olympia Documentation

Release 3.0

Mozilla Addons Team

Jul 24, 2017

Contents

Add-ons Server is the codebase for https://addons.mozilla.org/; the source lives at https://github.com/mozilla/addons-server.

In the past, this project was *olympia*; documentation that refers to olympia refers to this project.

CHAPTER 1

Contents

1.1 Security Bug Reports

This code and it's associated production website are included in Mozilla's web and services bug bounty program. If you find a security vulnerability, please submit it via the process outlined in the program and FAQ pages. Further technical details about this application are available from the Bug Bounty Onramp page.

Please submit all security-related bugs through Bugzilla using the web security bug form.

Never submit security-related bugs through a Github Issue or by email.

1.2 External API

This shows you how to use the addons.mozilla.org API at `/api/v3/` which is hosted at the following URLs:

Environment	URL
Production	`https://addons.mozilla.org/api/v3/`
Staging	`https://addons.allizom.org/api/v3/`
Development	`https://addons-dev.allizom.org/api/v3/`

Production Connect to this API for all normal operation.

Staging or Development Connect to these APIs if you need to work with a scratch database or you're testing features that aren't available in production yet. Your production account is not linked to any of these APIs.

Dive into the *overview section* and the *authentication section* for an example of how to get started using the API.

1.2.1 Overview

This describes the details of the requests and responses you can expect from the addons.mozilla.org API.

Requests

All requests should be made with the header:

```
Content-type: application/json
```

Responses

Status Codes

There are some common API responses that you can expect to receive at times.

GET /api/v3/...

Status Codes

- 200 OK – Success.
- 201 Created – Creation successful.
- 202 Accepted – The request has been accepted for processing. This usually means one or more asyncronous tasks is being executed in the background so results aren't immediately visible.
- 204 No Content – Success (no content is returned).
- 400 Bad Request – There was a problem with the parameters sent with this request.
- 401 Unauthorized – Authentication is required or failed.
- 403 Forbidden – You are not permitted to perform this action.
- 404 Not Found – The requested resource could not be found.
- 500 Internal Server Error – An unknown error occurred.
- 503 Service Unavailable – The site is in maintenance mode at this current time and the operation can not be performed.

Bad Requests

When returning a `HTTP 400 Bad Request` response, the API will try to return some information about the error(s) in the body of the response, as a JSON object. The keys of that object indicate the field(s) that caused an error, and for each, a list of messages will be provided (often only one message will be present, but sometimes more). If the error is not attached to a specific field the key `non_field_errors` will be used instead.

Example:

```
{
    "username": ["This field is required."],
    "non_field_errors": ["Error not linked to a specific field."]
}
```

Unauthorized and Permission Denied

When returning `HTTP 401 Unauthorized` and `HTTP 403 Permission Denied` responses, the API will try to return some information about the error in the body of the response, as a JSON object. A `detail` property

will contain a message explaining the error. In addition, in some cases, an optional `code` property will be present and will contain a constant corresponding to specific problems to help clients address the situation programmatically. The constants are as follows:

Value	Description
ERROR_INVALID_HEADER	The `Authorization` header is invalid.
ERROR_SIGNATURE_EXPIRED	The signature of the token indicates it has expired.
ERROR_DECODING_SIGNATURE	The token was impossible to decode and probably invalid.

Pagination

By default, all endpoints returning a list of results are paginated. The default number of items per page is 25 and clients can use the *page_size* query parameter to change it to any value between 1 and 50. Exceptions to those rules are possible but will be noted in the corresponding documentation for affected endpoints.

The following properties will be available in paginated responses:

- *next*: the URL for the next page in the pagination.
- *previous*: the URL for the previous page in the pagination.
- *page_size*: The number of items per page in the pagination.
- *count*: the total number of records.
- *results*: the array containing the results for this page.

Translated fields

Fields that can be translated by users (typically name, description) have a special behaviour. The default is to return them as an object, with languages as keys and translations as values:

```
{
    "name": {
        "en-US": "Games",
        "fr": "Jeux",
        "kn": ""
    }
}
```

However, for performance, if you pass the *lang* parameter to a *GET* request, then only the most relevant translation (the specified language or the fallback, depending on whether a translation is available in the requested language) will be returned as a string.

```
{
    "name": "Games"
}
```

This behaviour also applies to *POST*, *PATCH* and *PUT* requests: you can either submit an object containing several translations, or just a string. If only a string is supplied, it will only be used to translate the field in the current language.

Cross Origin

All APIs are available with Cross-Origin Resource Sharing unless otherwise specified.

1.2.2 Authentication (External)

To access the API as an external consumer, you need to include a JSON Web Token (JWT) in the `Authorization` header for every request. This header acts as a one-time token that authenticates your user account. No JWT claims are made about the actual API request you are making.

If you are building an app that lives on the AMO domain, read the *documentation for internal authentication* instead.

Access Credentials

To create JWTs, first obtain a **key** and **secret** from the API Credentials Management Page.

Note: Keep your API keys secret and *never* commit them to a public code repository or share them with anyone, including Mozilla contributors.

If someone obtains your secret they can make API requests on behalf of your user account.

Create a JWT for each request

Prior to making every API request, you need to generate a fresh JWT. The JWT will have a short expiration time and is only valid for a single request so you can't cache or reuse it. You only need to include a few standard fields; here's what the raw JSON object needs to look like before it's signed:

```
{
    "iss": "your-api-key",
    "jti": "0.47362944623455405",
    "iat": 1447273096,
    "exp": 1447273156
}
```

iss This is a standard JWT claim identifying the *issuer*. Set this to the **API key** you generated on the credentials management page. For example: `user:543210:23`.

jti This is a standard JWT claim declaring a *JWT ID*. This value needs to have a high probability of being unique across all recent requests made by your issuer ID. This value is a type of cryptographic nonce designed to prevent replay attacks.

iat This is a standard JWT claim indicating the *issued at time*. It should be a Unix epoch timestamp and **must be in UTC time**.

exp This is a standard JWT claim indicating the *expiration time*. It should be a Unix epoch timestamp in UTC time and must be **no longer than five minutes** past the issued at time.

> Changed in version 2016-10-06: We increased the expiration time from 60 seconds to five minutes to workaround support for large and slow uploads.

Note: If you're having trouble authenticating, make sure your system clock is correct and consider synchronizing it with something like tlsdate.

Take this JSON object and sign it with the **API secret** you generated on the credentials management page. You must sign the JWT using the `HMAC-SHA256` algorithm (which is typically the default). The final JWT will be a blob of base64 encoded text, something like:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE0NDcyNzMwOTZ9.MG9LJiEK5_
↪Db8WpF5cWWRebXCtUB48EJzxKIBqQhSOo
```

Here is an example of creating a JWT in NodeJS using the node-jsonwebtoken library:

```
var jwt = require('jsonwebtoken');

var issuedAt = Math.floor(Date.now() / 1000);
var payload = {
  iss: 'your-api-key',
  jti: Math.random().toString(),
  iat: issuedAt,
  exp: issuedAt + 60,
};

var secret = 'your-api-secret';  // store this securely.
var token = jwt.sign(payload, secret, {
  algorithm: 'HS256',  // HMAC-SHA256 signing algorithm
});
```

Create an Authorization header

When making each request, put your generated JSON Web Token (JWT) into an HTTP Authorization header prefixed with `JWT`, like this:

```
Authorization: JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE0NDcyNzMwOTZ9.
↪MG9LJiEK5_Db8WpF5cWWRebXCtUB48EJzxKIBqQhSOo
```

Example request

Using the *profile* as an example endpoint, here's what a JWT authenticated HTTP request would look like in curl:

```
curl "https://addons.mozilla.org/api/v3/accounts/profile/" \
     -H "Authorization: JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJpYXQiOjE0NDcyNzMwOTZ9.MG9LJiEK5_Db8WpF5cWWRebXCtUB48EJzxKIBqQhSOo"
```

Find a JWT library

There are robust open source libraries for creating JWTs in all major programming languages.

1.2.3 Authentication (internal)

This documents how to use authentication in your API requests when you are working on a web application that lives on AMO domain or subdomain. If you are looking for how to authenticate with the API from an external client, using your API keys, read the *documentation for external authentication* instead.

When using this authentication mechanism, the server is responsible for creating an API Token when the user logs in, and sends it back in the response. The clients must then include that token as an `Authorization` header on requests that need authentication. The clients never generate JWTs themselves.

Fetching the token

A fresh token, valid for 30 days, is automatically generated and added to the responses of the following endpoints:

- `/api/v3/accounts/login/`
- `/api/v3/accounts/register/`
- `/api/v3/accounts/authenticate/`

A token may also be obtained through the JSON API as outlined in the *internal login JSON API* section. This is only accessible through the VPN and requires using the following endpoints:

- `/api/v3/internal/accounts/login/start/`
- `/api/v3/internal/accounts/login/`

The token is available in two forms:

- For the endpoints mentionned above, as a property called `token`.
- For all endpoints, as a cookie called `api_auth_token`. This cookie expires after 30 days and is set as `HttpOnly`.

The response will contain some profile data for personalization:

`GET /api/v3/accounts/login/`

Response JSON Object

- **`id`** (*`int`*) – The numeric user id.
- **`email`** (*`string`*) – Email address used by the user to log in and create this account.
- **`name`** (*`string`*) – The name chosen by the user, or the username if not set.
- **`picture_url`** (*`string`*) – URL to a photo of the user, or */static/img/anon_user.png* if not set.
- **`username`** (*`string`*) – username chosen by the user, used in the account url. If not set will be a randomly generated string.
- **`permissions`** (*`array`*) – A list of the additional *permissions* this user has.

Permissions can be any arbritary string in the format *app:action*. Either *app* or *action* can be the wildcard *, so **:** means the user has permission to do all actions (i.e. full admin).

The following are some commonly tested permissions; see https://github.com/mozilla/addons-server/blob/master/src/olympia/constants/permissions.py for the full list.

Value	Description
AdminTools:View	Can access the website admin interface index page. Inner pages may require other/additional permissions.
Addons:Edit	Allows viewing and editing of any add-ons details in developer tools.
Addons:Review	Can access the add-on reviewer tools to approve/reject add-on submissions.
Personas:Review	Can access the theme reviewer tools to approve/reject theme submissions.

Creating an Authorization header

When making an authenticated API request, put your generated API Token into an HTTP Authorization header prefixed with `Bearer`, like this:

```
Authorization: Bearer␣
↪eyJhdXRoX2hhc2giOiJiY2E0MTZkN2RiMGU3NjFmYTA2NDE4MjAzZWU1NTMwOTM4OGZhNzcxIiwidXNlcl9pZCI6MTIzNDV9:1
↪gNo3EWU8IfL8
```

1.2.4 Accounts

The following API endpoints cover a users account.

Account

This endpoint returns information about a user's account, by the account id. Most of the information is optional and provided by the user so may be missing or inaccurate.

GET /api/v3/accounts/account/(int:user_id|string:username)/

Response JSON Object

- **id** (*int*) – The numeric user id.
- **username** (*string*) – username chosen by the user, used in the account url. If not set will be a randomly generated string.
- **name** (*string*) – The name chosen by the user, or the username if not set.
- **average_addon_rating** (*float*) – The average rating for addons the developer has listed on the website.
- **num_addons_listed** (*int*) – The number of addons the developer has listed on the website.
- **biography** (*string|null*) – More details about the user.
- **created** (*string*) – The date when this user first logged in and created this account.
- **homepage** (*string|null*) – The user's website.
- **location** (*string|null*) – The location of the user.
- **occupation** (*string|null*) – The occupation of the user.
- **picture_url** (*string*) – URL to a photo of the user, or */static/img/anon_user.png* if not set.
- **picture_type** (*string|null*) – the image type (only 'image/png' is supported) if a user defined photo has been provided, or none if no photo has been provided.
- **is_addon_developer** (*boolean*) – The user has developed and listed add-ons on this website.
- **is_artist** (*boolean*) – The user has developed and listed themes on this website.

If you authenticate and access your own account by specifing your own `user_id` the following additional fields are returned. If you have *Users:Edit* permission you will see these extra fields for all user accounts.

GET /api/v3/accounts/account/(int:user_id|string:username)/

Response JSON Object

- **email** (*string*) – Email address used by the user to login and create this account.

- **display_name** (*string|null*) – The name chosen by the user.
- **is_verified** (*boolean*) – The user has been verified via FirefoxAccounts.
- **read_dev_agreement** (*boolean*) – The user has read, and agreed to, the developer agreement that is required to submit addons.
- **deleted** (*boolean*) – Is the account deleted.
- **last_login** (*string*) – The date of the last successful log in to the website.
- **last_login_ip** (*string*) – The IP address of the last successfull log in to the website.

Status Codes

- 200 OK – account found.
- 400 Bad Request – an error occurred, check the `error` value in the JSON.
- 404 Not Found – no account with that user id.

Important:

- `Biography` can contain HTML, or other unsanitized content, and it is the responsibiliy of the client to clean and escape it appropriately before display.

Profile

Note: This API requires *authentication*.

This endpoint is a shortcut to your own account. It returns an *account object*

GET /api/v3/accounts/profile/

Edit

Note: This API requires *authentication* and *Users:Edit* permission to edit accounts other than your own.

This endpoint allows some of the details for an account to be updated. Any fields in the *account* (or *self*) but not listed below are not editable and will be ignored in the patch request.

PATCH /api/v3/accounts/account/(int:user_id|string:username)/

Request JSON Object

- **biography** (*string|null*) – More details about the user. No links are allowed.
- **display_name** (*string|null*) – The name chosen by the user.
- **homepage** (*string|null*) – The user's website.
- **location** (*string|null*) – The location of the user.
- **occupation** (*string|null*) – The occupation of the user.

- **username** (*string|null*) – username to be used in the account url. The username can only contain letters, numbers, underscores or hyphens. All-number usernames are prohibited as they conflict with user-ids.

Uploading a picture

To upload a picture for the profile the request must be sent as content-type *multipart/form-data* instead of JSON. Images must be either PNG or JPG; the maximum file size is 4MB. Other *editable values* can be set at the same time.

PATCH /api/v3/accounts/account/(int:user_id|string:username)/

Request:

```
curl "https://addons.mozilla.org/api/v3/accounts/account/12345/"
    -g -XPATCH --form "picture_upload=@photo.png"
    -H "Authorization: Bearer <token>"
```

Parameters

- **user-id** – The numeric user id.

Form Parameters

- **picture_upload** – The user's picture to upload.

Request Headers

- Content-Type – multipart/form-data

Deleting the picture

To delete the account profile picture call the special endpoint.

DELETE /api/v3/accounts/account/(int:user_id|string:username)/picture

Delete

Note: This API requires *authentication* and *Users:Edit* permission to delete accounts other than your own.

Note: Accounts of users who are authors of Add-ons can't be deleted. All Add-ons (and Themes) must be deleted or transfered to other users first.

This endpoint allows the account to be deleted. The reviews and ratings created by the user will not be deleted; but all the user's details are cleared.

DELETE /api/v3/accounts/account/(int:user_id|string:username)/

Notifications List

Note: This API requires *authentication* and *Users:Edit* permission to list notifications on accounts other than your own.

This endpoint allows you to list the account notifications set for the specified user. The result is an unpaginated list of the fields below. There are currently 11 notification types.

GET /api/v3/accounts/account/(int:user_id|string:username)/notifications/

Response JSON Object

- **name** (*string*) – The notification short name.
- **enabled** (*boolean*) – If the notification is enabled (defaults to True).
- **mandatory** (*boolean*) – If the notification can be set by the user.

Notifications Update

Note: This API requires *authentication* and *Users:Edit* permission to set notification preferences on accounts other than your own.

This endpoint allows account notifications to be set or updated. The request should be a dict of *name*:True|False pairs. Any number of notifications can be changed; only non-mandatory notifications can be changed - attempting to set a mandatory notification will return an error.

POST /api/v3/accounts/account/(int:user_id|string:username)/notifications/

Request JSON Object

- **<name>** (*boolean*) – Is the notification enabled?

Super-creation

Note: This API requires *authentication*.

This allows you to generate a new user account and sign in as that user.

Important:

- Your API user must be in the `Accounts:SuperCreate` group to access this endpoint. Use `manage.py createsuperuser --add-to-supercreate-group` to create a superuser with proper access.
- This endpoint is not available in all *API environments*.

POST /api/v3/accounts/super-create/

Request:

Parameters

- **email** – assign the user a specific email address. A fake email will be assigned by default.
- **username** – assign the user a specific username. A random username will be assigned by default.
- **fxa_id** – assign the user a Firefox Accounts ID, like one returned in the `uuid` parameter of a profile request. This is empty by default, meaning the user's account will need to be migrated to a Firefox Account.

- **group** – assign the user to a permission group. Valid choices:
  - **reviewer**: can access add-on reviewer pages, formerly known as Editor Tools
  - **admin**: can access any protected page

```
curl "https://addons.mozilla.org/api/v3/accounts/super-create/" \
    -X POST -H "Authorization: JWT <jwt-token>"
```

Response:

```
{
    "username": "super-created-7ee304ce",
    "display_name": "Super Created 7ee304ce",
    "user_id": 10985,
    "email": "super-created-7ee304ce@addons.mozilla.org",
    "fxa_id": null,
    "groups": [],
    "session_cookie": {
        "encoded": "sessionid=.eJyrVopPLC3JiC8tTi2KT...",
        "name": "sessionid",
        "value": ".eJyrVopPLC3JiC8tTi2KT..."
    }
}
```

Status Codes

- 201 Created – Account created.
- 422 Unprocessable Entity – Incorrect request parameters.

The session cookie will enable you to sign in for a limited time as this new user. You can pass it to any login-protected view like this:

```
curl --cookie sessionid=... -s -D - \
    "https://addons.mozilla.org/en-US/developers/addon/submit/1" \
    -o /dev/null
```

Session

Log out of the current session. This is for use with the *internal authentication* that authenticates browser sessions.

DELETE /api/v3/accounts/session/

Request:

```
curl "https://addons.mozilla.org/api/v3/accounts/session/"
    -H "Authorization: Bearer <jwt-token>" -X DELETE
```

Response:

```
{
    "ok": true
}
```

Status Codes

- 200 OK – session logged out.

- 401 Unauthorized – authentication failed.

1.2.5 Activity

Note: These APIs are experimental and are currently being worked on. Endpoints may change without warning. The only authentication method available at the moment is *the internal one*.

Review Notes List

This endpoint allows you to list the approval/rejection review history for a version of an add-on.

GET /api/v3/addons/addon/(int:addon_id|string:addon_slug|string:addon_guid)/versions/(int: *id*) **/ revie**

Note: All add-ons require authentication and either reviewer permissions or a user account listed as a developer of the add-on.

Response JSON Object

- **count** (*int*) – The number of versions for this add-on.
- **next** (*string*) – The URL of the next page of results.
- **previous** (*string*) – The URL of the previous page of results.
- **results** (*array*) – An array of *per version review notes*.

Review Notes Detail

This endpoint allows you to fetch a single review note for a specific version of an add-on.

GET /api/v3/addons/addon/(int:addon_id|string:addon_slug|string:addon_guid)/versions/(int: *id*) **/ revie**

int: *id***/**

Response JSON Object

- **id** (*int*) – The id for a review note.
- **action** (*string*) – The *type of review note*.
- **action_label** (*string*) – The text label of the action.
- **user.id** (*int*) – The id of the reviewer or author who left the review note.
- **user.name** (*string*) – The name of the reviewer or author.
- **user.url** (*string*) – The link to the profile page for of the reviewer or author.
- **comments** (*string*) – The text content of the review note.
- **date** (*string*) – The date the review note was created.

Possible values for the `action` field:

Value	Description
approved	Version, or file in the version, was approved
rejected	Version, or file in the version, was rejected
review-requested	Developer requested review
more-information-requested	Reviewer requested more information from developer
super-review-requested	Add-on was referred to an admin for attention
comment	Reviewer added comment for other reviewers
review-note	Generic review comment

Incoming Mail End-point

This endpoint allows a mail server or similar to submit a json object containing single email into AMO which will be processed. The only type of email currently supported is a reply to an activity email (e.g an add-on review, or a reply to an add-on review). Any other content or invalid emails will be discarded.

`POST /api/v3/activity/mail`

Note: This API endpoint uses a custom authentication method. The value *SecretKey* in the submitted json must match one defined in *settings.INBOUND_EMAIL_SECRET_KEY*. The IP address of the request must match one defined in *settings.ALLOWED_CLIENTS_EMAIL_API*, if defined.

Request JSON Object

- **`SecretKey`** (*`string`*) – A value that matches *settings.INBOUND_EMAIL_SECRET_KEY*.
- **`Message.TextBody`** (*`string`*) – The plain text body of the email.
- **`To`** (*`array`*) – Array of To email addresses. All will be parsed, and the first matching the correct format used.
- **`To[].EmailAddress`** (*`string`*) – An email address in the format *reviewreply+randomuuidstring@addons.mozilla.org*.

1.2.6 Add-ons

Note: These APIs are experimental and are currently being worked on. Endpoints may change without warning. The only authentication method available at the moment is *the internal one*.

Featured

This endpoint allows you to list featured add-ons matching some parameters. Results are sorted randomly and therefore, the standard pagination parameters are not accepted. The query parameter `page_size` is allowed but only serves to customize the number of results returned, clients can not request a specific page.

`GET /api/v3/addons/featured/`

Query Parameters

- **app** (*string*) – **Required**. Filter by *add-on application* availability.
- **category** (*string*) – Filter by *category slug*. `app` and `type` parameters need to be set, otherwise this parameter is ignored.
- **lang** (*string*) – Request add-ons featured for this specific language to be returned alongside add-ons featured globally. Also activate translations for that query. (See *translated fields*)
- **type** (*string*) – Filter by *add-on type*.
- **page_size** (*int*) – Maximum number of results to return. Defaults to 25.

Response JSON Object

- **results** (*array*) – An array of *add-ons*.

Search

This endpoint allows you to search through public add-ons.

GET /api/v3/addons/search/

Query Parameters

- **q** (*string*) – The search query.
- **app** (*string*) – Filter by *add-on application* availability.
- **appversion** (*string*) – Filter by application version compatibility. Pass the full version as a string, e.g. `46.0`. Only valid when the `app` parameter is also present.
- **author** (*string*) – Filter by exact author username.
- **category** (*string*) – Filter by *category slug*. `app` and `type` parameters need to be set, otherwise this parameter is ignored.
- **lang** (*string*) – Activate translations in the specific language for that query. (See *translated fields*)
- **page** (*int*) – 1-based page number. Defaults to 1.
- **page_size** (*int*) – Maximum number of results to return for the requested page. Defaults to 25.
- **platform** (*string*) – Filter by *add-on platform* availability.
- **tag** (*string*) – Filter by exact tag name. Multiple tag names can be specified, separated by comma(s).
- **type** (*string*) – Filter by *add-on type*.
- **sort** (*string*) – The sort parameter. The available parameters are documented in the *table below*.

Response JSON Object

- **count** (*int*) – The number of results for this query.
- **next** (*string*) – The URL of the next page of results.
- **previous** (*string*) – The URL of the previous page of results.
- **results** (*array*) – An array of *add-ons*.

Available sorting parameters:

Parameter	Description
created	Creation date, descending.
downloads	Number of weekly downloads, descending.
hotness	Hotness (average number of users progression), descending.
rating	Bayesian rating, descending.
relevance	Search query relevance, descending.
updated	Last updated date, descending.
users	Average number of daily users, descending.

The default is to sort by relevance if a search query (`q`) is present, otherwise sort by number of weekly downloads, descending.

You can combine multiple parameters by separating them with a comma. For instance, to sort search results by downloads and then by creation date, use *sort=downloads,created*.

Autocomplete

Similar to *add-ons search endpoint* above, this endpoint allows you to search through public add-ons. Because it's meant as a backend for autocomplete though, there are a couple key differences:

- No pagination is supported. There are no `next`, `prev` or `count` fields, and passing `page_size` or `page` has no effect, a maximum of 10 results will be returned at all times.
- Only a subset of fields are returned.

`GET /api/v3/addons/autocomplete/`

Query Parameters

- **`q`** (*`string`*) – The search query.
- **`app`** (*`string`*) – Filter by *add-on application* availability.
- **`appversion`** (*`string`*) – Filter by application version compatibility. Pass the full version as a string, e.g. `46.0`. Only valid when the `app` parameter is also present.
- **`author`** (*`string`*) – Filter by exact author username.
- **`category`** (*`string`*) – Filter by *category slug*. `app` and `type` parameters need to be set, otherwise this parameter is ignored.
- **`lang`** (*`string`*) – Activate translations in the specific language for that query. (See *translated fields*)
- **`platform`** (*`string`*) – Filter by *add-on platform* availability.
- **`tag`** (*`string`*) – Filter by exact tag name. Multiple tag names can be specified, separated by comma(s).
- **`type`** (*`string`*) – Filter by *add-on type*.
- **`sort`** (*`string`*) – The sort parameter. The available parameters are documented in the *table below*.

Response JSON Object

- **`results`** (*`array`*) – An array of *add-ons*. Only the `id`, `icon_url`, `name` and `url` fields are supported though.

Detail

This endpoint allows you to fetch a specific add-on by id, slug or guid.

Note: Non-public add-ons, or add-ons with only unlisted versions, require authentication and either reviewer permissions or a user account listed as a developer of the add-on.

Note: This endpoint will have the add-ons it can access reduced to public add-ons and non-public add-ons that you own in the future. If you have permission to access non-public add-ons you do not own please use the *internal add-on detail API*.

GET /api/v3/addons/addon/(int:id|string:slug|string:guid)/

Query Parameters

- **lang** (*string*) – Activate translations in the specific language for that query. (See *translated fields*)

Response JSON Object

- **id** (*int*) – The add-on id on AMO.
- **authors** (*array*) – Array holding information about the authors for the add-on.
- **authors[].id** (*int*) – The id for an author.
- **authors[].name** (*string*) – The name for an author.
- **authors[].url** (*string*) – The link to the profile page for an author.
- **authors[].picture_url** (*string*) – URL to a photo of the user, or */static/img/anon_user.png* if not set. For performance reasons this field is omitted from search results.
- **average_daily_users** (*int*) – The average number of users for the add-on per day.
- **categories** (*object*) – Object holding the categories the add-on belongs to.
- **categories[app_name]** (*array*) – Array holding the *category slugs* the add-on belongs to for a given *add-on application*. (Combine with the add-on `type` to determine the name of the category).
- **current_beta_version** (*object*) – Object holding the current beta *version* of the add-on, if it exists. For performance reasons the `release_notes` field is omitted and the `license` field omits the `text` property.
- **current_version** (*object*) – Object holding the current *version* of the add-on. For performance reasons the `release_notes` field is omitted and the `license` field omits the `text` property.
- **default_locale** (*string*) – The add-on default locale for translations.
- **description** (*string|object|null*) – The add-on description (See *translated fields*).
- **edit_url** (*string*) – The URL to the developer edit page for the add-on.
- **guid** (*string*) – The add-on extension identifier.

- **has_eula** (*boolean*) – The add-on has an End-User License Agreement that the user needs to agree with before installing (See *add-on EULA and privacy policy*).
- **has_privacy_policy** (*boolean*) – The add-on has a Privacy Policy (See *add-on EULA and privacy policy*).
- **homepage** (*string|object|null*) – The add-on homepage (See *translated fields*).
- **icon_url** (*string*) – The URL to icon for the add-on (including a cachebusting query string).
- **is_disabled** (*boolean*) – Whether the add-on is disabled or not.
- **is_experimental** (*boolean*) – Whether the add-on has been marked by the developer as experimental or not.
- **is_featured** (*boolean*) – The add-on appears in a featured collection.
- **is_source_public** (*boolean*) – Whether the add-on source is publicly viewable or not.
- **name** (*string|object|null*) – The add-on name (See *translated fields*).
- **last_updated** (*string*) – The date of the last time the add-on was updated by its developer(s).
- **latest_unlisted_version** (*object|null*) – Object holding the latest unlisted *version* of the add-on. This field is only present if the user has unlisted reviewer permissions, or is listed as a developer of the add-on.
- **previews** (*array*) – Array holding information about the previews for the add-on.
- **previews[].id** (*int*) – The id for a preview.
- **previews[].caption** (*string|object|null*) – The caption describing a preview (See *translated fields*).
- **previews[].image_url** (*string*) – The URL (including a cachebusting query string) to the preview image.
- **previews[].thumbnail_url** (*string*) – The URL (including a cachebusting query string) to the preview image thumbnail.
- **public_stats** (*boolean*) – Boolean indicating whether the add-on stats are public or not.
- **ratings** (*object*) – Object holding ratings summary information about the add-on.
- **ratings.count** (*int*) – The number of user ratings for the add-on.
- **ratings.average** (*float*) – The average user rating for the add-on.
- **ratings.bayesian_average** (*float*) – The bayesian average user rating for the add-on.
- **requires_payment** (*boolean*) – Does the add-on require payment, non-free services or software, or additional hardware.
- **review_url** (*string*) – The URL to the review page for the add-on.
- **slug** (*string*) – The add-on slug.
- **status** (*string*) – The *add-on status*.
- **summary** (*string|object|null*) – The add-on summary (See *translated fields*).

- **support_email** (*string|object|null*) – The add-on support email (See *translated fields*).
- **support_url** (*string|object|null*) – The add-on support URL (See *translated fields*).
- **tags** (*array*) – List containing the text of the tags set on the add-on.
- **theme_data** (*object*) – Object holding lightweight theme (Persona) data. Only present for themes (Persona).
- **type** (*string*) – The *add-on type*.
- **url** (*string*) – The (absolute) add-on detail URL.
- **weekly_downloads** (*int*) – The number of downloads for the add-on per week.

Possible values for the `status` field / parameter:

Value	Description
beta	Beta (Valid for files only)
lite	Preliminarily Reviewed
public	Fully Reviewed
deleted	Deleted
pending	Pending approval (Valid for themes only)
disabled	Disabled by Mozilla
rejected	Rejected (Valid for themes only)
nominated	Awaiting Full Review
incomplete	Incomplete
unreviewed	Awaiting Preliminary Review
lite-nominated	Preliminarily Reviewed and Awaiting Full Review
review-pending	Flagged for further review (Valid for themes only)

Possible values for the keys in the `compatibility` field, as well as the `app` parameter in the search API:

Value	Description
android	Firefox for Android
firefox	Firefox
seamonkey	SeaMonkey
thunderbird	Thunderbird

Possible values for the `current_version.files[].platform` field:

Value	Description
all	All
mac	Mac
linux	Linux
android	Android
windows	Windows

Possible values for the `type` field / parameter:

Note: For backwards-compatibility reasons, the value for Theme is `persona`. `theme` refers to a Complete Theme.

Value	Description
theme	Complete Theme
search	Search Engine
persona	Theme
language	Language Pack (Application)
extension	Extension
dictionary	Dictionary

Add-on and Version Submission

See *Uploading a version*.

Versions List

This endpoint allows you to list all versions belonging to a specific add-on.

GET /api/v3/addons/addon/(int:addon_id|string:addon_slug|string:addon_guid)/versions/

Note: Non-public add-ons, or add-ons with only unlisted versions, require authentication and either reviewer permissions or a user account listed as a developer of the add-on.

Query Parameters

- **filter** (*string*) – The *filter* to apply.
- **lang** (*string*) – Activate translations in the specific language for that query. (See *translated fields*)
- **page** (*int*) – 1-based page number. Defaults to 1.
- **page_size** (*int*) – Maximum number of results to return for the requested page. Defaults to 25.

Response JSON Object

- **count** (*int*) – The number of versions for this add-on.
- **next** (*string*) – The URL of the next page of results.
- **previous** (*string*) – The URL of the previous page of results.
- **results** (*array*) – An array of *versions*.

By default, the version list API will only return public versions (excluding versions that have incomplete, disabled, deleted, rejected or flagged for further review files) - you can change that with the `filter` query parameter, which may require authentication and specific permissions depending on the value:

Value	Description
all_without_unlisted	Show all listed versions attached to this add-on. Requires either reviewer permissions or a user account listed as a developer of the add-on.
all_with_unlisted	Show all versions (including unlisted) attached to this add-on. Requires either reviewer permissions or a user account listed as a developer of the add-on.
all_with_deleted	Show all versions attached to this add-on, including deleted ones. Requires admin permissions.
only_beta	Show beta versions only.

Version Detail

This endpoint allows you to fetch a single version belonging to a specific add-on.

GET /api/v3/addons/addon/(int:addon_id|string:addon_slug|string:addon_guid)/versions/(int:id)/

Query Parameters

- **lang** (*string*) – Activate translations in the specific language for that query. (See *translated fields*)

Response JSON Object

- **id** (*int*) – The version id.
- **channel** (*string*) – The version channel, which determines its visibility on the site. Can be either `unlisted` or `listed`.
- **compatibility** (*object*) – Object detailing which *applications* the version is compatible with.
- **compatibility[app_name].max** (*object*) – Maximum version of the corresponding app the version is compatible with. Should only be enforced by clients if `is_strict_compatibility_enabled` is `true`.
- **compatibility[app_name].min** (*object*) – Minimum version of the corresponding app the version is compatible with.
- **edit_url** (*string*) – The URL to the developer edit page for the version.
- **files** (*array*) – Array holding information about the files for the version.
- **files[].id** (*int*) – The id for a file.
- **files[].created** (*string*) – The creation date for a file.
- **files[].hash** (*string*) – The hash for a file.
- **files[].platform** (*string*) – The *platform* for a file.
- **files[].id** – The size for a file, in bytes.
- **files[].is_restart_required** (*boolean*) – Whether the file requires a browser restart to work once installed or not.
- **files[].is_webextension** (*boolean*) – Whether the file is a WebExtension or not.
- **files[].status** (*int*) – The *status* for a file.
- **files[].url** (*string*) – The (absolute) URL to download a file. An optional `src` query parameter can be added to indicate the source page (See *download sources*).

- **files[].permissions[]** (*array*) – Array of the webextension permissions for this File, as strings. Empty for non-webextensions.
- **license** (*object*) – Object holding information about the license for the version.
- **license.name** (*string|object|null*) – The name of the license (See *translated fields*).
- **license.text** (*string|object|null*) – The text of the license (See *translated fields*).
- **license.url** (*string|null*) – The URL of the full text of license.
- **release_notes** (*string|object|null*) – The release notes for this version (See *translated fields*).
- **reviewed** (*string*) – The date the version was reviewed at.
- **is_strict_compatibility_enabled** (*boolean*) – Whether or not this version has strictCompatibility. set.
- **version** (*string*) – The version number string for the version.

Add-on Feature Compatibility

This endpoint allows you to fetch feature compatibility information for a a specific add-on by id, slug or guid.

GET /api/v3/addons/addon/(int:id|string:slug|string:guid)/feature_compatibility/

Note: Non-public add-ons, or add-ons with only unlisted versions, require authentication and either reviewer permissions or a user account listed as a developer of the add-on.

Response JSON Object

- **e10s** (*int*) – The add-on e10s compatibility. Can be one of the following:

Value	Description
compatible	multiprocessCompatible marked as true in the install.rdf.
compatible-webextension	A WebExtension, so compatible.
incompatible	multiprocessCompatible marked as false in the install.rdf.
unknown	multiprocessCompatible has not been set.

Add-on EULA and Privacy Policy

This endpoint allows you to fetch an add-on EULA and privacy policy.

GET /api/v3/addons/addon/(int:id|string:slug|string:guid)/eula_policy/

Note: Non-public add-ons, or add-ons with only unlisted versions, require authentication and either reviewer permissions or a user account listed as a developer of the add-on.

Response JSON Object

- **eula** (*string|object|null*) – The text of the EULA, if present (See *translated fields*).
- **privacy_policy** (*string|object|null*) – The text of the Privacy Policy, if present (See *translated fields*).

Language Tools

This endpoint allows you to list all public language tools add-ons available on AMO.

GET /api/v3/addons/language-tools/

Note: Because this endpoint is intended to be used to feed a page that displays all available language tools in a single page, it is not paginated as normal, and instead will return all results without obeying regular pagination parameters. The ordering is left undefined, it's up to the clients to re-order results as needed before displaying the add-ons to the end-users.

Query Parameters

- **app** (*string*) – Mandatory. Filter by *add-on application* availability.
- **lang** (*string*) – Activate translations in the specific language for that query. (See *translated fields*)

Response JSON Object

- **results** (*array*) – An array of language tools.
- **results[].id** (*int*) – The add-on id on AMO.
- **results[].current_version** (*object*) – Object holding the current *version* of the add-on. For performance reasons the `release_notes` field is omitted and the `license` field omits the `text` property.
- **results[].default_locale** (*string*) – The add-on default locale for translations.
- **results[].name** (*string|object|null*) – The add-on name (See *translated fields*).
- **results[].locale_disambiguation** (*string*) – Free text field allowing clients to distinguish between multiple dictionaries in the same locale but different spellings. Only present when using the Language Tools endpoint.
- **results[].target_locale** (*string*) – For dictionaries and language packs, the locale the add-on is meant for. Only present when using the Language Tools endpoint.
- **results[].type** (*string*) – The *add-on type*.
- **results[].url** (*string*) – The (absolute) add-on detail URL.

1.2.7 Categories

Note: These APIs are experimental and are currently being worked on. Endpoints may change without warning.

Category List

Categories are defined by a name, a slug, a type and an application. Slugs are only guaranteed to be unique for a given `app` and `type` combination, and can therefore be re-used for different categories.

This endpoint is not paginated.

`GET /api/v3/addons/categories/`

Response JSON Object

- **`id`** (*`int`*) – The category id.
- **`name`** (*`string`*) – The category name. Returns the already translated string.
- **`slug`** (*`string`*) – The category slug. See *csv table* for more possible values.
- **`application`** (*`string`*) – Application, see *add-on application* for more details.
- **`misc`** (*`boolean`*) – Whether or not the category is miscellaneous.
- **`type`** (*`string`*) – Category type, see *add-on type* for more details.
- **`weight`** (*`int`*) – Category weight used in sort ordering.
- **`description`** (*`string`*) – The category description. Returns the already translated string.

Current categories

Name	Slug	Type	Application
Alerts & Updates	alerts-updates	extension	firefox
Appearance	appearance	extension	firefox
Bookmarks	bookmarks	extension	firefox
Download Management	download-management	extension	firefox
Feeds, News & Blogging	feeds-news-blogging	extension	firefox
Games & Entertainment	games-entertainment	extension	firefox
Language Support	language-support	extension	firefox
Photos, Music & Videos	photos-music-videos	extension	firefox
Privacy & Security	privacy-security	extension	firefox
Search Tools	search-tools	extension	firefox
Shopping	shopping	extension	firefox
Social & Communication	social-communication	extension	firefox
Tabs	tabs	extension	firefox
Web Development	web-development	extension	firefox
Other	other	extension	firefox
Animals	animals	theme	firefox
Compact	compact	theme	firefox
Large	large	theme	firefox
Miscellaneous	miscellaneous	theme	firefox
Modern	modern	theme	firefox
Nature	nature	theme	firefox
OS Integration	os-integration	theme	firefox
Retro	retro	theme	firefox
Sports	sports	theme	firefox
General	general	dictionary	firefox

Continued on next page

Table 1.1 – continued from previous page

Name	Slug	Type	Application
Bookmarks	bookmarks	search	firefox
Business	business	search	firefox
Dictionaries & Encyclopedias	dictionaries-encyclopedias	search	firefox
General	general	search	firefox
Kids	kids	search	firefox
Multiple Search	multiple-search	search	firefox
Music	music	search	firefox
News & Blogs	news-blogs	search	firefox
Photos & Images	photos-images	search	firefox
Shopping & E-Commerce	shopping-e-commerce	search	firefox
Social & People	social-people	search	firefox
Sports	sports	search	firefox
Travel	travel	search	firefox
Video	video	search	firefox
General	general	language	firefox
Abstract	abstract	persona	firefox
Causes	causes	persona	firefox
Fashion	fashion	persona	firefox
Film and TV	film-and-tv	persona	firefox
Firefox	firefox	persona	firefox
Foxkeh	foxkeh	persona	firefox
Holiday	holiday	persona	firefox
Music	music	persona	firefox
Nature	nature	persona	firefox
Other	other	persona	firefox
Scenery	scenery	persona	firefox
Seasonal	seasonal	persona	firefox
Solid	solid	persona	firefox
Sports	sports	persona	firefox
Websites	websites	persona	firefox
Appearance and Customization	appearance	extension	thunderbird
Calendar and Date/Time	calendar	extension	thunderbird
Chat and IM	chat	extension	thunderbird
Contacts	contacts	extension	thunderbird
Folders and Filters	folders-and-filters	extension	thunderbird
Import/Export	importexport	extension	thunderbird
Language Support	language-support	extension	thunderbird
Message Composition	composition	extension	thunderbird
Message and News Reading	message-and-news-reading	extension	thunderbird
Miscellaneous	miscellaneous	extension	thunderbird
Privacy and Security	privacy-and-security	extension	thunderbird
Tags	tags	extension	thunderbird
Compact	compact	theme	thunderbird
Miscellaneous	miscellaneous	theme	thunderbird
Modern	modern	theme	thunderbird
Nature	nature	theme	thunderbird
General	general	dictionary	thunderbird
General	general	language	thunderbird
Bookmarks	bookmarks	extension	seamonkey

Continued on next page

Table 1.1 – continued from previous page

Name	Slug	Type	Application
Downloading and File Management	downloading-and-file-management	extension	seamonkey
Interface Customizations	interface-customizations	extension	seamonkey
Language Support and Translation	language-support-and-translation	extension	seamonkey
Miscellaneous	miscellaneous	extension	seamonkey
Photos and Media	photos-and-media	extension	seamonkey
Privacy and Security	privacy-and-security	extension	seamonkey
RSS, News and Blogging	rss-news-and-blogging	extension	seamonkey
Search Tools	search-tools	extension	seamonkey
Site-specific	site-specific	extension	seamonkey
Web and Developer Tools	web-and-developer-tools	extension	seamonkey
Miscellaneous	miscellaneous	theme	seamonkey
General	general	dictionary	seamonkey
General	general	language	seamonkey
Device Features & Location	device-features-location	extension	android
Experimental	experimental	extension	android
Feeds, News, & Blogging	feeds-news-blogging	extension	android
Performance	performance	extension	android
Photos & Media	photos-media	extension	android
Security & Privacy	security-privacy	extension	android
Shopping	shopping	extension	android
Social Networking	social-networking	extension	android
Sports & Games	sports-games	extension	android
User Interface	user-interface	extension	android

1.2.8 Collections

The following API endpoints cover user created collections.

List

Note: This API requires *authentication* and *Collections:Edit* permission to list collections other than your own.

This endpoint allows you to list all collections authored by the specified user. The results are sorted by the most recently updated collection first.

GET /api/v3/accounts/account/(int:user_id|string:username)/collections/

Response JSON Object

- **count** (*int*) – The number of results for this query.
- **next** (*string*) – The URL of the next page of results.
- **previous** (*string*) – The URL of the previous page of results.
- **results** (*array*) – An array of *collections*.

Detail

This endpoint allows you to fetch a single collection by its `slug`. It returns any `public` collection by the specified user. You can access a non-`public` collection only if it was authored by you, the authenticated user. If your account

has the *Collections:Edit* permission then you can access any collection.

GET /api/v3/accounts/account/(int:user_id|string:username)/collections/(string:*collection_slug*)/

Response JSON Object

- **id** (*int*) – The id for the collection.
- **addon_count** (*int*) – The number of add-ons in this collection.
- **author.id** (*int*) – The id of the author (creator) of the collection.
- **author.name** (*string*) – The name of the author.
- **author.url** (*string*) – The link to the profile page for of the author.
- **default_locale** (*string*) – The default locale of the description and name fields. (See *translated fields*).
- **description** (*string|object|null*) – The description the author added to the collection. (See *translated fields*).
- **modified** (*string*) – The date the collection was last updated.
- **name** (*string|object*) – The name of the collection. (See *translated fields*).
- **public** (*boolean*) – Whether the collection is *listed* - publicly viewable.
- **slug** (*string*) – The name used in the URL.
- **url** (*string*) – The (absolute) collection detail URL.
- **uuid** (*string*) – A unique identifier for this collection; primarily used to count addon installations that come via this collection.

Create

Note: This API requires *authentication*.

This endpoint allows a collection to be created under your account. Any fields in the *collection* but not listed below are not settable and will be ignored in the request.

POST /api/v3/accounts/account/(int:user_id|string:username)/collections/

Request JSON Object

- **default_locale** (*string|null*) – The default locale of the description and name fields. Defaults to *en-US*. (See *translated fields*).
- **description** (*string|object|null*) – The description the author added to the collection. (See *translated fields*).
- **name** (*string|object*) – The name of the collection. (required) (See *translated fields*).
- **public** (*boolean*) – Whether the collection is *listed* - publicly viewable. Defaults to *True*.
- **slug** (*string*) – The name used in the URL (required).

Edit

Note: This API requires *authentication* and *Collections:Edit* permission to edit collections other than your own.

This endpoint allows some of the details for a collection to be updated. Any fields in the *collection* but not listed below are not editable and will be ignored in the patch request.

PATCH /api/v3/accounts/account/(int:user_id|string:username)/collections/(string: *collection_slug*) **/**

Request JSON Object

- **default_locale** (*string*) – The default locale of the description and name fields. (See *translated fields*).
- **description** (*string|object|null*) – The description the author added to the collection. (See *translated fields*).
- **name** (*string|object*) – The name of the collection. (See *translated fields*).
- **public** (*boolean*) – Whether the collection is *listed* - publicly viewable.
- **slug** (*string*) – The name used in the URL.

Delete

Note: This API requires *authentication* and *Collections:Edit* permission to delete collections other than your own.

This endpoint allows the collection to be deleted.

DELETE /api/v3/accounts/account/(int:user_id|string:username)/collections/(string: *collection_slug*) **/**

Collection Add-ons List

This endpoint lists the add-ons in a collection, together with collector's notes.

GET /api/v3/accounts/account/(int:user_id|string:username)/collections/(string: *collection_slug*) **/ addons/**

Query Parameters

- **sort** (*string*) – The sort parameter. The available parameters are documented in the *table below*.

Response JSON Object

- **count** (*int*) – The number of results for this query.
- **next** (*string*) – The URL of the next page of results.
- **previous** (*string*) – The URL of the previous page of results.
- **results** (*array*) – An array of *items* in this collection.

Available sorting parameters:

Parameter	Description
added	Date the add-on was added to the collection, ascending.
popularity	Number of total weekly downloads of the add-on, ascending.
name	Add-on name, ascending.

All sort parameters can be reversed, e.g. '-added' for descending dates. The default sorting is by popularity, descending ('-popularity').

Collection Add-ons Detail

This endpoint gets details of a single add-on in a collection, together with collector's notes.

GET /api/v3/accounts/account/(int:user_id|string:username)/collections/(string: *collection_slug*) **/addons/(int:addon_id|stri**

Response JSON Object

- **addon** (*object*) – The *add-on* for this item.
- **notes** (*string|object|null*) – The collectors notes for this item. (See *translated fields*).
- **downloads** (*int*) – The downloads that occured via this collection.

Collection Add-ons Create

Note: This API requires *authentication* and *Collections:Edit* permission to edit collections other than your own.

This endpoint allows a single add-on to be added to a collection, optionally with collector's notes.

POST /api/v3/accounts/account/(int:user_id|string:username)/collections/(string: *collection_slug*) **/addons/**

Request JSON Object

- **addon** (*string*) – The add-on id or slug to be added (required).

- **notes** (*string|object|null*) – The collectors notes for this item. (See *translated fields*).

Collection Add-ons Edit

Note: This API requires *authentication* and *Collections:Edit* permission to edit collections other than your own.

This endpoint allows the collector's notes for single add-on to be updated.

PATCH /api/v3/accounts/account/(int:user_id|string:username)/collections/(string: *collection_slug*) **/addons/(int:addon_id|st**

Request JSON Object

- **notes** (*string|object|null*) – The collectors notes for this item. (See *translated fields*).

Collection Add-ons Delete

Note: This API requires *authentication* and *Collections:Edit* permission to edit collections other than your own.

This endpoint allows a single add-on to be removed from a collection.

DELETE /api/v3/accounts/account/(int:user_id|string:username)/collections/(string: *collection_slug*) **/addons/(int:addon_id|s**

1.2.9 Discovery

Note: These APIs are experimental and are currently being worked on. Endpoints may change without warning.

Discovery Content

This endpoint allows you to fetch content for the new Discovery Pane in Firefox (about:addons).

GET /api/v3/discovery/

Response JSON Object

- **count** (*int*) – The number of results for this query.
- **results** (*array*) – The array containing the results for this query.

- **results[].heading** (*string*) – The heading for this item. May contain some HTML tags.
- **results[].description** (*string|null*) – The description for this item, if any. May contain some HTML tags.
- **results[].addon** (*object*) – The *add-on* for this item. Only a subset of fields are present: `id`, `current_version` (with only the `compatibility` and `files` fields present), `guid`, `icon_url`, `name`, `slug`, `theme_data`, `type` and `url`.

1.2.10 Download Sources

When requesting an add-on file URL, clients have the option to indicate what is the source of the request. This will then be used to group by downloads by sources in the add-on statistics page.

To indicate the source, add the `src` query parameter to the download URL. The following values are recognized:

Name	Description
api	Add-ons Manager
discovery-promo	Add-ons Manager Promo
discovery-featured	Add-ons Manager Featured
discovery-learnmore	Add-ons Manager Learn More
ss	Search Suggestions
search	Search Results
homepagepromo	Homepage Promo
hp-btn-promo	Homepage Promo
hp-dl-promo	Homepage Promo
hp-hc-featured	Homepage Featured
hp-dl-featured	Homepage Featured
hp-hc-upandcoming	Homepage Up and Coming
hp-dl-upandcoming	Homepage Up and Coming
hp-dl-mostpopular	Homepage Most Popular
dp-btn-primary	Detail Page
dp-btn-version	Detail Page (bottom)
addondetail	Detail Page
addon-detail-version	Detail Page (bottom)
dp-btn-devchannel	Detail Page (Development Channel)
oftenusedwith	Often Used With
dp-hc-oftenusedwith	Often Used With
dp-dl-oftenusedwith	Often Used With
dp-hc-othersby	Others By Author
dp-dl-othersby	Others By Author
dp-hc-dependencies	Dependencies
dp-dl-dependencies	Dependencies
dp-hc-upsell	Upsell
dp-dl-upsell	Upsell
developers	Meet the Developer
userprofile	User Profile
version-history	Version History
sharingapi	Sharing
category	Category Pages

Continued on next page

Table 1.2 – continued from previous page

Name	Description
collection	Collections
cb-hc-featured	Category Landing Featured Carousel
cb-dl-featured	Category Landing Featured Carousel
cb-hc-toprated	Category Landing Top Rated
cb-dl-toprated	Category Landing Top Rated
cb-hc-mostpopular	Category Landing Most Popular
cb-dl-mostpopular	Category Landing Most Popular
cb-hc-recentlyadded	Category Landing Recently Added
cb-dl-recentlyadded	Category Landing Recently Added
cb-btn-featured	Browse Listing Featured Sort
cb-dl-featured	Browse Listing Featured Sort
cb-btn-users	Browse Listing Users Sort
cb-dl-users	Browse Listing Users Sort
cb-btn-rating	Browse Listing Rating Sort
cb-dl-rating	Browse Listing Rating Sort
cb-btn-created	Browse Listing Created Sort
cb-dl-created	Browse Listing Created Sort
cb-btn-name	Browse Listing Name Sort
cb-dl-name	Browse Listing Name Sort
cb-btn-popular	Browse Listing Popular Sort
cb-dl-popular	Browse Listing Popular Sort
cb-btn-updated	Browse Listing Updated Sort
cb-dl-updated	Browse Listing Updated Sort
cb-btn-hotness	Browse Listing Up and Coming Sort
cb-dl-hotness	Browse Listing Up and Coming Sort
find-replacement	Find replacement service for obsolete add-ons

1.2.11 Internal

Note: These APIs are experimental and are currently being worked on. Endpoints may change without warning. The only authentication method available at the moment is *the internal one*.

Add-on Search

This endpoint allows you to search through all add-ons. It's similar to the *regular add-on search API*, but is not limited to public add-ons, and can return disabled, unreviewer, unlisted or even deleted add-ons.

Note: Requires authentication and *AdminTools::View* or *ReviewerAdminTools::View* permissions.

GET /api/v3/internal/addons/search/

Parameters

- **q** (*string*) – The search query.
- **app** (*string*) – Filter by *add-on application* availability.

- **appversion** (*string*) – Filter by application version compatibility. Pass the full version as a string, e.g. `46.0`. Only valid when the `app` parameter is also present.
- **platform** (*string*) – Filter by *add-on platform* availability.
- **type** (*string*) – Filter by *add-on type*.
- **status** (*string*) – Filter by *add-on status*.
- **sort** (*string*) – The sort parameter. See *add-on search sorting parameters*.

Response JSON Object

- **count** (*int*) – The number of results for this query.
- **next** (*string*) – The URL of the next page of results.
- **previous** (*string*) – The URL of the previous page of results.
- **results** (*array*) – An array of *add-ons*.

Add-on Detail

This endpoint allows you to retrieve the details of an add-on. It is the same as the *regular add-on detail API*, but that endpoint may have its scope reduced to public add-ons and add-ons you own in the future. If you need to access add-ons you do not own or that have been deleted and you have sufficient permissions use this endpoint.

> **Note:** Unlisted or non-public add-ons require authentication and either reviewer permissions or a user account listed as a developer of the add-on.

GET /api/v3/addons/addon/(int:id|string:slug|string:guid)/

Query Parameters

- **lang** (*string*) – Activate translations in the specific language for that query. (See *translated fields*)

Response JSON Object

- **id** (*int*) – The add-on id on AMO.
- **authors** (*array*) – Array holding information about the authors for the add-on.
- **authors[].id** (*int*) – The id for an author.
- **authors[].name** (*string*) – The name for an author.
- **authors[].url** (*string*) – The link to the profile page for an author.
- **average_daily_users** (*int*) – The average number of users for the add-on per day.
- **categories** (*object*) – Object holding the categories the add-on belongs to.
- **categories[app_name]** (*array*) – Array holding the *category slugs* the add-on belongs to for a given *add-on application*. (Combine with the add-on `type` to determine the name of the category).
- **compatibility** (*object*) – Object detailing the add-on *add-on application* and version compatibility.
- **compatibility[app_name].max** (*object*) – Maximum version of the corresponding app the add-on is compatible with.

- **compatibility[app_name].min** (*object*) – Minimum version of the corresponding app the add-on is compatible with.
- **current_beta_version** (*object*) – Object holding the current beta *version* of the add-on, if it exists. For performance reasons the `license` and `release_notes` fields are omitted.
- **current_version** (*object*) – Object holding the current *version* of the add-on. For performance reasons the `license` and `release_notes` fields are omitted.
- **default_locale** (*string*) – The add-on default locale for translations.
- **description** (*string|object|null*) – The add-on description (See *translated fields*).
- **edit_url** (*string*) – The URL to the developer edit page for the add-on.
- **guid** (*string*) – The add-on extension identifier.
- **has_eula** (*boolean*) – The add-on has an End-User License Agreement that the user needs to agree with before installing (See *add-on EULA and privacy policy*).
- **has_privacy_policy** (*boolean*) – The add-on has a Privacy Policy (See *add-on EULA and privacy policy*).
- **homepage** (*string|object|null*) – The add-on homepage (See *translated fields*).
- **icon_url** (*string*) – The URL to icon for the add-on (including a cachebusting query string).
- **is_disabled** (*boolean*) – Whether the add-on is disabled or not.
- **is_experimental** (*boolean*) – Whether the add-on has been marked by the developer as experimental or not.
- **is_listed** (*boolean*) – Whether the add-on is listed or not.
- **is_source_public** (*boolean*) – Whether the add-on source is publicly viewable or not.
- **name** (*string|object|null*) – The add-on name (See *translated fields*).
- **last_updated** (*string*) – The date of the last time the add-on was updated by its developer(s).
- **previews** (*array*) – Array holding information about the previews for the add-on.
- **previews[].id** (*int*) – The id for a preview.
- **previews[].caption** (*string|object|null*) – The caption describing a preview (See *translated fields*).
- **previews[].image_url** (*string*) – The URL (including a cachebusting query string) to the preview image.
- **previews[].thumbnail_url** (*string*) – The URL (including a cachebusting query string) to the preview image thumbnail.
- **public_stats** (*boolean*) – Boolean indicating whether the add-on stats are public or not.
- **ratings** (*object*) – Object holding ratings summary information about the add-on.
- **ratings.count** (*int*) – The number of user ratings for the add-on.
- **ratings.average** (*float*) – The average user rating for the add-on.

- **review_url** (*string*) – The URL to the review page for the add-on.
- **slug** (*string*) – The add-on slug.
- **status** (*string*) – The *add-on status*.
- **summary** (*string|object|null*) – The add-on summary (See *translated fields*).
- **support_email** (*string|object|null*) – The add-on support email (See *translated fields*).
- **support_url** (*string|object|null*) – The add-on support URL (See *translated fields*).
- **tags** (*array*) – List containing the text of the tags set on the add-on.
- **theme_data** (*object*) – Object holding lightweight theme (Persona) data. Only present for themes (Persona).
- **type** (*string*) – The *add-on type*.
- **url** (*string*) – The (absolute) add-on detail URL.
- **weekly_downloads** (*int*) – The number of downloads for the add-on per week.

Internal Login JSON API

The JSON API login flow is initiated by accessing the start endpoint which will add an `fxa_state` to the user's session and redirect them to Firefox Accounts. When the user finishes authenticating with Firefox Accounts they will be redirected to the client application which can make a request to the login endpoint to exchange the Firefox Accounts token and state for a JWT.

GET /api/v3/internal/accounts/login/start/

Parameters

- **to** (*string*) – A path to append to the state. The state will be returned from FxA as `state:path`, the path will be URL safe base64 encoded.

Status Codes

- 302 Found – Redirect user to Firefox Accounts.

1.2.12 Reviews

Note: These APIs are experimental and are currently being worked on. Endpoints may change without warning. The only authentication method available at the moment is *the internal one*.

List reviews

This endpoint allows you to fetch reviews for a given add-on or user. Either `addon` or `user` query parameters are required, and they can be combined together.

When `addon`, `user` and `version` are passed on the same request, `page_size` will automatically be set to `1`, since an user can only post one review per version of a given add-on. This can be useful to find out if a user has already posted a review for the current version of an add-on.

GET /api/v3/reviews/review/

Query Parameters

- **addon** (*string*) – The *add-on* id, slug, or guid to fetch reviews from. When passed, the reviews shown will always be the latest posted by each user on this particular add-on (which means there should only be one review per user in the results), unless the `version` parameter is also passed.
- **filter** (*string*) – The *filter(s)* to apply.
- **user** (*string*) – The user id to fetch reviews from.
- **show_grouped_ratings** (*boolean*) – Whether or not to show ratings aggregates for this add-on in the response (Use "true"/"1" as truthy values, "0"/"false" as falsy ones).
- **version** (*string*) – The version id to fetch reviews from.
- **page** (*int*) – 1-based page number. Defaults to 1.
- **page_size** (*int*) – Maximum number of results to return for the requested page. Defaults to 25.

Response JSON Object

- **count** (*int*) – The number of results for this query.
- **next** (*string*) – The URL of the next page of results.
- **previous** (*string*) – The URL of the previous page of results.
- **results** (*array*) – An array of *reviews*.
- **grouped_ratings** (*object*) – Only present if `show_grouped_ratings` query parameter is present. An object with 5 key-value pairs, the keys representing each possible rating (Though a number, it has to be converted to a string because of the JSON formatting) and the values being the number of times the corresponding rating has been posted for this add-on, e.g. `{"1": 4, "2": 8, "3": 15, "4": 16: "5": 23}`.

By default, the review list API will only return not-deleted reviews, and include reviews without text. You can change that with the `filter` query parameter. You can filter by multiple values, e.g. `filter=with_deleted,without_empty_body,with_yours`

Value	Description
with_deleted	Returns deleted reviews too. This requires the Addons:Edit permission.
without_empty_body	Excludes reviews that only contain a rating, and no textual content.
with_yours	Used in combination *without_empty_body* to include your own reviews, even if they have no text.

Detail

This endpoint allows you to fetch a review by its id.

GET /api/v3/reviews/review/(int: *id*)/

Response JSON Object

- **id** (*int*) – The review id.
- **addon** (*object*) – An object included for convenience that contains only two properties: `id` and `slug`, corresponding to the add-on id and slug.

- **body** (*string|null*) – The text of the review.
- **is_latest** (*boolean*) – Boolean indicating whether the review is the latest posted by the user on the same add-on.
- **previous_count** (*int*) – The number of reviews posted by the user on the same add-on before this one.
- **rating** (*int*) – The rating the user gave as part of the review.
- **reply** (*object|null*) – The review object containing the developer reply to this review, if any (The fields `rating`, `reply` and `version` are omitted).
- **title** (*string|null*) – The title of the review.
- **version.id** (*int*) – The add-on version id the review applies to.
- **version.version** (*string*) – The add-on version string the review applies to.
- **user** (*object*) – Object holding information about the user who posted the review.
- **user.id** (*string*) – The user id.
- **user.name** (*string*) – The user name.
- **user.url** (*string*) – The user profile URL.

Post

This endpoint allows you to post a new review for a given add-on and version. If successful a *review object* is returned.

> **Note:** Requires authentication.

POST /api/v3/reviews/review/

Request JSON Object

- **addon** (*string*) – The add-on id the review applies to (required).
- **body** (*string|null*) – The text of the review.
- **title** (*string|null*) – The title of the review.
- **rating** (*int*) – The rating the user wants to give as part of the review (required).
- **version** (*int*) – The add-on version id the review applies to (required).

Edit

This endpoint allows you to edit an existing review by its id. If successful a *review object* is returned.

> **Note:** Requires authentication and Addons:Edit permissions or the user account that posted the review.
>
> Only body, title and rating are allowed for modification.

PATCH /api/v3/reviews/review/(int: *id*)/

Request JSON Object

- **body** (*string|null*) – The text of the review.

- **title** (*string|null*) – The title of the review.
- **rating** (*int*) – The rating the user wants to give as part of the review.

Delete

This endpoint allows you to delete an existing review by its id.

> **Note:** Requires authentication and Addons:Edit permission or the user account that posted the review. Even with the right permission, users can not delete a review from somebody else if it was posted on an add-on they are listed as a developer of.

DELETE /api/v3/reviews/review/(int: *id*)/

Reply

This endpoint allows you to reply to an existing user review. If successful a *review reply object* is returned.

> **Note:** Requires authentication and either Addons:Edit permission or a user account listed as a developer of the add-on.

POST /api/v3/reviews/review/(int: *id*)/reply/

Request JSON Object

- **body** (*string*) – The text of the reply (required).
- **title** (*string|null*) – The title of the reply.

Flag

This endpoint allows you to flag an existing user review, to let a moderator know that something may be wrong with it.

An empty response will be returned on success.

> **Note:** Requires authentication and a user account different from the one that posted the review.

POST /api/v3/reviews/review/(int: *id*)/flag/

Request JSON Object

- **flag** (*string*) – A *constant* describing the reason behind the flagging.
- **note** (*string|null*) – A note to explain further the reason behind the flagging. This field is required if the flag is `review_flag_reason_other`, and passing it will automatically change the flag to that value.

Available constants for the `flag` property:

Constant	Description
review_flag_reason_spam	Spam or otherwise non-review content
review_flag_reason_language	Inappropriate language/dialog
review_flag_reason_bug_support	Misplaced bug report or support request
review_flag_reason_other	Other (please specify)

1.2.13 Signing

Note: This API requires *authentication*.

The following API endpoints help you get your add-on signed by Mozilla so it can be installed into Firefox without error. See extension signing for more details about Firefox's signing policy.

Client Libraries

The following libraries will make it easier to use the signing API:

- sign-addon, for general programattic use in NodeJS
- web-ext sign, for developing Web Extensions

If you are using `curl` to interact with the API you should be sure to pass the `-g` flag to skip "URL globbing" which won't interact well with add-on Ids that have { } characters in them.

Uploading a version

You can upload a new version for signing by issuing a `PUT` request and including the contents of your add-on in the `upload` parameter as multi-part formdata. This will create a pending version on the add-on and will prevent future submissions to this version unless validation or review fails.

If the upload succeeded then it will be submitted for validation and you will be able to check its status.

`PUT /api/v3/addons/[string:addon-id]/versions/[string:version]/`

Request:

```
curl "https://addons.mozilla.org/api/v3/addons/@my-addon/versions/1.0/"
    -g -XPUT --form "upload=@build/my-addon.xpi"
    -H "Authorization: JWT <jwt-token>"
```

Parameters

- **`addon-id`** – The id for the add-on.
- **`version`** – The version of the add-on. A version ending with `a`, `alpha`, `b`, or `beta` and an optional number is detected as beta. For example: `2.0-beta1` or `1.2a`.
- **`channel`** – (optional) The version channel this version should uploaded to, which determines its visibility on the site. Can be either `unlisted` or `listed`. See note below.

Form Parameters

- **`upload`** – The add-on file being uploaded.

Request Headers

- Content-Type – multipart/form-data

Note: `channel` is only valid for new versions on existing add-ons - if the add-on is new the version will be created as `unlisted`. If the parameter isn't supplied then the channel of the most recent version (submitted either via this API or the website) will assumed. (e.g. if you submit a version as `listed` then the next version will be `listed`, if you don't specify the channel).

Response:

The response body will be the same as the *Checking the status of your upload* response.

Status Codes

- 201 Created – new add-on and version created.
- 202 Accepted – new version created.
- 400 Bad Request – an error occurred, check the *error* value in the JSON.
- 401 Unauthorized – authentication failed.
- 403 Forbidden – you do not own this add-on.
- 409 Conflict – version already exists.

Uploading without an ID

Note: This is only valid for WebExtensions. All other add-on types require an add-on ID and have to use the regular endpoint to *upload a version*.

`POST /api/v3/addons/`

Request:

```
curl "https://addons.mozilla.org/api/v3/addons/"
    -g -XPOST -F "upload=@build/my-addon.xpi" -F "version=1.0"
    -H "Authorization: JWT <jwt-token>"
```

Form Parameters

- **`upload`** – The add-on file being uploaded.
- **`version`** – The version of the add-on. A version ending with `a`, `alpha`, `b`, or `beta` and an optional number is detected as beta. For example: `2.0-beta1` or `1.2a`.

Request Headers

- Content-Type – multipart/form-data

Response:

The response body will be the same as the *Checking the status of your upload* response.

Status Codes

- 201 Created – new add-on and version created.
- 202 Accepted – new version created.

- 400 Bad Request – an error occurred, check the *error* value in the JSON.
- 401 Unauthorized – authentication failed.
- 403 Forbidden – you do not own this add-on.
- 409 Conflict – version already exists.

Creating an add-on

If this is the first time that your add-on's UUID has been seen then the add-on will be created as an unlisted add-on when the version is uploaded.

Checking the status of your upload

You can check the status of your upload by issuing a `GET` request. There are a few things that will happen once a version is uploaded and the status of those events is included in the response.

Once validation is completed (whether it passes or fails) then the `processed` property will be `true`. You can check if validation passed using the `valid` property and check the results with `validation_results`.

If validation passed then your add-on will be submitted for review. In the case of unlisted add-ons this will happen automatically. If your add-on is listed then it will be reviewed by a human and that will take a bit longer. You can check the `automated_signing` property to see if signing will happen automatically or after a manual review. Once review is complete then the `reviewed` property will be set and you can check the results with the `passed_review` property.

GET /api/v3/addons/[string:addon-id]/versions/[string:version]/(uploads/[string:upload-pk],

Request:

```
curl "https://addons.mozilla.org/api/v3/addons/@my-addon/versions/1.0/"
    -g -H "Authorization: JWT <jwt-token>"
```

Parameters

- **addon-id** – the id for the add-on.
- **version** – the version of the add-on.
- **upload-pk** – (optional) the pk for a specific upload.

Response:

```
{
    "guid": "420854ee-7a85-42b9-822f-8e03dc5f6de9",
    "active": true,
    "automated_signing": true,
    "files": [
        {
            "download_url": "https://addons.mozilla.org/api/v3/downloads/file/100/
↪example-id.0-fx+an.xpi?src=api",
            "hash":
↪"sha256:1bb945266bf370170a656350d9b640cbcaf70e671cf753c410e604219cdd9267",
            "signed": true
        }
    ],
    "passed_review": true,
    "pk": "f68abbb3b1624c098fe979a409fe3ce9",
```

```
    "processed": true,
    "reviewed": true,
    "url": "https://addons.mozilla.org/api/v3/addons/@example-id.0/uploads/
↪f68abbb3b1624c098fe979a409fe3ce9/",
    "valid": true,
    "validation_results": {},
    "validation_url": "https://addons.mozilla.org/en-US/developers/upload/
↪f68abbb3b1624c098fe979a409fe3ce9",
    "version": "1.0"
}
```

Response JSON Object

- **guid** – The GUID of the addon.
- **active** – version is active.
- **automated_signing** – If true, the version will be signed automatically. If false it will end up in the manual review queue when valid.
- **files[].download_url** – URL to *download the add-on file*.
- **files[].hash** – Hash of the file contents, prefixed by the hashing algorithm used. Example: `sha256:1bb945266bf3701...`. In the case of signed files, the hash will be that of the final signed file, not the original unsigned file.
- **files[].signed** – if the file is signed.
- **passed_review** – if the version has passed review.
- **pk** – the pk for this upload.
- **processed** – if the version has been processed by the validator.
- **reviewed** – if the version has been reviewed.
- **url** – URL to check the status of this upload.
- **valid** – if the version passed validation.
- **validation_results** – the validation results (removed from the example for brevity).
- **validation_url** – a URL to the validation results in HTML format.
- **version** – the version.

Status Codes

- 200 OK – request successful.
- 401 Unauthorized – authentication failed.
- 403 Forbidden – you do not own this add-on.
- 404 Not Found – add-on or version not found.

Downloading signed files

When checking on your *request to sign a version*, a successful response will give you an API URL to download the signed files. This endpoint returns the actual file data for download.

GET /api/v3/file/[int:file_id]/[string:base_filename]

Request:

```
curl "https://addons.mozilla.org/api/v3/file/123/some-addon.xpi?src=api"
    -g -H "Authorization: JWT <jwt-token>"
```

Parameters

- **file_id** – the primary key of the add-on file.
- **base_filename** – the base filename. This is just a convenience for clients so that they write meaningful file names to disk.

Response:

There are two possible responses:

- Binary data containing the file
- A header that redirects you to a mirror URL for the file. In this case, the initial response will include a `SHA-256` hash of the file in the header `X-Target-Digest`. Clients should check that the final downloaded file matches this hash.

Status Codes

- 200 OK – request successful.
- 302 Found – file resides at a mirror URL
- 401 Unauthorized – authentication failed.
- 404 Not Found – file does not exist or requester does not have access to it.

1.2.14 Statistics

Note: This API requires *authentication*.

However this does not affect the permission model for addons. If you enabled public stats the data is accessible to everyone otherwise only authors are allowed to access the data.

The following API endpoints help you get your archived addon statistics.

Archiving

We are archiving statistics data that is older than one year.

We currently archive the following data:

name	model_name	description
update counts	updatecounts	How many users updated this addon
download counts	downloadcounts	How many users have this addon installed
theme update counts	themeupdatecount	How many users have this theme installed

List monthly archived data

The archive is structured by year/month, to see what data is archived for a specific month use the following api:

GET /api/v3/statistics/archive/[string:addon-slug]/[string:year]/[string:month]/
Request:

```
curl "https://addons.mozilla.org/api/v3/statistics/archive/my-addon/2016/01/"
    -H "Authorization: JWT <jwt-token>"
```

Parameters

- **addon-slug** – The slug for the add-on.
- **year** – The year you want to fetch.
- **month** – The month you want to fetch, please make sure to use 2 char months, e.g 01 instead of 1.

Response:

```
[
    {
        "date": "2016-01-18",
        "addon_id": 3615,
        "model_name": "themeupdatecount",
    }
]
```

Response JSON Object

- **date** – The full date for the data.
- **addon_id** – The addon-id for the data.
- **model_name** – The type of data.

Status Codes

- 200 OK – Archived data exists, you'll get the list of data-points.
- 401 Unauthorized – Authentication failed.
- 403 Forbidden – You do not own this add-on.
- 404 Not Found – No data exists for this query

Get archived data points

Now that you have an overview of what data exists, use the following api to access the actual data points for a specific model and date.

GET /api/v3/statistics/archive/[string:addon-slug]/[string:year]/[string:month]/[string:day
Request:

```
curl "https://addons.mozilla.org/api/v3/statistics/archive/my-addon/2016/01/18/
↪themeupdatecount/"
    -H "Authorization: JWT <jwt-token>"
```

Parameters

- **addon-slug** – The slug for the add-on.
- **year** – The year you want to fetch.
- **month** – The month you want to fetch, please make sure to use 2 char months, e.g 01 instead of 1.
- **day** – The day you want to fetch, please make sure to use 2 char months, e.g 01 instead of 1.

Response:

```
[
    {
        "date": "2016-01-18",
        "count": 123,
        "addon": 3615
    }
]
```

Response JSON Object

- **date** – The full date for the data.
- **count** – The actual statistics data.
- **addon** – The addon id, can be used to relate and group data.

Status Codes

- 200 OK – Archived data exists, you'll get the data.
- 401 Unauthorized – Authentication failed.
- 403 Forbidden – You do not own this add-on.
- 404 Not Found – No data exists for this query

1.2.15 GitHub Webhooks

Note: This is an Experimental API and can change at any time.

This API provides an endpoint that works with GitHub to provide add-on validation as a GitHub webhook. This end point is designed to be called specifically from GitHub and will only send API responses back to *api.github.com*.

To set this up on a GitHub repository you will need to:

- Go to *Settings* > *Webhooks & Services*
- Add a new Webhook with Payload URL of *https://addons.mozilla.org/api/v3/github/validate/*
- Click *Send me everything*
- Click *Update webhook*

At this point the validator will be able to get the data, but won't be able to write a response to GitHub. To enable responses to GitHub:

- Go to *Settings* > *Collaborators*
- Enter *addons-robot* and select the entry

- Click *Add collaborator*
- You will have to wait for a Mozilla person to respond to the invite

If this service proves useful and this service transitions from its Experimental API state, we will remove as many of these steps as possible.

The validator will run when you create or alter a pull request.

POST /api/v3/github/validate/

Request:

A GitHub API webhook body. Currently only *pull_request* events are processed, all others are ignored.

Response:

Status Codes

- 201 Created – request has been processed and a pending message sent back to GitHub.
- 200 OK – request is not a *pull_request*, it's been accepted.
- 422 Unprocessable Entity – body is invalid.

1.3 Server Install

The following documentation covers how to install and develop the server using Docker.

1.3.1 Install with Docker

Want the easiest way to start contributing to AMO? Try our Docker-based development environment.

First you'll need to install Docker. Please read their docs for the installation steps specific to your operating system.

There are two options for running docker depending on the platform you are running.

- Run docker on the host machine directly (recommended)
- Run docker-machine which will run docker inside a virtual-machine

Historically Mac and Windows could only run Docker via a vm. That has recently changed with the arrival of docker-for-mac and docker-for-windows.

If your platform can run Docker directly either on Linux, with docker-for-mac or docker-for-windows then this is the easiest way to run `addons-server`.

If you have problems, due to not meeting the minimum specifications for docker-for-windows or you'd prefer to keep running docker-machine vms then docker-machine will still work just fine. See the docs for creating the vm here *Creating the docker-machine vm*

Note: If you're on a Mac and already have a working docker-machine setup you can run that and docker-for-mac (*but not docker-for-windows*) side by side. The only caveat is it's recommended that you keep the versions of Docker on the vm and the host in-sync to ensure compatibility when you switch between them.

Setting up the containers

Note: docker-toolbox, docker-for-mac and docker-for-windows will install `docker-compose` for you. If you're on Linux and you need it, you can install it manually with:

```
pip install docker-compose
```

Next once you have Docker up and running follow these steps on your host machine:

```
# Checkout the addons-server sourcecode.
git clone git://github.com/mozilla/addons-server.git
cd addons-server
# Download the containers
docker-compose pull  # Can take a while depending on your internet bandwidth.
# Start up the containers
docker-compose up -d
make initialize_docker  # Answer yes, and create your superuser when asked.
# On Windows you can substitute `make initialize_docker` for the command:
docker-compose exec web make initialize
```

Note: Docker requires the code checkout to exist within your home directory so that Docker can mount the source-code into the container.

Accessing the web server

By default our docker-compose config exposes the web-server on port 80 of localhost.

We use `olympia.dev` as the default hostname to access your container server (e.g. for Firefox Accounts). To be able access the development environment using `http://olympia.dev` you'll need to edit your `/etc/hosts` file on your native operating system. For example:

```
[ip-address]  olympia.dev
```

Typically the IP address is localhost (127.0.0.1) but if you're using docker-machine see *Accessing the web-server with docker-machine* for details of how to get the ip of the Docker vm.

By default we configure *OLYMPIA_SITE_URL* to point to *http://olympia.dev*.

If you choose a different hostname you'll need to set that environment variable and restart the Docker containers:

```
docker-compose stop # only needed if running
export OLYMPIA_SITE_URL=http://[YOUR_HOSTNAME}
docker-compose up -d
```

Running common commands

Run the tests using `make`, *outside* of the Docker container:

```
make test
# or
docker-compose exec web py.test src/olympia/
```

You can run commands inside the Docker container by sshing into it using:

```
make shell
# or
docker-compose exec web bash
```

Then to run the tests inside the Docker container you can run:

```
py.test
```

You can also run single commands from your host machine without opening a shell on each container. Here is an example of running the `py.test` command on the `web` container:

```
docker-compose run web py.test
```

If you'd like to use a python debugger to interactively debug Django view code, check out the *Debugging* section.

Note: If you see an error like `No such container: addonsserver_web_1` and your containers are running you can overwrite the base name for docker containers with the `COMPOSE_PROJECT_NAME` environment variable. If your container is named `localaddons_web_1` you would set `COMPOSE_PROJECT_NAME=localaddons`.

Updating your containers

Any time you update Olympia (e.g., by running `git pull`), you should make sure to update your Docker image and database with any new requirements or migrations:

```
docker-compose stop
docker-compose pull
docker-compose up -d
make update_docker  # Runs database migrations and rebuilds assets.
# On Windows you can substitute `make update_docker` for the following two commands:
docker-compose exec worker make update_deps
docker-compose exec web make update
```

Gotchas!

Here's a list of a few of the issues you might face when using Docker.

Can't access the web server?

Check you've created a hosts file entry pointing `olympia.dev` to the relevant IP address.

If containers are failing to start use `docker-compose ps` to check their running status.

Another way to find out what's wrong is to run `docker-compose logs`.

Getting "Programming error [table] doesn't exist"?

Make sure you've run the `make initialize_docker` step as detailed in the initial setup instructions.

Port collisions (nginx container fails to start)

If you're already running a service on port 80 or 8000 on your host machine, the `nginx` container will fail to start. This is because the `docker-compose.override.yml` file tells `nginx` to listen on port 80 and the web service to listen on port 8000 by default.

This problem will manifest itself by the services failing to start. Here's an example for the most common case of `nginx` not starting due to a collision on port 80:

```
ERROR: for nginx  Cannot start service nginx:.....
...Error starting userland proxy: Bind for 0.0.0.0:80: unexpected error (Failure␣
↪EADDRINUSE)
ERROR: Encountered errors while bringing up the project.
```

You can check what's running on that port by using (sudo is required if you're looking at port < 1024):

```
sudo lsof -i :80
```

We specify the ports `nginx` listens on in the `docker-compose.override.yml` file. If you wish to override the ports you can do so by creating a new `docker-compose` config and starting the containers using that config alongside the default config.

For example if you create a file called `docker-compose-ports.yml`:

```
nginx:
  ports:
    - 8880:80
```

Next you would stop and start the containers with the following:

```
docker-compose stop # only needed if running
docker-compose -f docker-compose.yml -f docker-compose-ports.yml up -d
```

Now the container `nginx` is listening on 8880 on the host. You can now proxy to the container `nginx` from the host `nginx` with the following `nginx` config:

```
server {
    listen       80;
    server_name  olympia.dev;
    location / {
        proxy_pass   http://olympia.dev:8880;
    }
}
```

Persisting changes

Please note: any command that would result in files added or modified outside of the `addons-server` folder (e.g. modifying pip or npm dependencies) won't persist, and thus won't survive after the running container exits.

Note: If you need to persist any changes to the image, they should be carried out via the `Dockerfile`. Commits to master will result in the Dockerfile being rebuilt on the Docker hub.

Restarting docker-machine vms following a reboot

If you quit docker-machine, or restart your computer, docker-machine will need to start again using:

```
docker-machine start addons-dev
```

You'll then need to *export the variables* again, and start the services:

```
docker-compose up -d
```

Hacking on the Docker image

If you want to test out changes to the Olympia Docker image locally, use the normal Docker commands such as this to build a new image:

```
cd addons-server
docker build -t addons/addons-server .
docker-compose up -d
```

After you test your new image, commit to master and the image will be published to Docker Hub for other developers to use after they pull image changes.

1.3.2 Installation with Docker machine

Creating the docker-machine vm

Your first step is to create a vm - this step assumes we're using virtualbox as the driver:

```
docker-machine create --driver=virtualbox addons-dev
```

Then you can export the variables so that docker-compose can talk to the docker service. This command will tell you how to do that:

```
docker-machine env addons-dev
```

On a mac it's a case of running:

```
eval $(docker-machine env addons-dev)
```

Now you have the vm running you can follow the standard docker instructions: *Install with Docker*

Accessing the web-server with docker-machine

If you're using docker-machine, you can get the ip like so:

```
docker-machine ip addons-dev
```

Note: If you're still using boot2docker then the command is *boot2docker ip*. At this point you can look at installing docker-toolbox and migrating your old boot2docker vm across to running via docker-machine. See docker-toolbox for more info.

Now you can connect to port 80 of that ip address. Here's an example (your ip might be different):

```
http://192.168.99.100/
```

Note: `docker-machine` hands out IP addresses as each VM boots; your IP addresses can change over time. You can find out which IP is in use using `docker-machine ip [machine name]`

The following documentation covers how to install the individual components, this documentation is deprecated in favour of using Docker. This documentation may be out of date or incomplete.

1.3.3 Install Olympia manually (deprecated)

The approved installation is *via Docker*. The following pages might be helpful but are deprecated and may be out of date.

Installing Olympia the long way

Note: The following documentation is deprecated. The approved installation is *via Docker*.

The following instructions walk you through installing and configuring all required services from scratch.

We're going to use all the hottest tools to set up a nice environment. Skip steps at your own peril. Here we go!

Requirements

To get started, you'll need:

- Python 2.7 (2.7 -> 2.7.10)
- Node 0.10.x or higher
- MySQL
- ElasticSearch
- libxml2 (for building lxml, used in tests)

OS X and *Ubuntu* instructions follow.

There are a lot of advanced dependencies we're going to skip for a fast start. They have their own *section*.

If you're on a Linux distro that splits all its packages into `-dev` and normal stuff, make sure you're getting all those `-dev` packages.

On Ubuntu

The following command will install the required development files on Ubuntu or, if you're running a recent version, you can install them automatically:

```
sudo apt-get install python-dev python-virtualenv libxml2-dev libxslt1-dev␣
↪libmysqlclient-dev memcached libssl-dev swig openssl curl libjpeg-dev zlib1g-dev␣
↪libsasl2-dev nodejs nodejs-legacy
```

Note: As of writing, M2Crypto is only compatible with swig <=3.0.4 version's. So, if you encounter a libssl exception while running `make full_init`, you might have to downgrade swig to version <=3.0.4.

On OS X

The best solution for installing UNIX tools on OS X is Homebrew.

The following packages will get you set for olympia:

```
brew install python libxml2 mysql libmemcached openssl swig jpeg
```

Note: As of writing, M2Crypto is only compatible with swig <=3.0.4 version's. So, if you encounter a libssl exception while running `make full_init`, you might have to downgrade swig to version <=3.0.4.

MySQL

You'll probably need to *configure MySQL after install* (especially on Mac OS X) according to advanced installation.

See *Database* for creating and managing the database.

Elasticsearch

You'll need an Elasticsearch server up and running during the init script. See *Elasticsearch* for more instructions.

Use the Source

Grab olympia from github with:

```
git clone git://github.com/mozilla/olympia.git
cd olympia
```

`olympia.git` is all the source code. *Updating* is detailed later on.

virtualenv and virtualenvwrapper

virtualenv is a tool to create isolated Python environments. This will let you put all of Olympia's dependencies in a single directory rather than your global Python directory. For ultimate convenience, we'll also use virtualenvwrapper which adds commands to your shell.

Are you ready to bootstrap virtualenv and virtualenvwrapper? Since each shell setup is different, you can install everything you need and configure your shell using the virtualenv-burrito. Type this:

```
curl -sL https://raw.github.com/brainsik/virtualenv-burrito/master/virtualenv-burrito.
↪sh | $SHELL
```

Open a new shell to test it out. You should have the `workon` and `mkvirtualenv` commands.

virtualenvwrapper Hooks (optional)

virtualenvwrapper lets you run hooks when creating, activating, and deleting virtual environments. These hooks can change settings, the shell environment, or anything else you want to do from a shell script. For complete hook documentation, see http://www.doughellmann.com/docs/virtualenvwrapper/hooks.html.

You can find some lovely hooks to get started at http://gist.github.com/536998. The hook files should go in `$WORKON_HOME` (`$HOME/Envs` from above), and `premkvirtualenv` should be made executable.

Getting Packages

Now we're ready to go, so create an environment for olympia:

```
mkvirtualenv olympia
```

That creates a clean environment named olympia using your default python. You can get out of the environment by restarting your shell or calling `deactivate`.

To get back into the olympia environment later, type:

```
workon olympia  # requires virtualenvwrapper
```

Note: Olympia requires Python 2.7.

Note: If you want to use a different Python binary, pass the name (if it is on your path) or the full path to mkvirtualenv with `--python`:

```
mkvirtualenv --python=/usr/local/bin/python2.7 olympia
```

Finish the install

First make sure you have a recent pip for security reasons:

```
pip install --upgrade pip
```

From inside your activated virtualenv, install the required python packages, initialize the database, create a super user, compress the assets, . . . :

```
make full_init
```

Settings

Most of olympia is already configured in `settings.py`, but there's some things you may want to configure locally. All your local settings go into `local_settings.py`. The settings template for developers, included below, is at docs/settings/local_settings.dev.py.

```
from settings import *  # noqa

INTERNAL_IPS = ('127.0.0.1',)
```

I'm extending `INSTALLED_APPS` and `MIDDLEWARE_CLASSES` to include the Django Debug Toolbar. It's awesome, you want it.

The file `local_settings.py` is for local use only; it will be ignored by git.

Database

By default, Olympia connects to the `olympia` database running on `localhost` as the user `root`, with no password. To create a database, run:

```
$ mysql -u root -p
mysql> CREATE DATABASE olympia CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

If you want to change settings, you can either add the database settings in your *local_settings.py* or set the environment variable `DATABASE_URL`:

```
export DATABASE_URL=mysql://<user>:<password>@<hostname>/<database>
```

If you've changed the user and password information, you need to grant permissions to the new user:

```
$ mysql -u root -p
mysql> GRANT ALL ON olympia.* TO <YOUR_USER>@localhost IDENTIFIED BY '<YOUR_PASSWORD>
↪';
```

Finally, to run the test suite, you'll need to add an extra grant in MySQL for your database user:

```
$ mysql -u root -p
mysql> GRANT ALL ON test_olympia.* TO <YOUR_USER>@localhost IDENTIFIED BY '<YOUR_
↪PASSWORD>';
```

Warning: Don't forget to change `<YOUR_USER>` and `<YOUR_PASSWORD>` to your actual database credentials.

The database is initialized automatically using the `make full_init` command you saw earlier.

Database Migrations

Each incremental change we add to the database is done with a versioned SQL (and sometimes Python) file. To keep your local DB fresh and up to date, run migrations like this:

```
$ schematic migrations/
```

If, at some point, you want to start from scratch and recreate the database, you can just run the `make initialize_db` command. This will also fake all the schematic migrations, and allow you to create a superuser.

Run the Server

If you've gotten the system requirements, downloaded `olympia`, set up your virtualenv with the compiled packages, and configured your settings and database, you're good to go.

```
./manage.py runserver
```

Note: If you don't have a LESS compiler already installed, opening http://localhost:8000 in your browser will raise a 500 server error. If you don't want to run through the *Manual installation* documentation just right now, you can disable all LESS pre-processing by adding the following line to your `local_settings.py` file:

```
LESS_PREPROCESS = False
```

Be aware, however, that this will make the site VERY slow, as a huge amount of LESS files will be served to your browser on EACH request, and each of those will be compiled on the fly by the LESS javascript compiler.

Create an Admin User

To connect to the site, you first need to register a new user "the standard way" by filling in the registration form.

Once this is done, you can either activate this user using the link in the confirmation email sent (it's displayed in the console, check your server logs), or use the following handy management command:

```
./manage.py activate_user <email of your user>
```

If you want to grant yourself admin privileges, pass in the `--set-admin` option:

```
./manage.py activate_user --set-admin <email of your user>
```

Updating

To run a full update of olympia (including source files, pip requirements and database migrations):

```
make full_update
```

If you want to do it manually, then check the Makefile.

The *Contributing* page has more on managing branches.

Contact

Come talk to us on irc://irc.mozilla.org/amo if you have questions, issues, or compliments.

Submitting a Patch

See the *Contributing* page.

Advanced Installation

In production we use things like memcached, rabbitmq + celery, elasticsearch, LESS, and Stylus. Learn more about installing these on the *Manual installation* page.

Note: Although we make an effort to keep advanced items as optional installs you might need to install some components in order to run tests or start up the development server.

Manual installation

Note: The following documentation is deprecated. The approved installation is *via Docker*.

Getting Fancy

MySQL

On your dev machine, MySQL probably needs some tweaks. Locate your my.cnf (or create one) then, at the very least, make UTF8 the default encoding:

```
[mysqld]
character-set-server=utf8
```

Here are some other helpful settings:

```
[mysqld]
default-storage-engine=innodb
character-set-server=utf8
skip-sync-frm=OFF
innodb_file_per_table
```

On Mac OS X with homebrew, put my.cnf in `/usr/local/Cellar/mysql/5.5.15/my.cnf` then restart like:

```
launchctl unload -w ~/Library/LaunchAgents/com.mysql.mysqld.plist
launchctl load -w ~/Library/LaunchAgents/com.mysql.mysqld.plist
```

Note: some of the options above were renamed between MySQL versions

Here are more tips for optimizing MySQL on your dev machine.

Memcached

We slipped this in with the basic install. The package was `memcached` on Ubuntu and `libmemcached` on OS X. Your default settings already use the following, so you shouldn't need to change anything:

```
CACHES = {
    'default': {
        'BACKEND': 'caching.backends.memcached.MemcachedCache',
        'LOCATION': ['localhost:11211'],
        'TIMEOUT': 500,
    }
}
```

RabbitMQ and Celery

See the *Celery* page for installation instructions. The *example settings* set `CELERY_ALWAYS_EAGER = True`. If you're setting up Rabbit and want to use `celery`, make sure you remove that line from your `local_settings.py`.

Elasticsearch

See *Elasticsearch* for more instructions.

Redis

On OS X the package is called `redis`. Get it running with the `launchctl` script included in homebrew. To let olympia know about Redis, add this to `local_settings.py`:

```
CACHE_MACHINE_USE_REDIS = True
REDIS_BACKEND = 'redis://'
```

The `REDIS_BACKEND` is parsed like `CACHE_BACKEND` if you need something other than the default settings.

Node.js

Node.js is needed for Stylus and LESS, which in turn are needed to precompile the CSS files.

If you want to serve the CSS files from another domain than the webserver, you will need to precompile them. Otherwise you can have them compiled on the fly, using javascript in your browser, if you set `LESS_PREPROCESS = False` in your local settings.

First, we need to install node and npm:

```
brew install node
curl https://www.npmjs.org/install.sh | sh
```

Optionally make the local scripts available on your path if you don't already have this in your profile:

```
export PATH="./node_modules/.bin/:${PATH}"
```

Not working?

- If you're having trouble installing node, try http://shapeshed.com/journal/setting-up-nodejs-and-npm-on-mac-osx/. You need brew, which we used earlier.
- If you're having trouble with npm, check out the README on https://github.com/isaacs/npm

Stylus CSS

Learn about Stylus at http://learnboost.github.com/stylus/

```
cd olympia
npm install
```

In your `local_settings.py` ensure you are pointing to the correct executable for `stylus`:

```
STYLUS_BIN = path('node_modules/stylus/bin/stylus')
```

LESS CSS

We're slowing switching over from regular CSS to LESS. You can learn more about LESS at http://lesscss.org.

If you already ran `npm install` you don't need to do anything more.

In your `local_settings.py` ensure you are pointing to the correct executable for `less`:

```
LESS_BIN = path('node_modules/less/bin/lessc')
```

You can make the CSS live refresh on save by adding `#!watch` to the URL or by adding the following to your `local_settings.py`:

```
LESS_LIVE_REFRESH = True
```

If you want syntax highlighting, try:

- vim: http://leafo.net/lessphp/vim/
- emacs: http://jdhuntington.com/emacs/less-css-mode.el
- TextMate: https://github.com/appden/less.tmbundle
- Coda: http://groups.google.com/group/coda-users/browse_thread/thread/b3327b0cb893e439?pli=1

Generating additional add-ons

Note: If you previously used the `make full_init` command during the *Installing Olympia the long way* process, it's not necessary to generate additional add-ons for initialisation/development purpose.

If you need more add-ons, you can generate additional ones using the following command:

```
python manage.py generate_addons <num_addons> [--owner <email>] [--app <application>]
```

where `num_addons` is the number of add-ons that you want to generate, `email` (optional) is the email address of the owner of the generated add-ons and `application` (optional) the name of the application (either `firefox`, `thunderbird`, `seamonkey` or `android`).

By default the `email` will be `nobody@mozilla.org` and the `application` will be `firefox` if not specified.

Add-ons will have 1 preview image, 2 translations (French and Spanish), 5 ratings and might be featured randomly.

If you didn't run the `make full_init` command during the *Installing Olympia the long way* process, categories from production (Alerts & Updates, Appearance, and so on) will be created and randomly populated with generated add-ons. Otherwise, the existing categories will be filled with newly generated add-ons.

Celery

Note: The following documentation is deprecated. The approved installation is *via Docker*.

Celery is a task queue powered by RabbitMQ. You can use it for anything that doesn't need to complete in the current request-response cycle. Or use it wherever Les tells you to use it.

For example, each addon has a `current_version` cached property. This query on initial run causes strain on our database. We can create a denormalized database field called `current_version` on the `addons` table.

We'll need to populate regularly so it has fairly up-to-date data. We can do this in a process outside the request-response cycle. This is where Celery comes in.

Installation

RabbitMQ

Celery depends on RabbitMQ. If you use `homebrew` you can install this:

```
brew install rabbitmq
```

Setting up rabbitmq invovles some configuration. You may want to define the following

```
# On a Mac, you can find this in System Preferences > Sharing
export HOSTNAME='<laptop name>.local'
```

Then run the following commands:

```
# Set your host up so it's semi-permanent
sudo scutil --set HostName $HOSTNAME

# Update your hosts by either:
# 1) Manually editing /etc/hosts
# 2) `echo 127.0.0.1 $HOSTNAME >> /etc/hosts`

# RabbitMQ insists on writing to /var
sudo rabbitmq-server -detached

# Setup rabitty things (sudo is required to read the cookie file)
sudo rabbitmqctl add_user olympia olympia
sudo rabbitmqctl add_vhost olympia
sudo rabbitmqctl set_permissions -p olympia olympia ".*" ".*" ".*"
```

Back in safe and happy django-land you should be able to run:

```
celery -A olympia worker -E
```

Celery understands python and any tasks that you have defined in your app are now runnable asynchronously.

Celery Tasks

Any python function can be set as a celery task. For example, let's say we want to update our `current_version` but we don't care how quickly it happens, just that it happens. We can define it like so:

```
@task(rate_limit='2/m')
def _update_addons_current_version(data, **kw):
    task_log.debug("[%s@%s] Updating addons current_versions." %
                   (len(data), _update_addons_current_version.rate_limit))
    for pk in data:
        try:
            addon = Addon.objects.get(pk=pk)
            addon.update_version()
        except Addon.DoesNotExist:
            task_log.debug("Missing addon: %d" % pk)
```

`@task` is a decorator for Celery to find our tasks. We can specify a `rate_limit` like `2/m` which means `celery` will only run this command 2 times a minute at most. This keeps write-heavy tasks from killing your database.

If we run this command like so:

```
from celery.task.sets import TaskSet

all_pks = Addon.objects.all().values_list('pk', flat=True)
ts = [_update_addons_current_version.subtask(args=[pks])
      for pks in amo.utils.chunked(all_pks, 300)]
TaskSet(ts).apply_async()
```

All the Addons with ids in `pks` will (eventually) have their `current_versions` updated.

Cron Jobs

This is all good, but let's automate this. In Olympia we can create cron jobs like so:

```
@cronjobs.register
def update_addons_current_version():
    """Update the current_version field of the addons."""
    d = Addon.objects.valid().exclude(
          type=amo.ADDON_PERSONA).values_list('id', flat=True)

    with establish_connection() as conn:
        for chunk in chunked(d, 1000):
            print chunk
            _update_addons_current_version.apply_async(args=[chunk],
                                                       connection=conn)
```

This job will hit all the addons and run the task we defined in small batches of 1000.

We'll need to add this to both the `prod` and `preview` crontabs so that they can be run in production.

Better than Cron

Of course, cron is old school. We want to do better than cron, or at least not rely on brute force tactics.

For a surgical strike, we can call `_update_addons_current_version` any time we add a new version to that addon. Celery will execute it at the prescribed rate, and your data will be updated ... eventually.

During Development

`celery` only knows about code as it was defined at instantiation time. If you change your `@task` function, you'll need to `HUP` the process.

However, if you've got the `@task` running perfectly you can tweak all the code, including cron jobs that call it without need of restart.

Elasticsearch

Note: The following documentation is deprecated. The approved installation is *via Docker*.

Elasticsearch is a search server. Documents (key-values) get stored, configurable queries come in, Elasticsearch scores these documents, and returns the most relevant hits.

Also check out elasticsearch-head, a plugin with web front-end to elasticsearch that can be easier than talking to elasticsearch over curl, or Marvel, which includes a query editors with autocompletion.

Installation

Elasticsearch comes with most package managers.:

```
brew install elasticsearch  # or whatever your package manager is called.
```

If Elasticsearch isn't packaged for your system, you can install it manually, here are some good instructions on how to do so.

On Ubuntu, you should just download and install a .deb from the download page.

Launching and Setting Up

Launch the Elasticsearch service. If you used homebrew, `brew info elasticsearch` will show you the commands to launch. If you used aptitude, Elasticsearch will come with a start-stop daemon in /etc/init.d. On Ubuntu, if you have installed from a .deb, you can type:

    sudo service elasticsearch start

Olympia has commands that sets up mappings and indexes objects such as add-ons and apps for you. Setting up the mappings is analogous to defining the structure of a table, indexing is analogous to storing rows.

For AMO, this will set up all indexes and start the indexing processeses:

```
./manage.py reindex
```

Or you could use the makefile target:

```
make reindex
```

If you need to add arguments:

```
make ARGS='--with-stats --wipe --force' reindex
```

Indexing

Olympia has other indexing commands. It is worth noting that the index is maintained incrementally through post_save and post_delete hooks:

```
./manage.py cron reindex_addons  # Index all the add-ons.

./manage.py index_stats  # Index all the update and download counts.

./manage.py cron reindex_collections  # Index all the collections.

./manage.py cron reindex_users  # Index all the users.

./manage.py cron compatibility_report  # Set up the compatibility index.

./manage.py weekly_downloads # Index weekly downloads.
```

Querying Elasticsearch in Django

For now, we have our own query builder (which is an historical clone of elasticutils), but we will switch to the official one very soon.

We attach elasticutils to Django models with a mixin. This lets us do things like `.search()` which returns an object which acts a lot like Django's ORM's object manager. `.filter(**kwargs)` can be run on this search object:

```
query_results = list(
    MyModel.search().filter(my_field=a_str.lower())
    .values_dict('that_field'))
```

Testing with Elasticsearch

All test cases using Elasticsearch should inherit from `amo.tests.ESTestCase`. All such tests are marked with the `es_tests` pytest marker. To run only those tests:

```
py.test -m es_tests
```

or

```
make test_es
```

Troubleshooting

I got a CircularReference error on .search() - check that a whole object is not being passed into the filters, but rather just a field's value.

I indexed something into Elasticsearch, but my query returns nothing - check whether the query contains upper-case letters or hyphens. If so, try lowercasing your query filter. For hyphens, set the field's mapping to not be analyzed:

```
'my_field': {'type': 'string', 'index': 'not_analyzed'}
```

Try running .values_dict on the query as mentioned above.

Trouble-shooting the development installation

Note: The following documentation is deprecated. The approved installation is *via Docker*.

Image processing isn't working

If adding images to apps or extensions doesn't seem to work then there's a couple of settings you should check.

Checking your image settings

Check that `CELERY_ALWAYS_EAGER` is set to `True` in your settings file. This means it will process tasks without a celery worker running:

```
CELERY_ALWAYS_EAGER = True
```

If that yields no joy you can try running a celery worker in the foreground, set `CELERY_ALWAYS_EAGER = False` and run:

```
celery -A olympia worker -E
```

Note: This requires a RabbitMQ server to be set up as detailed in the *Celery* instructions.

This may help you to see where the image processing tasks are failing. For example it might show that Pillow is failing due to missing dependencies.

Checking your Pillow installation (Ubuntu)

When you run you should see at least JPEG and ZLIB are supported

If that's the case you should see this in the output of `pip install -I Pillow`:

```
--------------------------------------------------------------------
*** TKINTER support not available
--- JPEG support available
--- ZLIB (PNG/ZIP) support available
*** FREETYPE2 support not available
*** LITTLECMS support not available
--------------------------------------------------------------------
```

If you don't then this suggests Pillow can't find your image libraries:

To fix this double-check you have the necessary development libraries installed first (e.g: `sudo apt-get install libjpeg-dev zlib1g-dev`)

Now run the following for 32bit:

```
sudo ln -s /usr/lib/i386-linux-gnu/libz.so /usr/lib
sudo ln -s /usr/lib/i386-linux-gnu/libjpeg.so /usr/lib
```

Or this if your running 64bit:

```
sudo ln -s /usr/lib/x86_64-linux-gnu/libz.so /usr/lib
sudo ln -s /usr/lib/x86_64-linux-gnu/libjpeg.so /usr/lib
```

Note: If you don't know what arch you are running run `uname -m` if the output is `x86_64` then it's 64-bit, otherwise it's 32bit e.g. `i686`

Now re-install Pillow:

```
pip install -I Pillow
```

And you should see the necessary image libraries are now working with Pillow correctly.

ES is timing out

This can be caused by `number_of_replicas` not being set to 0 for the local indexes.

To check the settings run:

```
curl -s localhost:9200/_cluster/state\?pretty | fgrep number_of_replicas -B 5
```

If you see any that aren't 0 do the following:

Set `ES_DEFAULT_NUM_REPLICAS` to `0` in your local settings.

To set them to zero immediately run:

```
curl -XPUT localhost:9200/_settings -d '{ "index" : { "number_of_replicas" : 0 } }'
```

1.4 Development

1.4.1 Running Tests

Run tests from outside the docker container with `make`:

```
make test
```

Other, more niche, test commands:

```
make test_failed # rerun the failed tests from the previous run
make test_force_db # run the entire test suite with a new database
make tdd # run the entire test suite, but stop on the first error
```

Using py.test directly

For advanced users. To run the entire test suite you never need to use `py.test` directly.

You can connect to the docker container using `make shell`; then use `py.test` directly, which allows for finer-grained control of the test suite.

Run your tests like this:

```
py.test
```

For running subsets of the entire test suite, you can specify which tests run using different methods:

- *py.test -m es_tests* to run the tests that are marked as *es_tests*
- *py.test -k test_no_license* to run all the tests that have *test_no_license* in their name
- *py.test src/olympia/addons/tests/test_views.py::TestLicensePage::test_no_license* to run this specific test

For more, see the Pytest usage documentation.

1.4.2 Debugging

The *docker setup* uses supervisord to run the django runserver. This means if you want to access the management server from a shell to run things like ipdb you still can.

Using ipdb

As with ipdb normally just add a line in your code at the relevant point:

```
import ipdb; ipdb.set_trace()
```

Next connect to the running web container:

```
make debug
```

This will bring the Django management server to the foreground and you can interact with ipdb as you would normally. To quit you can just type `Ctrl+c`.

All being well it should look like this:

```
$ make debug
docker exec -t -i olympia_web_1 supervisorctl fg olympia
:/opt/rh/python27/root/usr/lib/python2.7/site-packages/celery/utils/__init__.py:93
11:02:08 py.warnings:WARNING /opt/rh/python27/root/usr/lib/python2.7/site-packages/
↪jwt/api_jws.py:118: DeprecationWarning: The verify parameter is deprecated. Please␣
↪use options instead.
'Please use options instead.', DeprecationWarning)
:/opt/rh/python27/root/usr/lib/python2.7/site-packages/jwt/api_jws.py:118
[21/Oct/2015 11:02:08] "PUT /en-US/firefox/api/v3/addons/%40unlisted/versions/0.0.5/␣
↪HTTP/1.1" 400 36
Validating models...

0 errors found
October 21, 2015 - 13:52:07
Django version 1.6.11, using settings 'settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.
[21/Oct/2015 13:57:56] "GET /static/img/app-icons/16/sprite.png HTTP/1.1" 200 3810
13:58:01 py.warnings:WARNING /opt/rh/python27/root/usr/lib/python2.7/site-packages/
↪celery/task/sets.py:23: CDeprecationWarning:
    celery.task.sets and TaskSet is deprecated and scheduled for removal in
    version 4.0. Please use "group" instead (see the Canvas section in the userguide)

""")
:/opt/rh/python27/root/usr/lib/python2.7/site-packages/celery/utils/__init__.py:93
> /code/src/olympia/browse/views.py(148)themes()
    147     import ipdb;ipdb.set_trace()
--> 148     TYPE = amo.ADDON_THEME
    149     if category is not None:

ipdb> n
> /code/src/olympia/browse/views.py(149)themes()
    148     TYPE = amo.ADDON_THEME
--> 149     if category is not None:
    150         q = Category.objects.filter(application=request.APP.id, type=TYPE)

ipdb>
```

Logging

Logs for the celery and Django processes can be found on your machine in the *logs* directory.

Using the Django Debug Toolbar

The Django Debug Toolbar is very powerful and useful when viewing pages from the website, to check the view used, its parameters, the SQL queries, the templates rendered and their context.

To use it please see the official getting started docs: https://django-debug-toolbar.readthedocs.io/en/1.4/installation.html#quick-setup

---

Note: You must know that using the Django Debug Toolbar will slow the website quite a lot. You can mitigate this by deselecting the checkbox next to the `SQL` panel.

Also, please note that you should only use the Django Debug Toolbar if you need it, as it makes CSP report only for your local dev.

Note: You might have to disable CSP by setting *CSP_REPORT_ONLY = True* in your local settings because django debug toolbar uses "data:" for its logo, and it uses "unsafe eval" for some panels like the templates or SQL ones.

1.4.3 Adding Python Dependencies

To add a new dependency you'll to carry out the following. First install hashin:

```
pip install hashin
```

Next add the dependency you want to add to the relevant requirements file.

Note: If you add just the package name the script will automatically get the latest version for you.

Once you've done that you can run the requirements script:

```
./scripts/hash_requirements.py <requirement-file>
```

This will add hashes and sort the requirements for you adding comments to show any package dependencies.

When it's run check the diff and make edits to fix any issues before submitting a PR with the additions.

1.4.4 Error Pages

When running Django locally you get verbose error pages instead of the standard ones. To access the HTML for the standard error pages, you can access the urls:

```
/services/403
/services/404
/services/500
```

1.4.5 Testing

We're using a mix of Django's Unit Testing and pytest with pytest-django, and Selenium for our automated testing. This gives us a lot of power and flexibility to test all aspects of the site.

Selenium tests are in a separate Selenium repository.

Configuration

Configuration for your unit tests is handled automatically. The only thing you'll need to ensure is that the database credentials in your settings has full permissions to modify a database with `test_` prepended to it. By default the database name is `olympia`, so the test database is `test_olympia`. Optionally, in particular if the code you are working on is related to search, you'll want to run Elasticsearch tests. Obviously, you need Elasticsearch to be installed. See *Elasticsearch* page for details.

If you don't want to run the Elasticsearch tests, you can use the `test_no_es` target in the Makefile:

```
make test_no_es
```

On the contrary, if you only want to run Elasticsearch tests, use the `test_es` target:

```
make test_es
```

Running Tests

To run the whole test suite use:

```
py.test
```

There are a lot of options you can pass to adjust the output. Read pytest and pytest-django docs for the full set, but some common ones are:

- `-v` to provide more verbose information about the test run
- `-s` tells pytest to not capture the logging output
- `--create-db` tells pytest-django to recreate the database instead of reusing the one from the previous run
- `-x --pdb` to stop on the first failure, and drop into a python debugger
- `--lf` to re-run the last test failed
- `-m test_es` will only run tests that are marked with the `test_es` mark
- `-k foobar` will only run tests that contain `foobar` in their name

There are a few useful makefile targets that you can use:

Run all the tests:

```
make test
```

If you need to rebuild the database:

```
make test_force_db
```

To fail and stop running tests on the first failure:

```
make tdd
```

If you wish to add arguments, or run a specific test, overload the variables (check the Makefile for more information):

```
make test ARGS='-v src/olympia/amo/tests/test_url_prefix.py::MiddlewareTest::test_get_
↪app'
```

If you wish to re-run only the tests failed from the previous run:

```
make test_failed
```

Database Setup

Our test runner is configured by default to reuse the database between each test run. If you really want to make a new database (e.g. when models have changed), use the `--create-db` parameter:

```
py.test --create-db
```

or

```
make test_force_db
```

Writing Tests

We support two types of automated tests right now and there are some details below but remember, if you're confused look at existing tests for examples.

Also, take some time to get familiar with pytest way of dealing with dependency injection, which they call fixtures (which should not be confused with Django's fixtures). They are very powerful, and can make your tests much more independent, cleaner, shorter, and more readable.

Unit/Functional Tests

Most tests are in this category. Our test classes extend `django.test.TestCase` and follow the standard rules for unit tests. We're using JSON fixtures for the data.

External calls

Connecting to remote services in tests is not recommended, developers should mock out those calls instead.

Why Tests Fail

Tests usually fail for one of two reasons: The code has changed or the data has changed. An third reason is **time**. Some tests have time-dependent data usually in the fixtures. For example, some featured items have expiration dates.

We can usually save our future-selves time by setting these expirations far in the future.

Localization Tests

If you want test that your localization works then you can add in locales in the test directory. For an example see `devhub/tests/locale`. These locales are not in the normal path so should not show up unless you add them to the *LOCALE_PATH*. If you change the .po files for these test locales, you will need to recompile the .mo files manually, for example:

```
msgfmt --check-format -o django.mo django.po
```

1.4.6 Style Guide

Writing code for olympia? Awesome! Please help keep our code readable by, whenever possible, adhering to these style conventions.

Python

- see https://wiki.mozilla.org/Webdev:Python

Markup

- `<!DOCTYPE html>`
- double-quote attributes
- Soft tab (2 space) indentation
- Title-Case `<label>` tags - “Display Name” vs “Display name”
- to clearfix, use the class `c` on an element

JavaScript

- Soft tabs (4 space) indentation
- Single quotes around strings (unless the string contains single quotes)
- variable names for jQuery objects start with $. for example:
  - `var $el = $(el);`
- Element IDs and classes that are not generated by Python should be separated by hyphens, eg: #some-module.
- Protect all functions and objects from global scope to avoid potential name collisions. When exposing functions to other scripts use the `z` namespace.
- Always protect code from loading on pages it’s not supposed to load on. For example:

```
$(document).ready(function() {
    if ($('#something-on-your-page').length) {
        initYourCode();
    }

    function initYourCode() {
        // ...
    }
});
```

1.4.7 Contributing

If you’re not sure this is the correct place to file an issue then please file an issue on the mozilla/addons project instead.

Before contributing code, please note:

- You agree to license your contributions under the license.
- Please ask on the dev-addons mailing list before submitting pull-requests for new features or large changes that are not related to existing issues.

- Follow PEP8, jshint and our other style guide conventions.
- Please write tests and read the docs on addons-server.

Ready to get started? Follow these steps.

Note to staff: If you come across a potential "good first bug" for contributors, please tag it with "**maybe good first bug**". The community team triages these every other week to ensure they have mentors assigned, onboarding information, and basic steps to get started. This gives new contributors a better experience when they pick a "good first bug" to work on.

Thank you for contributing!

1.4.8 Push From Master

We deploy from the master branch once a month. If you commit something to master that needs additional QA time, be sure to use a waffle feature flag.

Local Branches

Most new code is developed in local one-off branches, usually encompassing one or two patches to fix a bug. Upstream doesn't care how you do local development, but we don't want to see a merge commit every time you merge a single patch from a branch. Merge commits make for a noisy history, which is not fun to look at and can make it difficult to cherry-pick hotfixes to a release branch. We do like to see merge commits when you're committing a set of related patches from a feature branch. The rule of thumb is to rebase and use fast-forward merge for single patches or a branch of unrelated bug fixes, but to use a merge commit if you have multiple commits that form a cohesive unit.

Here are some tips on Using topic branches and interactive rebasing effectively.

1.4.9 Using the VPN with docker on OSX

If you need to access services behind a VPN, the docker setup should by default allow outgoing traffic over the VPN as it does for your host. If this isn't working you might find that it will work if you start up the vm *after* you have started the VPN connection.

To do this simply stop the containers:

```
docker-compose stop
```

Stop the docker-machine vm:

```
# Assumes you've called the vm 'addons-dev'
docker-machine stop addons-dev
```

Then connect to your VPN and restart the docker vm:

```
docker-machine start addons-dev
```

and fire up the env containers again:

```
docker-compose up -d
```

1.4.10 Access Control Lists

ACL versus Django Permissions

Currently we use the `is_superuser` flag in the `User` model to indicate that a user can access the admin site.

Outside of that we use the `access.models.GroupUser` to define what access groups a user is a part of.

How permissions work

Permissions that you can use as filters can be either explicit or general.

For example `Admin:EditAddons` means only someone with that permission will validate.

If you simply require that a user has *some* permission in the *Admin* group you can use `Admin:%`. The `%` means "any."

Similarly a user might be in a group that has explicit or general permissions. They may have `Admin:EditAddons` which means they can see things with that same permission, or things that require `Admin:%`.

If a user has a wildcard, they will have more permissions. For example, `Admin:*` means they have permission to see anything that begins with `Admin:`.

The notion of a superuser has a permission of `*:*` and therefore they can see everything.

1.4.11 Logging

Logging is fun. We all want to be lumberjacks. My muscle-memory wants to put `print` statements everywhere, but it's better to use `log.debug` instead. `print` statements make mod_wsgi sad, and they're not much use in production. Plus, `django-debug-toolbar` can hijack the logger and show all the log statements generated during the last request. When `DEBUG = True`, all logs will be printed to the development console where you started the server. In production, we're piping everything into `syslog`.

Configuration

The root logger is set up from `log_settings_base.py` in the `src/olympia/lib` of addons-server. It sets up sensible defaults, but you can twiddle with these settings:

LOG_LEVEL This setting is required, and defaults to `logging.DEBUG`, which will let just about anything pass through. To reconfigure, import logging in your settings file and pick a different level:

```
import logging
LOG_LEVEL = logging.WARN
```

USE_SYSLOG Set this to `True` if you want logging sent to syslog.

USE_MOZLOG Set this to `True` if you want logging sent to the console using mozlog format.

LOGGING See PEP 391 and log_settings.py for formatting help. Each section of LOGGING will get merged into the corresponding section of log_settings.py. Handlers and log levels are set up automatically based on LOG_LEVEL and DEBUG unless you set them here. Messages will not propagate through a logger unless propagate: True is set.

```
LOGGING = {
    'loggers': {
        'caching': {'handlers': ['null']},
    },
}
```

If you want to add more to this in `local_settings.py`, do something like this:

```
LOGGING['loggers'].update({
    'z.paypal': {
        'level': logging.DEBUG,
    },
    'z.es': {
        'handlers': ['null'],
    },
})
```

Using Loggers

The `olympia.core.logger` package uses global objects to make the same logging configuration available to all code loaded in the interpreter. Loggers are created in a pseudo-namespace structure, so app-level loggers can inherit settings from a root logger. olympia's root namespace is just `"z"`, in the interest of brevity. In the caching package, we create a logger that inherits the configuration by naming it `"z.caching"`:

```
import olympia.core.logger

log = olympia.core.logger.getLogger('z.caching')

log.debug("I'm in the caching package.")
```

Logs can be nested as much as you want. Maintaining log namespaces is useful because we can turn up the logging output for a particular section of olympia without becoming overwhelmed with logging from all other parts.

olympia.core.logging vs. logging

`olympia.core.logger.getLogger` should be used everywhere. It returns a `LoggingAdapter` that inserts the current user's IP address and username into the log message. For code that lives outside the request-response cycle, it will insert empty values, keeping the message formatting the same.

Complete logging docs: http://docs.python.org/library/logging.html

1.4.12 Services

Services contain a couple of scripts that are run as separate wsgi instances on the services. Usually they are hosted on separate domains. They are stand alone wsgi scripts. The goal is to avoid a whole pile of Django imports, middleware, sessions and so on that we really don't need.

To run the scripts you'll want a wsgi server. You can do this using gunicorn, for example:

```
pip install gunicorn
```

Then you can do:

```
cd services
gunicorn --log-level=DEBUG -c wsgi/receiptverify.py -b 127.0.0.1:9000 --debug␣
↪verify:application
```

To test:

```
curl -d "this is a bogus receipt" http://127.0.0.1:9000/verify/123
```

1.4.13 Translating Fields on Models

The `olympia.translations` app defines a `olympia.translations.models.Translation` model, but for the most part, you shouldn't have to use that directly. When you want to create a foreign key to the `translations` table, use `olympia.translations.fields.TranslatedField`. This subclasses Django's `django.db.models.ForeignKey` to make it work with our special handling of translation rows.

A minimal model with translations in addons-server would look like this:

```
from django.db import models

from olympia.amo.models import ModelBase
from olympia.translations.fields import TranslatedField, save_signal

class MyModel(ModelBase):
    description = TranslatedField()

models.signals.pre_save.connect(save_signal,
                                sender=MyModel,
                                dispatch_uid='mymodel_translations')
```

How it works behind the scenes

As mentioned above, a `TranslatedField` is actually a `ForeignKey` to the `translations` table. However, to support multiple languages, we use a special feature of MySQL allowing you to have a `ForeignKey` pointing to multiple rows.

When querying

Our base manager has a `_with_translations()` method that is automatically called when you instanciate a queryset. It does 2 things:

- Stick an extra lang=lang in the query to prevent query caching from returning objects in the wrong language
- Call `olympia.translations.transformers.get_trans()` which does the black magic.

`get_trans()` is called, and calls in turn `olympia.translations.transformer.build_query()` and builds a custom SQL query. This query is the heart of the magic. For each field, it setups a join on the translations table, trying to find a translation in the current language (using `olympia.translation.get_language()`) and then in the language returned by `get_fallback()` on the instance (for addons, that's `default_locale`; if the `get_fallback()` method doesn't exist, it will use `settings.LANGUAGE_CODE`, which should be `en-US` in addons-server).

Only those 2 languages are considered, and a double join + `IF` / `ELSE` is done every time, for each field.

This query is then ran on the slave (`get_trans()` gets a cursor using `connections[multidb.get_slave()]`) to fetch the translations, and some Translation objects are instantiated from the results and set on the instance(s) of the original query.

To complete the mechanism, `TranslationDescriptor.__get__` returns the `Translation`, and `Translations.__unicode__` returns the translated string as you'd expect, making the whole thing transparent.

When setting

Everytime you set a translated field to a string value, `TranslationDescriptor __set__` method is called. It determines which method to call (because you can also assign a dict with multiple translations in multiple languages at the same time). In this case, it calls `translation_from_string()` method, still on the "hidden" `TranslationDescriptor` instance. The current language is passed at this point, using `olympia.translation_utils.get_language()`.

From there, `translation_from_string()` figures out whether it's a new translation of a field we had no translation for, a new translation of a field we already had but in a new language, or an update to an existing translation.

It instantiates a new `Translation` object with the correct values if necessary, or just updates the correct one. It then places that object in a queue of Translation instances to be saved later.

When you eventually call `obj.save()`, the `pre_save` signal is sent. If you followed the example above, that means `olympia.translations.fields.save_signal` is then called, and it unqueues all Translation objects and saves them. It's important to do this on `pre_save` to prevent foreign key constraint errors.

When deleting

Deleting all translations for a field is done using `olympia.translations.models.delete_translation()`. It sets the field to `NULL` and then deletes all the attached translations.

Deleting a *specific* translation (like a translation in spanish, but keeping the english one intact) is implemented but not recommended at the moment. The reason why is twofold:

1. MySQL doesn't let you delete something that still has a FK pointing to it, even if there are other rows that match the FK. When you call `delete()` on a translation, if it was the last translation for that field, we set the FK to `NULL` and delete the translation normally. However, if there were any other translations, instead we temporarily disable the constraints to let you delete just the one you want.
2. Remember how fetching works? If you deleted a translation that is part of the fallback, then when you fetch that object, depending on your locale you'll get an empty string for that field, even if there are `Translation` objects in other languages available!

For additional discussion on this topic, see https://bugzilla.mozilla.org/show_bug.cgi?id=902435

Ordering by a translated field

`olympia.translations.query.order_by_translation` allows you to order a `QuerySet` by a translated field, honoring the current and fallback locales like it's done when querying.

1.4.14 Building Docs

To simply build the docs:

```
docker-compose run web make docs
```

If you're working on the docs, use `make loop` to keep your built pages up-to-date:

```
make shell
cd docs
make loop
```

Open `docs/_build/html/index.html` in a web browser.

1.4.15 Add-ons Server Documentation

This is the documentation for the use of the addons-server and its services. All documentation is in plain text files using reStructuredText and Sphinx.

To build the documentation, you need the dependencies from `requirements/docs.txt`. Those are automatically installed together with `make update_deps`, so if you've installed that already (following the *Installing Olympia the long way* page), you're all set.

If you're unsure, activate your `virtualenv` and run:

```
make update_deps
```

The documentation is viewable at http://addons-server.readthedocs.io/, and covers development using Add-ons Server, the source code for Add-ons.

Its source location is in the /docs folder.

Note: this project was once called *olympia*, this documentation often uses that term.

Build the documentation

This is as simple as running:

```
make docs
```

This is the same as `cd`'ing to the `docs` folder, and running `make html` from there.

We include a daemon that can watch and regenerate the built HTML when documentation source files change. To use it, go to the `docs` folder and run:

```
python watcher.py 'make html' $(find . -name '*.rst')
```

Once done, check the result by opening the following file in your browser:

    /path/to/olympia/docs/_build/html/index.html

1.5 Third-Party Usage

Running your own Add-ons server will likely require a few changes. There is currently no easy way to provide custom templates and since Firefox Accounts is used for authentication there is no way to authenticate a user outside of a Mozilla property.

If you would like to run your own Add-ons server you may want to update addons-server to support custom templates and move the Firefox Accounts management to a django authentication backend.

Another option would be to add any APIs that you required and write a custom frontend. This work is already underway and should be completed at some point but help is always welcome. You can find the API work in this project and the frontend work in addons-frontend.

HTTP Routing Table

/api

```
       11
PATCH /api/v3/accounts/account/(int:user_id|string:username)/collections/(string:collectio
       29
PATCH /api/v3/accounts/account/(int:user_id|string:username)/collections/(string:collectio
       31
PATCH /api/v3/reviews/review/(int:id)/,
       38
```