Mozilla Addon Builder
Package Building System

Piotr Zalewa

May 4, 2010, alpha 0.1

Download this document from
`http://github.com/zalun/FlightDeck/raw/master/Docs/Package%20Building\%20System.pdf`
If in doubts, please take a look at the accompanied slides at
`http://github.com/zalun/FlightDeck/raw/master/Docs/Addon\%20Builder\%20-\%20Build\%20System.pdf`

1 Assumptions for the current iteration

1. Name of the Package is not unique anymore.
   Packages are identified by it's *unique ID*. There may and probably often will be many Packages with the same name[1].
   `/library/123456/`

2. Version is a tag.
   Version is important. It is used to tag major *Revisions*. If a package is called without any Version specified (as above), the latest versioned Revision will be used.
   `/library/123456/version/0.1/`

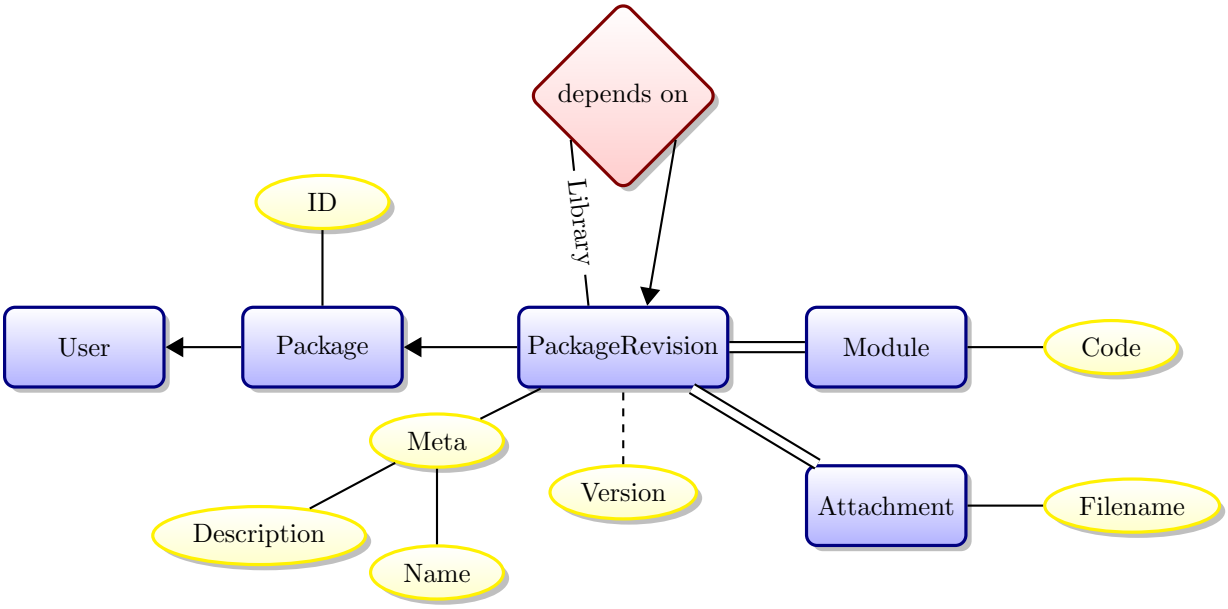
3. Revision Number is used to precisely identify a Revision.
   It is completely parallel to the Package Version
   `/library/123456/revision/654/`

4. No collaborative editing.
   Althought there will be no connection between Packages owned by different Users, design the system to not complicate future implementation of such functionality.

5. Package remembers wich SDK version was used to build it.
   This is very complicated also on the front-end side. It will be created during the next iteration.

[1] Check if it will not make any problem with Addons and uploading to `AMO`

2 Logical structure



3 Exporting XPI

Be aware that it is possible and common to export XPI[2] from partially unsaved data. This happens when User will use the "Try in browser" functionality. In this case XPI may not be send to AMO[3].

3.1 Creating directory structure

Directory structure should be as close as standard Jetpack SDK as possible. Jetpack SDK should be copied to a temporary directory as more than one Addon compilation could take action at the same time. Desired revisions of Libraries and Addons will be exported into packages directory.

```
/tmp/jetpack-sdk-\{hash\}/
|-- bin/
|   |-- activate
|   |-- cfx
|   `-- [...]
|-- packages/
|   |-- jetpack-core/
|   `-- [...]
|-- python-lib/
|-- static-files/
`-- [...]
```

Figure 1: Parts of the tree of a copied Jetpack SDK directory

[2] An XPI (pronounced "zippy" and derived from XPInstall) installer module is a ZIP file that contains an install script or a manifest at the root of the file, and a number of data files.

[3] http://addons.mozilla.org/

3.2 Export Packages with Modules

1. Create Package and its Modules directories
   `/tmp/jetpack-sdk-{hash}/packages/{Package:name}/`
   `/tmp/jetpack-sdk-{hash}/packages/{Package:name}/lib/`

2. Use collected data to create the Manifest.
   `/tmp/jetpack-sdk-{hash}/packages/{Package:name}/package.json`

3. Create Module files
   Iterate over the assigned Modules and create a "`.js`" file with its content inside Package's `lib/` directory.

4. Export dependencies
   Iterate over Libraries on which a Package depends and repeat this section (*Export the Package with Modules*) for every Library.

3.3 Building `XPI`

1. Set virtual environment to the temporary Jetpack SDK

2. Change directory to `/tmp/jetpack-sdk-{hash}/packages/{Package:name}/`

3. Call `cfx xpi`.
   The {`Package:name`}`.xpi` file will be created in current directory.

4. Send location to the front-end to be used in further actions
   In example calling the *FlightDeck Addon*[4] to download and install the `XPI`.

3.4 Uploading to AMO

`XPI` needs to be created from a database object. Then use `mechanize` lib to login to `AMO`and upload the file faking it was done directly from the browser.

To be continued...

[4]FlightDeck Addon is a Jetpack extension allowing to temporary installation of the `XPI`. It needs to be called with an URL of the `XPI`.