# CONTENTS

## 1 Repositories
1.1 mozilla-central .................................................. 3
1.2 GitHub ......................................................... 3

## 2 Browser ......................................................... 5

## 3 Localization (l10n) ............................................. 7
3.1 Projects ......................................................... 7
3.2 Routine Extraction of new strings ................................ 8
3.3 When can I expect to see my strings land? ..................... 9
3.4 Debug locales .................................................. 9

## 4 Server ........................................................ 11
4.1 Push Duty ...................................................... 11

## 5 UX .............................................................. 17
5.1 Add-ons Discovery Pane UX Specs ............................. 17
5.2 AMO to Firefox Account Migration UX Specs ................. 21
5.3 Management UX ............................................... 24
5.4 Submission UX ................................................ 25

## 6 Random stuff .................................................. 27
6.1 Github labels .................................................. 27

## 7 Indices and tables .............................................. 29
This refers primarily to the work done by the Add-ons team at Mozilla.

If you are an Add-on Developer wanting to work on developing an Add-on for Firefox, then you probably do not want to be looking at this documentation. Instead checkout MDN or AMO.

Contents:
1.1 mozilla-central

Browser code is maintained in mozilla-central. This includes several areas related to add-ons. All of the following bug trackers are in Bugzilla:

1.1.1 Add-ons Manager

Handles installing, running and updating add-ons within the application (eg: Firefox). Also has pages like about: addons. In bugzilla - product: Toolkit, component: Add-ons Manager. Existing bugs or file a new bug.

1.1.2 WebExtensions

An API for building add-ons that works with e10s and is compatible with Google Chrome. In bugzilla - product: Toolkit, component: WebExtensions. Existing bugs or file a new bug. Documentation on MDN.

1.2 GitHub

Almost everything else is on GitHub and issues are tracked in GitHub. This is a non-exhaustive list. Other repositories and libraries do appear around these main libraries:

1.2.1 addons

These docs and an issue tracker. This repository serves as an umbrella for everything add-ons. Bug tracker is in GitHub and can be used for almost anything add-ons related. Existing bugs.

1.2.2 addons-server

The addons.mozilla.org website. The repository and issue tracker is on GitHub. Documentation is on readthedocs. In the past this repository has been known as remora, zamboni or olympia.
1.2.3 addons-code-manager

A web application to manage add-on source code, such as reviewing code for add-ons submitted to addons.mozilla.org. The repository and issue tracker is on GitHub.

1.2.4 addons-linter

The linter checks WebExtensions for common errors and potential problems. It is used on addons.mozilla.org and web-ext. It can also be run in stand-alone mode. The repository, issue tracker and documentation is on GitHub.

1.2.5 dispensary

The dispensary collects and offers hashes of popular JavaScript libraries, mainly for the Mozilla’s addons-linter. The repository and issue tracker is on GitHub.

1.2.6 web-ext

This is a command line tool to help build, run, and test WebExtensions. The repository and issue tracker is on GitHub. The documentation and command reference is on Extension Workshop.

1.2.7 extension-workshop

Extension Workshop is a launchpad for building Firefox extensions. It contains resources about development, publication and management of WebExtensions. The repository and issue tracker is on GitHub.
Everything browser (and likely WebExtensions related).
These notes are primarily to support localizers working on localizations related to https://addons.mozilla.org/.

3.1 Projects

There are two core projects for AMO that require localization:

• addons-server (addons-server on Pontoon)
• addons-frontend (addons-frontend on Pontoon)

Both of these projects use Pontoon for translations.

3.1.1 addons-server

Addons-server is the default front-end for desktop users of AMO, it is also the API backend and home to the developer hub and reviewer tools. It is written in Python using the Django Framework.

Testing changes to addons-server localizations

The strings will be available on dev or stage depending on when they were committed see When can I expect to see my strings land? for details.

The relevant hosts for addons-server are as follows:

• dev https://addons-dev.allizom.org/
• stage https://addons.allizom.org/
• prod https://addons.mozilla.org/

3.1.2 addons-frontend

Addons-frontend is the new frontend and default frontend for mobile user-agents, it will eventually replace the addons-server frontend for user-pages. It is written in JavaScript and uses React at its core. Addons-frontend currently provides an amo app (amo.po).
Testing amo localizations

The strings will be available on dev or stage depending on when they were committed see *When can I expect to see my strings land?* for details.

The hosts for the new frontend are the same as addons-server except to be routed to the new frontend you either need:

- A “mamo” cookie with the value “on”. You can set this by clicking “View mobile site” in the footer.
- A Mobile user-agent string.

3.1.3 Advanced Configuration for testing addons-frontend strings

This will be enough for basic string checks. If you need to test strings related to the install buttons you will need to enabled preferences to allow add-on installation to work on the development and stage servers.

To fully configure Firefox to install add-ons on either dev/stage you will need to additionally do the following:

- Use Firefox Nightly or Firefox Developer Edition
- A new profile is recommended since changing the dev certs will mark existing add-on installs as invalid (this change is not immediate).
- Go to about:config and set extensions.webapi.testing as a Boolean to true.
- To install an add-on, you will need to be able to install a signed add-on from dev or stage. Create the xpinstall.signatures.dev-root pref in about:config as a Boolean and set it to true.
- If you see warnings when trying to install a theme or theme previews don’t work go to about:preferences#security and add an exception for dev/stage (See the “Exceptions” button next to “Warn me when sites try to install add-ons”).
- Restart the browser.

3.2 Routine Extraction of new strings

Newly added strings are extracted every week after the tag which is at 9am PST on Tuesdays.
3.3 When can I expect to see my strings land?

Once translations are committed into our repositories they are visible on our dev server within 30mins (or as long as it takes to deploy the code).

Master is tagged and pushed to stage weekly at around 9am PST every Tuesday.

The code is released to production every Thursday. The push generally starts around 9am PST.

3.4 Debug locales

There are two debug locales generated from the English keys. These can be useful to detect strings that are not marked for extraction.

The debug locales are called dbl (Debug left to right) and dbr (Debug right to left). The tool used to build these is called potools and it is both HTML and placeholder aware, to provide more accurate simulated localizations.

To access a debug locale simply replace the current locale in the url of the page you are on.

About the add-ons server.

Contents:

4.1 Push Duty

The pushing of the server rotates each week to another developer. Current rotation is:

- bsilverberg
- eviljeff
- mat

Check out the Add-ons calendar for a list of events.

4.1.1 Before the push

The code that will go in production on Thursday is tagged on Tuesday at 09:00 PT. The following repositories are tagged:

- addons-server
- addons-frontend
- addons-code-manager
- extension-workshop

Project Dependencies

Project dependencies are not tagged as part of the push duty responsibilities. If you’re working on a feature in a project that’s a dependency of a project e.g. addons-linter, then it’s your responsibility to make a release and update the project that consumes that dependency in time for the tag.

This way we can ensure that:

- Dependency packages are built and released in time for the tag.
- The new feature in the new version of a package has been validated on -dev.

Making multiple releases of a package during a weekly milestone is totally fine since this helps with testing smaller sets of changes.
Security Fixes

Security fixes for addons-server live on a separate private repository:

- addons-server-security

To make merging easier, when making a pull request against this repository, the remote branch should not be published to one’s fork but to the repository itself. Once the PR has been reviewed, it should not be merged right away. Instead, merging to main is part of push duty and happens right before tagging:

```bash
$ git checkout main
$ git pull
$ git fetch security
$ git merge security/<branch-name>
$ git diff upstream/main
$ git push upstream main
$ git push security main
```

**Note:** mozilla/addons-server-security main branch should never be pushed to directly without pushing to mozilla/addons-server main first - the two should always stay in sync.

This means the merge or edit buttons in github web UI must never be used in that repository.

**Note:** Here we are using `upstream` and `security` remotes, which point to mozilla/addons-server and mozilla/addons-server-security, respectively. If your configuration is different you can substitute `upstream` and `security` for whatever you call the mozilla/addons-server and mozilla/addons-server-security repositories' remotes.

Tag the repos

Tags are of the format: `YYYY.MM.DD`

**Note:** The date is the date of the push, not the date of tagging.

**Note:** Once addons-frontend has been tagged a new docker image will be built on CircleCI and is required to deploy to stage.

It’s usually the main branch that is tagged:

```bash
$ git checkout main
$ git pull
$ git tag 2015.09.10
$ git push upstream 2015.09.10
```

**Note:** Here we are using “upstream” as the remote. If yours is different you can substitute “upstream” for whatever you call the mozilla/addons-server repo remote.

Get a compare link from github to compare this tag to the last tag. Add that compare link to the push doc so that people can clearly see what is pushing.
If tagging the main branch can’t be done (some feature is already on main, but not ready for production), then the commits that need to be released should be cherry-picked.

If you’re adding cherry-picks to a tag that already exists, it makes sense to create a new tag rather than overwrite the old one. The reason for this is that re-using a tag makes it less easy to see the process that was involved in arriving at that tag. Also, it’s entirely possible to make a mistake by using an old tag that exists locally rather than the newer version on the remote when tags are re-used.

When creating a new tag you can use the format `YYYY.MM.DD-SUFFIX` where suffix is a number that’s incremented with each revision. The first time this is done will look like this:

```
$ git checkout 2015.09.03
$ git cherry-pick <commit hash> # as many times as you need
$ git tag 2015.09.10-1
$ git push upstream 2015.09.10-1
```

And the second:

```
$ git checkout 2015.09.03-1
$ git cherry-pick <commit hash> # as many times as you need
$ git tag 2015.09.10-2
$ git push upstream 2015.09.10-2
```

Then update the push doc with the new comparison link for the updated tag.

**Extension Workshop**

The tag naming scheme for extension-workshop is a bit different. When tagging for stage, add the suffix `-stage` to the tag name. For example:

```
$ git checkout master
$ git pull
$ git tag 2022.02.10-stage
$ git push upstream 2022.02.10-stage
```

**Note:** When pushing to stage you **must** create a tag with the `-stage` suffix. If you create and push a tag without that suffix you will be deploying to **production**.

**Push to stage**

Our infrastructure automates pushing the tags to stage once the tags have been pushed up to the repository.

The only required step is to check that the tag has deployed by looking out for the automated push messages in the internal slack channel.

You can also check `/__version__` or `/__frontend_version__` on a given service which shows the currently deployed revision and tag e.g:

- Addons Server (stage)
- Addons Frontend (stage)
- Addons Code Manager (stage)

Note that for Extension Workshop, pushing a tag to `upstream` with a name including the `-stage` suffix, will automatically deploy the tag to stage. You should manually verify Extension Workshop on stage after the push has
completed. Visit https://extensionworkshop.allizom.org and view any pages that have been changed since the last push to verify they exist and are rendering properly.

**Extract locales**

Once you are done pushing the tags to stage:

- Run the `./bin/run-l10n-extraction` command in `addons-frontend` repository (documentation).
- Run the `./scripts/extract-l10n.sh` command in `addons-server` repository.

Because the `addons-server` is meant to be used inside the docker container, it doesn’t have access to your `git` credentials, so you’ll need to enter the container, run the script, exit the container and push the branch created by script. The `addons-frontend` one does that for you, but in both cases you’ll need to open the pull request yourself.

**Before the push**

Keep an eye out for any blocking bugs. Add them to the etherpad and find someone to work on them.

**4.1.2 Push**

The tag is pushed to production by ops (wezhou), once approved by QA (Krupa), on Thursdays. It is the responsibility of the push hero to follow-up with QA and ops, and be around during the push for any unexpected issues.

**Extension Workshop**

We push to extension-workshop on prod manually by creating a pushing tag without the `--stage` suffix. For example:

```
$ git checkout 2022.02.10-stage
$ git tag 2022.02.10
$ git push upstream 2022.02.10
```

You should manually verify Extension Workshop on prod after the push has completed. Visit https://extensionworkshop.com and view any pages that have been changed since the last push to verify they exist and are rendering properly.

**Monitoring the push**

The best places to monitor the results of the push are:

- Sentry
- Grafana
  - Ops dashboard
  - Prod Health dashboard
  - API usage/performance dashboard

The site performance graphs should show no large spikes or changes. Sentry should show no new errors, although it will show intermittent errors and the occasional error as the push occurs.
Once complete

Create a new github document for the next push, for example:

https://github.com/mozilla/addons/blob/main/releases/2019/05/09.md

You can use this handy template:

```markdown
# AMO Release Thursday 13th September 2018

This week’s push hero is @GithubUsername

## Notable things shipping:

## Blockers:

## Cherry-picks:

<!-- Link to the actual commits, NOT merge commits. The commits need to appear in chronological order so that 'git cherry-pick' will apply them correctly. -->

## Pushing:

- https://github.com/mozilla/addons-server/compare/YYYY.MM.DD...YYYY.MM.DD
- https://github.com/mozilla/addons-frontend/compare/YYYY.MM.DD...YYYY.MM.DD
- https://github.com/mozilla/addons-code-manager/compare/YYYY.MM.DD...YYYY.MM.DD

## Before we push:

## Before we start:

## Before we promote:

## After we're done:

```

Set the topic of the AMO Matrix channel and #remora slack channel to include the new github doc link and the relevant nickname of next week’s push hero.

### 4.1.3 Future Goals

Move to continuous deployment and change the way this is done dramatically.
UX for the server and client.

Contents:

5.1 Add-ons Discovery Pane UX Specs

So I can find useful and trusted add-ons, as a new add-ons user, I want to browse from a small list of curated content.

5.1.1 changelog

July 12 - Added a slight drop shadow behind the extension/theme blocks.

May 25 - Changed app icon background to #ffffff. - Added new screen for 2nd or 3rd visit user prompts (still a wip)
• When a user clicks “Click to play”, the copy to the left of the video thumbnail will fade away and the thumbnail will expand down and to the left.
• When a user clicks “close video” below the expanded video, the video will shrink up and to the right back to its original position. The copy to the left will fade back in.
5.1.3 Switch Interaction

- When a user turns on the switch, the install process begins: the switch shows download and install progress. When complete, a check mark appears in the switch.

- When a user turns off the switch, the uninstall process begins: the switch shows uninstall progress. When complete, the switch returns to original off state.
5.1.4 Theme Interaction

- When hovering over the theme image, the toolbar will change to show a preview.

5.1.5 AMO Link

- A button at the bottom of the discovery pane encourages users to visit AMO if they want to explore add-ons and themes further.

5.2 AMO to Firefox Account Migration UX Specs

So that I can reduce the numbers of logins I have, as a Mozilla Add-ons user, I want to switch from using my AMO login and start using a Firefox Account.
5.2.1 User Flow Diagram

Red line denotes common post-migration login experience.

5.2.2 Migrating from AMO

- When a currently authenticated AMO user returns to AMO, they will see a notice prompting them to upgrade their account.
- When a currently authenticated AMO user chooses the call to action in the notice, they are taken to the AMO login to confirm their identity.
- When an existing AMO user enters their email address, they are taken to the AMO login.
- When they log in successfully, they are shown a notice prompting them to upgrade their account.
- When they click Continue and do not have a Firefox Account, they taken to FxA registration that is pre-populated with their email.
- When they click Continue and do have a Firefox Account, they taken to FxA login that is pre-populated with their email.
- When they click Skip, they taken to AMO which contains an upgrade notice.
- When they attempt to use an email that is unverified with FxA, they are taken to the confirm your account screen.
- When they change their email to one that already exists in FxA or AMO, they see the message This account already exists on Mozilla Add-ons.
- When they are successful at migrating, they see the notice Great job! You can now log in to AMO using your FxA.
5.2.3 Registering

- When an unauthenticated user chooses to register, they will see a Join Mozilla Add-ons form that asks for their email, with secondary option to switch to Sign in.
- When a new user enters their email address, they are taken to the FxA registration.
- When an FxA user enters their email address, they are taken to the FxA login.
- When they change their email to one that already exists in FxA or AMO, they see the message This account already exists on Mozilla Add-ons.
- When a registering user successfully verifies their Firefox Account, they are taken to an incomplete AMO settings view that requires profile completion.
- When a registering user completes their AMO profile, the old AMO account can be retired from service.

5.2.4 Logging In

- When any unauthenticated user chooses to log in, they will see a Sign in to Mozilla Add-ons form that asks for their email, with secondary option to switch to Create an account.

5.2.5 Post-Migration

- When an already-migrated user returns to AMO, they will no longer see a notice prompting them to upgrade their account.
- When an already-migrated user chooses to log in, they will see a Sign in to Mozilla Add-ons form that asks for their email, with secondary option to switch to Create an account.
- When an already-migrated users clicks Continue, they taken to FxA login that is pre-populated with their email.
- When an already-migrated user logs in, they are taken to AMO.
5.3 Management UX

5.3.1 Flow

5.3.2 Wireframes
5.4 Submission UX

5.4.1 Flow

5.4.2 Wireframes
Random Stuff

Contents:

6.1 Github labels

Any repositories related to add-ons can have any labels of any format that it wants. But here’s a core list of labels and colours that we’d like to have a bit of consistency. Some key labels:

- blocks release
- blocked by upstream
- good first bug
- maybe good first bug
- not ready for merge
- pull request ready
- priority one
- priority two
- triaged

Colour wise:

- red (#b60205), pay attention to these, eg: “priority one”
- green (#0e8a16), projects that we might work on, eg: “Firefox Accounts” or “signing”
- light blue (#c7def8), often left over from Bugzilla, usually and end statement eg: “invalid”, “won’t fix”, “enhancement”
- purple (#5319e7), needs... eg: “needs ux”
- yellow (#fbca04), intermediary state eg: “maybe good first bug” or “in progress”
- blue (#0052cc), a project area at the technical level, eg: “code quality”

Other comments:

- no need to add in - or _ between words, eg “pull-request-ready” vs “pull request ready”
- you can be verbose, eg: “needs more qa” vs “qa+”
Note: We maintain a JSON file with a list of common GitHub labels. If you want to add those labels to a repository, you can use the following command (a personal access token with “write” permission should be generated).

```
$ npx github-labels -t <token> -c ./default-github-labels.json
$ # You can pass `-f` to remove all existing labels before creating the new ones
```

Filing bugs:

- Github on [this repository](#).
- Bugzilla
INDICES AND TABLES

- genindex
- modindex
- search