Add-on Builder Documentation

Release 0.9.18beta

Mozilla

February 25, 2014

Hi, this is the development documentation for FlightDeck.

You may find some valuable info on the official Wiki

To get started please read the *Installation section*.

Contents:

CHAPTER 1

Installation

1.1 Requirements

FligtDeck depends on:

- Python 2.6
- MySQL
- git

We also suggest:

- virtualenv
- virtualenvwrapper

1.2 Installing

It's outside the scope of this document, but we suggest you do all your work from within a virtualenv so your python packages don't conflict with others on the system. Now's the time to get in your virtualenv!

If you're going to be contributing, please fork http://github.com/mozilla/FlightDeck before continuing so you can push code to your own branches. Then, download the code, substituting your name:

```
git clone git@github.com:{your-username}/FlightDeck.git  # if you're not a developer, just use "mozil
cd FlightDeck
```

Install submodules:

```
git submodule update --init --recursive
```

Install any compiled libraries:

```
pip install -r requirements/compiled.txt
```

Configure the site by creating a settings_local.py in your root. Anything you put in here will override the defaults in settings.py. An example follows, note the first line is required:

```
from settings import *

DEBUG = True
TEMPLATE_DEBUG = DEBUG
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'flightdeck',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': '',
        'OPTIONS': {
            'init_command': 'SET storage_engine=InnoDB'
        },
        'TEST_CHARSET': 'utf8',
        'TEST_COLLATION': 'utf8_general_ci',

    }
}

UPLOAD_DIR = "/tmp/flightdeck"

SESSION_COOKIE_SECURE = False

ES_DISABLED = True # enable when ES daemon is running
ES_HOSTS = ['127.0.0.1:9201']
ES_INDEX = 'flightdeck'

CACHES['default']['BACKEND'] =
'django.core.cache.backends.locmem.LocMemCache'
```

Make sure that MySQL is running, then create the database you specified in settings_local.py:

```
mysql -u root -p

[MySQL messages snipped]

mysql> CREATE DATABASE flightdeck;
Query OK, 1 row affected (0.00 sec)
```

If this is a brand new installation you'll need to configure a database as well. This command will build the structure:

```
./manage.py syncdb
```

If you're using Elastic Search locally (this is not necessary for basic functionality) then be sure to setup the ES index mappings and index all your packages:

```
./manage.py cron setup_mapping
./manage.py cron index_all
```

FlightDeck needs to know about the SDKs you have available (in ./lib). This command will make a single version of the SDK available in FlightDeck's Libraries selector:

```
./manage.py add_core_lib addon-sdk-1.2.1
```

If you're writing code and would like to add some test data to the database you can load some fixtures:

```
./manage.py loaddata users packages
```

Run the development server:

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

In Firefox's about:config create a new string preference named extensions.addonBuilderHelper.trustedOrigins with the value `https://builder.addons.mozilla.org/,http://127.0.0.1:8000/`; install the Add-on Builder Helper (if you had it already installed, restart the browser after changing the preference)

Navigate the browser to http://127.0.0.1:8000/, log in with the username and password you entered while running `./manage.py syncdb`.

You're all done!

CHAPTER 2

Recipes

2.1 Create a local super user account

If you imported your database then you will need to create a user:

```
./manage.py createsuperuser
```

2.2 Building documentation

FlightDeck uses Sphinx-based documentation, so you have to install sphinx in order to build the docs:

```
pip install sphinx
make -C docs html
```

Note: If you get `ValueError: unknown locale: UTF-8`, run `export LC_ALL=en_US.UTF-8` before `make`.

2.3 Import live database dump

How to import a database dump from live:

```
[sudo] mysql flightdeck < flightdeck_dump.sql
```

If you run into an error when importing large sql dump files, you may need to restart your mysqld process with this parameter:

```
mysqld --max_allowed_packet=32M
```

The database dump might be missing a row in django_sites table, so if you get a django error saying "Site matching query does not exist" when you hit the login page then insert a row into django_site:

```
insert into django_site (id,domain,name) values (1,'example.com','example')
```

After importing the data, you will need to rebuild your ES index.

2.4 Elastic Search

ElasticSearch is a Lucene based search engine that powers FlightDeck search. We also use pyes a pythonic interface to ElasticSearch.

Running

You will need to point it at a config file that we've included in `scripts/es.yml`:

```
elasticsearch -f -Des.config=./scripts/es.yml
```

This configuraion can be overridden if necessary. More details are here.

Development

`settings.py` needs to be overridden in order to use ElasticSearch. Both `ES_DISABLED` needs to be `False` and `ES_HOSTS` needs to be set. This can be done in `settings_local.py`.

Testing

In order for testing to work `ES_HOSTS` needs to be defined (otherwise SkipTest will be raised) and ElasticSearch needs to be running. We specifically look at a single index, `test_flightdeck`, in order to avoid conflicts with development data.

Rebuilding Elastic Search index

Need to delete your Elastic Search index and start over?:

```
curl -XDELETE 'http://localhost:9201/flightdeck'
./manage.py cron setup_mapping
./manage.py cron index_all
```

2.5 Using with Celery

Majority of resources heavy tasks is done by delegating them to celery.

By default on development boxes celery is not running and tasks are run synchronously. To be able to test celery tasks one has to configure the development system to resemble the production one.

Celery requires a running messaging system. We use RabbitMQ.

To configure please copy the Celery section from `settings.py` to `settings_local.py` and uncomment it.

```
# These settings are for if you have celeryd running
BROKER_HOST = 'localhost'
BROKER_PORT = 5672
BROKER_USER = 'builder'
BROKER_PASSWORD = 'builder'
BROKER_VHOST = 'builder'
BROKER_CONNECTION_TIMEOUT = 0.1
CELERY_RESULT_BACKEND = 'amqp'
CELERY_IGNORE_RESULT = True
```

RabbitMQ CheatSheet

Create user, virtual host and give user all privileges:

```
sudo rabbitmqctl add_user builder builder
sudo rabbitmqctl add_vhost builder
sudo rabbitmqctl set_permission -p builder builder ".*" ".*" ".*"
```

From project directory run:

```
./manage.py celeryd -l INFO
```

2.6 Using Apache

Production environments will expect to be running through another webserver. An example `apache.conf`

An example Apache WSGI configuration `apache.wsgi`

CHAPTER 3

Contribution

Content

3.1 Code Workflow

We use a simplified Vincent Driessen's model

3.1.1 Decentralized, but Centralized

Every developer has his own repository (we called `origin` in this and following documents). There is only one referencing *upstream* repository. `upstream/master` branch is deployed to http://builder-addons-dev.allizom.org/ by Github's PUSH request. Production server is updated from tags (i.e. `0.9.12`) by IT team.

bug-*

Must branch of from `master` or other `bug-*`. Naming convention: `bug-{bugzilla_bug_id}-short_description`

Fixes one bug. It will be merged to the `upstream.master`

production

If there is a serious bug in production we branch off tag to `upstream/production` branch, fix it and tag it again.

hotfix-*

Must branch off from `upstream/production` Merges back to `upstream/production` and eventually `upstream/master`

This branch is for fixes to freezed code - in production

3.2 Cheatsheet - Typical git commands used by developer

Developer has a set of commands which are common for that workflow. Please take these advices as a starting point. They do not cover whole git functionality.

3.2.1 Syncing master branch

Master branch has to be usually in sync with the main master branch:

```
git checkout master
git pull upstream master
```

3.2.2 Fixing a bug

Checkout the branch which needs to be fixed. If it's `master` (most common case), first sync as above.

Create a branch with a bug number:

```
git checkout -b bug-12345-bug_description
```

Note: If the bug is a hotfix it will be called hotfix-12345-branch_description

Make some changes, publish the bug to the `origin` repository:

```
git commit [ list_of_files | -a ] -v
git push origin bug-12345-bug_description
```

Send a pull request.

After the bug has been succesfully resolved the branch may be removed:

```
git branch -d bug-12345-bug_description
```

3.2.3 Working with a fellow developer

Sometimes on one bug there will be working more people. It is advised to use the same branch name.

First create am alias for the remote repository:

```
git remote add -t bug_12345-bug_description fellow http://github.com/{fellow_username}/FlightDeck.git
```

Create a branch which will merge from the remote repository:

```
git checkout -b bug_12345-bug_description fellow/bug-12345-bug_description
```

Sending the changes to `origin` works as before:

```
git commit [ list_of_files | -a ] -v
git push origin bug-12345-bug_description
```

If you'd like later to load changes done by "fellow" - pull them from the remote branch:

```
git pull fellow/bug-12345-bug_description
```

3.3 Database Migration

Add-ons Builder uses Schematic. *"The worst schema versioning system, ever?"*.

3.3.1 Usage

Applying migrations

```
./vendor/src/schematic/schematic ./migrations/
```

Creating migrations

Create `migrations/{number}-{some_name}.[py/sql]` file (check `migrations` directory for examples). Python files will be executed and SQL run directly on database.

Troubleshooting

Schematic is storing current migration number in `schema_version` table. Change it if you've created database by `./manage.py syncdb`.

3.4 Vendor

Vendor is a submodule. It contains all 3rd party libraries needed to run the FlightDeck on the server (i.e. Django).

One can change it's content. To do so please clone it's repository located at https://github.com/mozilla/FlightDeck-lib into a separate project.

There is a nice tool called vending machine which helps with management:

```
pip install -e git://github.com/jbalogh/vending-machine#egg=vend
```

From the help:

```
usage: vend [-h] [-d DIR] {add,update,uninstall,sync,freeze} ...

positional arguments:
  {add,update,uninstall,sync,freeze}
    sync                sync requirements file with vendor
    freeze              freeze requirements for the vendor repo
    update              update a package or submodule
    uninstall           uninstall a package or submodule
    add                 add a package or submodule

optional arguments:
  -h, --help            show this help message and exit
  -d DIR, --dir DIR     path to the vendor directory
```

However, because `vendor` is a submodule, vending-machine should not be used in it's default behavior. Instead, FlightDeck-lib should be checked out from **'it's repository https://github.com/mozilla/FlightDeck-lib'_** to a separate folder, and you should set the `-d` argument of vend:

```
vend -d ./FlightDeck-lib add elasticutils
```

CHAPTER 4

Repackage

Repackage is a server service which is converting provided Add-on into a `xpi` using chosen Add-on SDK.

There are currently two types of this feature depending on the way Add-on is given.

4.1 Builder Add-on Repackage

Feature available for Add-ons created and saved in the Builder. Addons are identified by `PackageRevision`'s id in the database

4.1.1 API for Builder Add-ons

URL: `/repackage/rebuild-addons/`

method: POST

Fields:

secret password proving the request came from a trusted source

sdk_version version of the SDK which should be used to rebuild the package

addons JSON string - a list of dicts containing addons data. `[{"package_key": 1234, "version": "force.version", ... }]`. All of the `package.json` may be overwritten.

package_key is the unique identifier of the `PackageRevision` in the Builder

version (optional) is the way to force the version with which the xpi will be built.

priority (optional) if it is present set it to `'high'` - force the priority of the task.

pingback (optional) URL to pass the result

4.2 XPI Repackage

Decompile Add-on provided by `xpi` file and rebuild using chosen SDK. This feature is available for all add-ons build with Jetpack SDK.

4.2.1 API

URL: `/repackage/rebuild/`

method: POST

Fields:

One of the `location`,‘‘upload‘‘ or `addons` fields must be present. `location` and `upload` can't be provided together.

priority force the priority of the task

secret password proving the request came from a trusted source

location URL for the `XPI` file to download

upload `XPI` file uploading

addons JSON string - a table of dicts containing addons data. `[{"location": "ftp://{...}", "version": "force.version" }]`. It can use all of the `package.json` fields provided below, `filename`. It has to contain `location` in every dict.

pingback URL to pass the result

filename desired filename for the downloaded `XPI`

version, type, fullName, url, description, author, license, lib, data, tests, main (optional)

Force `package.json` fields. If `version` field contains a `{sdk_version}` string it will get replaced with SDK version used to repackage. Specifically `"version": "0.1.sdk.{sdk_version}"` will be replaced with `"version": "0.1.sdk.1.0b5"`.

sdk_version version of the SDK which should be used to rebuild the package

Examples of data creation for POST:

```
# single addon rebuild with download
post = {'addon': file_.version.addon_id,
        'file_id': file_.id,
        'priority': priority,
        'secret': settings.BUILDER_SECRET_KEY,
        'location': file_.get_url_path(None, 'builder'),
        'uuid': data['uuid'],
        'pingback': reverse('files.builder-pingback'),
        'version': 'force_version'}

# single addon rebuild with upload
post = {'addon': file_.version.addon_id,
        'file_id': file_.id,
        'priority': priority,
        'secret': settings.BUILDER_SECRET_KEY,
        'upload': file_.file,
        'uuid': data['uuid'],
        'pingback': reverse('files.builder-pingback'),
        'version': 'force_version'}
```

```
# bulk rebuild with download
addons = [{'location': f.get_url_path(None, 'builder'),
           'addon': f.version.addon_id,
           'file_id': f.id,
           'version': '%s.rebuild' % f.version} for f in addon_files]

post = {'priority': priority,
        'secret': settings.BUILDER_SECRET_KEY,
        'uuid': data['uuid'],
        'pingback': reverse('files.builder-pingback'),
        'addons': simplejson.dumps(addons)}

# bulk rebuild with upload
addons = []
files = {}
for f in addon_files:
    addons.append({'upload': 'upload_%s' % f.filename,
           'addon': f.version.addon_id,
           'file_id': f.id,
           'version': '%s.rebuild' % f.version})
    files['upload_%s' % f.filename] = f.file

post = {'priority': priority,
        'secret': settings.BUILDER_SECRET_KEY,
        'uuid': data['uuid'],
        'pingback': reverse('files.builder-pingback'),
        'addons': simplejson.dumps(addons)}
post.extend(files)
```

Returns

After the `XPI` has been created Builder will send the response to the pingback URL. Whole request will also be send back.

result "success" or "failure"

msg `stdout` if result is `success` else `stderr` returned by `cfx xpi`

location URL to download the rebuild `XPI` from

secret password proving the request came from a trusted source

request urlified request.POST used for initial request

API response

Response

Send to the pingback

```
data = {
    'id': rep.manifest['id'],
    'secret': settings.BUILDER_SECRET_KEY,
    'result': 'success' if not response[1] else 'failure',
    'msg': response[1] if response[1] else response[0],
    'location': reverse('jp_download_xpi', args=[hashtag, filename]),
    'request': post}
```

4.2.2 Scope

Note: The **scope** is fundamentally determined by the *Strategy* of the site. The *Structure* defines the way in which the various features and functions of the site fit together. Just what those features and functions are constitutes the **scope** of the site.

What is gonna be build

1. A tool to pull `xpi` from AMO and rebuild with given or default SDK

What will not be build

1. Open service for all people.

4.2.3 Strategy

Note: This strategy incorprates not only what the people running the site want to get out of it but what the users want to get out of the site as well. *Users wants to buy books, and we want sell them* Other objectives might not be easy to articulate.

Site objectives

Business Goals

1. An easy way to batch update Add-ons for AMO

Success Metrics

1. XPI is created under 10s

User Needs

1. Obtain a XPI without touching the SDK
2. Send a XPI and request it to be rebuild with the newest SDK
3. Send an id (or a list of ids) of the XPI on an AMO service and reuest it to be rebuild with the newest SDK

User Segmentation

1. AMO service adinistrators
2. Potentially Add-ons developers coding localy)

4.2.4 Structure

Note: The **structure** defines the way in which the various features and functions of the site fit together. It defines the path user has to go to reach any page of the site from the other page

Repackage is an **Application**.

It contains several views and celery tasks needed to complete the goal.

Repackage XPI build is different from Add-on Builder XPI build only in the way it's preparing the packages. Instead of reading them from database/request it's unpacking received XPI.

4.2.5 Implementation

repackage.helpers

class `repackage.helpers.`**`Extractor`**(*install_rdf*)

Extracts manifest from `install.rdf`

modified `Extractor` class from `zamboni/apps/versions/compare.py`

`ADDON_EXTENSION` = **'2'**

`find`(*name*, *ctx=None*)

Like $() for install.rdf, where name is the selector.

`find_root`()

`manifest` = **u'urn:mozilla:install-manifest'**

`read_manifest`(*package_overrides={}*)

Extracts data from `install.rdf`, assignes it to `self.data`

Param target_version (String) forces the version

Returns dict

`uri`(*name*)

class `repackage.helpers.`**`Repackage`**

`cleanup`()

closes all files opened during the repackaging

`download`(*location*)

Downloads the XPI (from `location`) and instantiates XPI in `self.xpi_zip`

This eventually will record statistics about build times

Param location (String) location of the file to download rebuild `XPI`

Returns None

`extract_packages`(*sdk_source_dir*)

Builds SDK environment and calls the `xpi.xpi_utils.build()`

Returns temporary sdk_dir

get_manifest(*package_overrides={}*)
extracts manifest from `install.rdf` it does not contain all dependencies, these will be appended during copying package files

Sets the `self.manifest` field

rebuild(*sdk_source_dir*, *hashtag*, *package_overrides={}*, *options=None*)
Drive the rebuild process

Param sdk_source_dir (String) absolute path of the SDK

Param hashtag (String) filename for the buid XPI

Param target_version (String)

retrieve(*xpi_from*)
Handles upload

Param xpi_from (element of request.FILES)

repackage.helpers.**increment_version**(*version*)
Modify version string to indicate changes made by repacking

Attr version (string) version to be modified

CHAPTER 5

AMO Integration

Builder is tightly integrated with AMO That involves *Syncing Packages*, and login

Contents:

5.1 Syncing Packages

5.1.1 Identification

`Package` has a field `amo_id` which used to store id of the related Package on the AMO. During the synchronization process program_id is updated, so all generated `XPI` are properly identified by AMO.

For validation purposes `PackageRevision` has the fields `amo_status` and `amo_version_name`.

5.1.2 Scenarios:

All of these scenarios are run by the author of the add-on and on the *edit_package-page*.

Create new add-on

Package created in the Builder can be exported to AMO. This action involves creating a new Addon on the AMO, uploading all necessary meta data and a `XPI` build on the Builder.

Update an existing add-on

If a Package is already synchronized, new version might be uploaded to AMO. This requires version name to be changed.

Synchronizing an add-on existing on AMO

> **Warning:** This features is under development

It might happen, that a user will move add-on development to the Builder. To upload a new version of the add-on one needs to link an AMO add-on with the Builder one.

User has an ability to display a list of his add-ons on AMO and choose which one should be linked to the currently displayed add-on.

Attributes `jetpack.models.Package.amo_id` and `jetpack.models.Package.jid` are saved in the separate view. If this was called as a part of uploading an add-on scenario, after the response is received *Update an existing add-on* is called.

5.2 Use Cases

5.2.1 Upload to AMO

- Add-on author clicks on the `UploadToAMO` link.
- Builder validates if all fields are correct (especially if that Add-on with the same version_name was already successfully uploaded to AMO).
- Builder is scheduling a task which creates `XPI`, changes the status to `STATUS_UPLOAD_SCHEDULED` and uploads it to AMO
- User receives a notification `Upload to AMO is scheduled` with a link to AMO Dashboard
- After the upload has been done:
  - User receives a notification from AMO
  - Status is changed to default AMOStatus (`STATUS_UNREVIEWED`)

5.3 Add-on statuses

One can check add-on status in the *modal-properties* or *page-dashboard*.

Every Add-on uploaded to AMO has a status which is one of the following:

- `STATUS_NULL` - Incomplete
- `STATUS_UNREVIEWED` - (**default**) - Awaiting Preliminary Review
- `STATUS_PENDING` - Pending approval
- `STATUS_NOMINATED` - Awaiting Full Review
- `STATUS_PUBLIC` - Fully Reviewed
- `STATUS_DISABLED` - Disabled by Mozilla
- `STATUS_LISTED` - Listed
- `STATUS_BETA` - Beta
- `STATUS_LITE` - Preliminarily Reviewed
- `STATUS_LITE_AND_NOMINATED` - Preliminarily Reviewed and Awaiting Full Review
- `STATUS_PURGATORY` - Pending a review choice

The AMOStatus needs to be stored (and updated) within `PackageRevision`.

Note: Updating PackageRevision status should be done by AMO on every status change. AMO should save the `jetpack.models.PackageRevision.pk` and `jetpack.models.Package.pk` with the uploaded Add-on version

Builder adds another statuses:

- `STATUS_UPLOAD_SCHEDULED` - Upload scheduled
- `STATUS_UPLOAD_FAILED` - Upload failed

If upload finished with success the default AMOStatus will be used.

CHAPTER 6

Elastic Search

ElasticSearch is a Lucene based search engine that powers FlightDeck search. We also use `pyes` (link) a pythonic interface to ElasticSearch.

6.1 Running ElasticSearch

FlightDeck was developed with ElasticSearch 14.4 so we recommend downloading that and running it. You will need to point it at a config file that we've included in `scripts/es/es.yml`:

```
elasticsearch -f -Des.config=$ROOT/scripts/es/es.yml
```

Where `$ROOT` is your FlightDeck home.

6.2 Configuration

This configuraion can be overridden if necessary. FlightDeck by default uses port 9201 and 9301. More details are here.

6.3 Development

`settings.py` needs to be overridden in order to use ElasticSearch. Both `ES_DISABLED` needs to be `False` and `ES_HOSTS` needs to be set. This can be done in `settings_local.py`.

6.4 Testing

In order for testing to work `ES_HOSTS` needs to be defined (otherwise SkipTest will be raised) and ElasticSearch needs to be running. We specifically look at a single index, `test_flightdeck`, in order to avoid conflicts with development data.

6.5 Todo

In the future we may need to:

- Add items and remove items asynchronously using Celery.
- Build a frontend for search.
- Add custom mapping.

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*

r