 reports redirection errors in one of two ways: displaying an error page to the user, or redirecting to an application with the error and error_description parameters. Which of these two occurs depends on what the exact error is. Any error that cast doubt on the application (for example, an unknown client_id) causes the first case, all others cause the later case. Note that the user rejecting an application is conceptually an error.

Possible errors are:

- invalid_request
- unauthorized_client
- access_denied
- unsupported_response_type
- invalid_scope
- server_error
- temporarily_unavailable

For the explicit flow, calls to https://your.base_url/oauth/access_token will respond with the same error codes, wrapped in a JSON object of the form

```
{ "error": { "type": "invalid_request", "message": "some reason" }
}
```

Note that there is no guarantee that message will be set.

Vectorized Requests

Discussion

Most methods that take ids in the API will take up to 100 of them in a single go. This allows applications to batch work and thereby avoid unnecessary round trips, which can be a significant user experience win on slow or high latency devices. Those methods with different vector limits will mention that in their individual documentation.

When passing a vector, separate each id with a semicolon. For example, /users/1;2;3;4;5?site=somesite would fetch users with ids 1 through 5 on somesite.

Vectors are not restricted to integer values, /tags/{tags}/synonyms takes a list of tags (strings) and /revisions/{ids} takes a list of revision ids (guids).

Note that for caching and throttling purposes, vectors are considered unordered. That is, /users/1;2;3 is semantically identical to /users/3;2;1.

Complex Queries

Discussion

Simple usage of the API focuses around getting large sets of data about sites quickly. It's fairly obvious how to grab all of a user's answers, even all of a large set of users' via vectorized requests, all recent comments, and so on. What's less obvious is how to cull our datasets to smaller chunks of data.

The API provides the sort, min, max, fromdate, and todate parameters on many methods to allow for more complicated queries. min and max specify the range of a field must fall in (that field being specified by sort) to be returned, while fromdate and todate always define the range of creation_date. Think these parameters as defining two "windows" in which data must fit to be returned.

min, max, fromdate, and todate are inclusive.

Duplicate Requests

Discussion

Sometimes, especially with spotty internet connections, a client can make a request to the API that is serviced but for which a response is never received. For example, an application may submit an upvote but due to loss of connectivity never receive an acknowledgement.

A reasonable solution to poor networking is to retry unacknowledged requests a small number of times. However, many actions on the Stack Exchange network can only be performed once leading to "mysterious" failures if an retried request actually succeeded earlier.

Starting in v2.2, the API accepts a request_id parameter along with every request. If a request_id is seen a second time in a small window the request will be failed immediately with a duplicate_request error. this means that any retried requests will fail sensibly, provided that the same request_id is used.

The exact window in which the API will recognize a duplicate request_id is subject to change, but clients can assume the window is no shorter than five minutes.

Custom Filters

Discussion

The API allows applications to exclude almost every field returned. For example, if an application did not care about a user's badge counts it could exclude user.badge_counts whenever it calls a method that returns users.

An application excludes fields by creating a filter (via /filter/create) and passing it to a method in the filter parameter.

Filters are immutable and non-expiring. An application can safely "bake in" any filters that are created, it is not necessary (or advisable) to create filters at runtime.

The motivation for filters are several fold. Filters allow applications to reduce API responses to just the fields they are concerned with, saving bandwidth. With the list of fields an application is actually concerned with, the API can avoid unnecessary queries thereby decreasing response time (and reducing load on our infrastructure). Finally, filters allow us to be more conservative in what the API returns by default without a proliferation of parameters (as was seen with body,answers, and comments in the 1.x API family).

Safety

Filters also carry a notion of safety, which is defined as follows. Any string returned as a result of an API call with a safe filter will be inline-able into HTML without script-injection concerns. That is to say, no additional sanitizing (encoding, HTML tag stripping, etc.) will be necessary on returned strings. Applications that wish to handle sanitizing themselves should create an unsafe filter. All filters are safe by default, under the assumption that double-encoding bugs are more desirable than script injections.

Note that this does not mean that "safe" filter is merely an "unsafe" one with all fields passed through UrlEncode(...). Many fields can and will contain HTML in all filter types (most notably, the *.body fields).

When using unsafe filters, the API returns the highest fidelity data it can reasonably access for the given request. This means that in cases where the "safe" data is the only accessible data it will be returned even in "unsafe" filters. Notably the *.body fields are unchanged, as they are stored in that form. Fields that are unchanged between safe and unsafe filters are denoted in their types documentation.

Built In Filters

The following filters are built in:

- default, each type documents which fields are returned under the default filter (for example, answers).
- withbody, which is default plus the *.body fields
- none, which is empty
- total, which includes just .total

Compatibility with V1.x

For ease of transition from earlier API versions, the filters _b, _ba, _bc, _bca, _a, _ac, and _c are also built in. These are unsafe, and exclude a combination of question and answer body, comments, and answers so as to mimic the body, answers, and comments parameters that have been removed in V2.0. New applications should not use these filters.

Paging

Discussion

Nearly all methods in the API accept the page and pagesize parameters for fetching specific pages of results from the API. page starts at and defaults to 1, pagesize can be any value between 0 and 100 and defaults to 30.

Fetching All Results

Oftentimes an application will be interested in fetching all the items that match specific criteria, but this can be complicated when the number of matches exceeds 100 (the maximum pagesize). To assist in this case, the API returns the has_more property on the common wrapper object if there are more results to be fetched. An application can use this property to fetch all results without having to speculatively issue queries or pay for the comparatively expensive total property.

Total

The total property is available on the common wrapper object (but not returned by default, add it via a filter) for cases where the count of items that match a set of constraints is more interesting than the items themselves.

total is also a useful property when displaying paging controls. In this case applications would want to include both items and total on a filter rather than making two separate requests.

Fetching total can be equally as expensive as fetching items. Put another way, an application fetching total when not needed is potentially halving its performance. It is for this reason that total is not returned by default.

Date Formats

Discussion

All dates in the API are in unix epoch time, which is the number of seconds since midnight UTC January 1st, 1970. The API does not accept or return fractional times, everything should be rounded to the nearest whole second.

The fromdate and todate parameters are accepted by many methods, and are always dates. min and maxparameters may accept dates, depending on the sort used.

All dates returned by the API are in properties ending _date. The documentation console indicates these with a calendar icon, hovering over them will show a human readable version.

Numbers

Discussion

The API typically works in integers, in all but a few cases returning integers and in all cases accepting only integers as parameters. The motivation for this is to prevent any unpleasant floating point confusion, especially across technologies that offer different floating point precision).

The few exceptions where a field on an object is a floating point value are indicated in the relevant type documentation pages. For example, the info object has three (questions_per_minute, answers_per_minute, and badges_per_minute) decimal fields.

Each method documentation page indicates the parameters that expect numeric values with a . All methods expect whole numbers, no method accepts decimal numbers.

The API guarantees that all numbers returned will fit in a signed 32-bit integer. Dates are returned as numbers as well, but are instead guaranteed to fit in a signed 64-bit integer.

Decompressing API Responses

Discussion

While our API is HTTP based, we chose to diverge from standard HTTP in one particular area. During normal operation, we guarantee that all responses are compressed, either with GZIP or DEFLATE. Both of these algorithms are rather old and commonplace, most platforms should have built-in tools for handling both and practically all will be able to handle at least GZIP.

Rationale

The motivation for this is simple, serving uncompressed content is a loss for all parties. Bandwidth is, in comparison to CPU time, exceptionally expensive and severely limited on many devices. It's really a no-brainer to require compression accordingly.

While effectively all browsers will always request compressed content, many (if not all) of the applications using our API will be on decidedly less mature HTTP stacks. The likelihood of many applications not opting into compression, and being materially worse for it, is unacceptable.

Errors

The API will attempt to preserve the guarantee of compression in the face of errors, however it is possible for errors to occur early enough in the stack for compression to not occur. In these cases, errors will be returned uncompressed.

History

There is a way to remain in compliance with the HTTP spec, which is to reject all requests that do not list "gzip" or "deflate" in their Accept-Encoding header. Unfortunately, this does not work in practice as far too many proxies (affecting ~1% of users in our experience) will strip out this header.

We experimented with this approach during the beta period of API version 1.0, and found that it effectively banned a small but non-trivial number of potential users. That many users access the Stack Exchange network from corporate machines exacerbates the problem.

How to properly consume API responses

First and foremost, always set the Accept-Encoding header. Our API will attempt to honor your requested encoding (either GZIP or DEFLATE), falling back to GZIP if the header doesn't arrive or is modified en route.

If Content-Encoding is set on the response, use the specified algorithm. If it is missing, assume GZIP.

If response is not compressed this suggests a proxy between the user and us is intentionally decompressing content, or errors are occurring very early in processing requests. You can detect uncompressed content by checking for the appropriate magic numbers, assuming your library cannot detect this error for you.

The API will never return an uncompressed response during normal operation.

Error Handling

Discussion

The Stack Overflow Enterprise API returns an error when a request to the API is not successful. The Method Calls section explains the different error codes that can be returned from the Stack Overflow Enterprise API.

Method Calls

Errors from method calls are reported on the common response wrapper object in the error_id, error_message, and error_name fields. Note that while it is possible to construct a filter that excludes these fields, in the face of an error there is no guarantee that filters will be applied.

The HTTP code will be 400 (Bad Request) for all errors unless the method was called via JSONP, in which case even an error will be returned as a 200 (OK). This is necessary, as a 400 code will generally prevent a client side app from reading the remaining error details if the call was via JSONP. In rare cases (typically dealing with network wide maintenance or hardware failure), errors may occur in processing a request before the API determines whether a request is via JSONP; in these cases a 400 (Bad Request) is returned.

Possible errors:

bad_parameter – 400

An invalid parameter was passed, this includes even "high level" parameters like key or site.

access_token_required – 401

A method that requires an access token (obtained via authentication) was called without one.

invalid_access_token – 402

An invalid access token was passed to a method.

access_denied – 403

A method which requires certain permissions was called with an access token that lacks those permissions.

no_method – 404

An attempt was made to call a method that does not exist. Note, calling methods that expect numeric ids (like /users/{ids}) with non-numeric ids can also result in this error.

key_required – 405

A method was called in a manner that requires an application key (generally, with an access token), but no key was passed.

duplicate_request – 409

A request identified by a request_id has already been run.

internal_error – 500

An unexpected error occurred in the API and has been logged.

throttle_violation – 502

An application has violated part of the rate limiting contract, so the request was terminated.

temporarily_unavailable – 503

Some or all of the API is unavailable. Applications should backoff on requests to the method invoked.

For testing purposes, the /errors/{id} will simulate any error given its code. For introspection purposes, the /errors method will return a list of all possible errors the API may return.

Common Wrapper Object

Discussion

All responses in the API share a common format, so as to make parsing these responses simpler.

The error_* fields, while technically eligible for filtering, will not actually be excluded in an error case. This is by design.

page and page_size are whatever was passed into the method.

If you're looking to just select total, exclude the items field in favor of excluding all the properties on the returned type.

When building filters, this common wrapper object has no name. Refer to it with a leading ., so the items field would be referred to via .items.

The backoff field is only set when the API detects the request took an unusually long time to run. When it is set an application must wait that number of seconds before calling that method again.

Fields

backoff	integer may be absent
error_id	integer, refers to an error may be absent
error_message	string may be absent
error_name	string may be absent
has_more	boolean
items	an array of the type found in type
Page, **X**	integer
Page_size, **X**	integer
quota_max	integer
quota_remaining	integer
Total, **X**	integer
Type, **X**	string

Fields marked with X are excluded in the default filter. All others are included in the default filter.

Throttles

Discussion

In order to prevent abuse the API implements a number of throttles.

Every application is subject to an IP based concurrent request throttle. If a single IP is making more than 30 requests a second, new requests will be dropped. The exact ban period is subject to change, but will be on the order of 30 seconds to a few minutes typically. Note that exactly what response an application gets (in terms of HTTP code, text, and so on) is undefined when subject to this ban; we consider > 30 requests/sec per IP to be very abusive and thus cut the requests off very harshly.

Types Of User Objects

Discussion

There are two different objects that represent a user in the API, the full user object and the smaller shallow_user object. Which is returned depends on the method being called, methods that are focused on users return the full object while others return the shallow one.

By way of example, /users returns full user objects but /questions returns shallow_users in question.owner.

When constructing filters, you need to be aware of this distinction as excluding properties on user does not exclude then on shallow_users , and vice versa.

While it is a rare occurrence, users can be deleted on Stack Exchange sites. Users can also never have existed, such as when a question is migrated between sites as there is no guarantee the asking user exists on both. When a user does not exist there is no way to get a full user object for them. However, in some cases a shallow_user record with user_type: "does_not_exist" can be returned; such as when a question with a deleted owner is fetched. Be aware that no fields other than user_type on a shallow_user are guaranteed to be set when the user does not exist, applications should deal gracefully with this case.