Mozilla Addon Builder
Package Building System

Piotr Zalewa

May 4, 2010, alpha 0.1

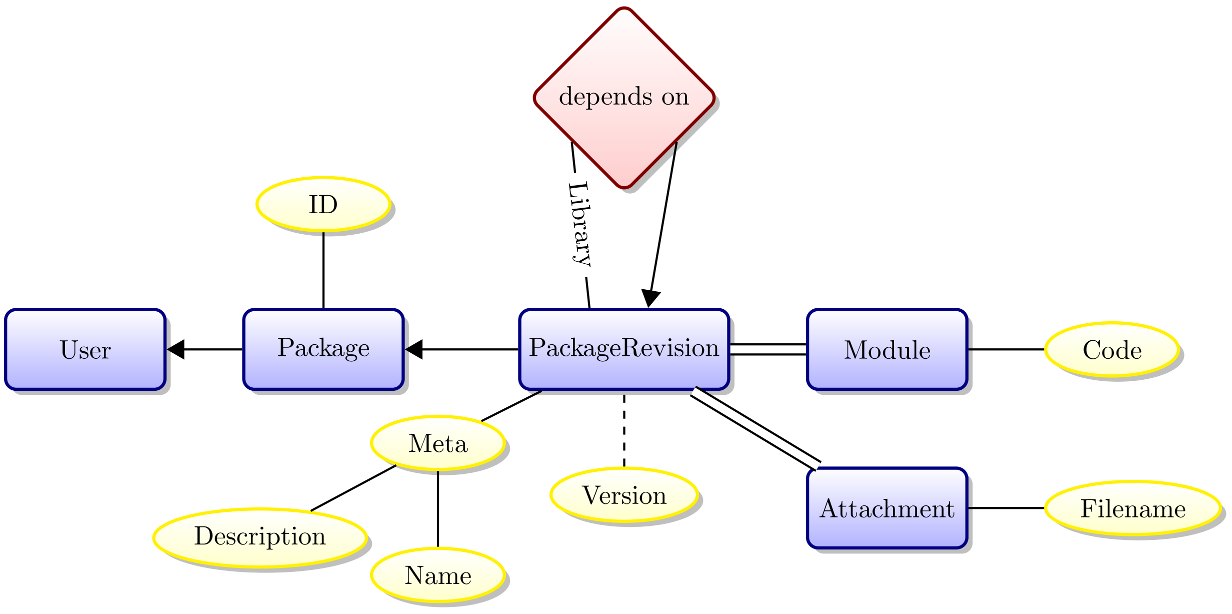
Download this document from
`http://github.com/zalun/FlightDeck/raw/master/Docs/Package%20Building\%20System.pdf`
If in doubts, please take a look at the accompanied slides at
`http://github.com/zalun/FlightDeck/raw/master/Docs/Addon\%20Builder\%20-\%20Build\%20System.pdf`

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, 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.
   That means there is no connection between Packages owned by different Users.
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.

Copy Jetpack SDK to a temporary dir

```
/tmp/jetpack-sdk-
|-- bin
|   |-- activate
|   |-- bin
|   `-- [...]
|-- packages
|   |-- development-mode
|   |-- jetpack-core
|   |-- nsjetpack
|   `-- test-harness
|-- python-lib
|-- static-files
`-- [...]
```

[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 Exporting Packages with Modules

1. Create Package and its Modules directories
   /tmp/packages_{hash}/{Package:name}/
   /tmp/packages_{hash}/{Package:name}/lib/
2. Use collected data to create the Manifest.
   /tmp/packages_{hash}/{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

System is already in a virtual environment knowing about Jetpack SDK. It is enough to change directory to /tmp/packages_{hash}/{Package:name}/ and call cfx xpi. The {Package:name}.xpi file will be created in current directory. Its location is then send to the front-end to be used in further actions, usually calling the *FlightDeck Addon*[4] to download and install the XPI.

3.4 Uploading to AMO

Create XPI from the database object. Use mechanize lib to login to AMOand upload the file faking it was done directly from the browser.

4 Editing Package and its Modules

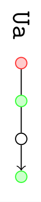
How database evolves by changing the Packages and Modules. This description will be used later to design structure and functionalities of the system.

4.1 Starting point

All next scenarios start from the Ua:La.1 defined as below.

$\texttt{La} \Longrightarrow \texttt{Ua:La.1} \supset \{\texttt{Ua:La.1:Ma}\}$

Package La is created by User Ua.
La's HEAD is PackageRevision identified as Ua:La.1
It contains only one module - Ma
Following steps had to happen to achieve above status:

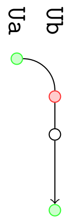


Ua creates empty Library La	$\texttt{Ua:La.0} \supset \{\}$
System sets La's HEAD	$\texttt{La} \Longrightarrow \texttt{Ua:La.0}$
Ua adds new Milestone Ma to La	$\texttt{Ua:La.1} \supset \{\texttt{Ua:La.1:Ma}\}$
Ua sets the HEAD	$\texttt{La} \Longrightarrow \texttt{Ua.La.1}$

[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.

4.2 Scenario (1 Module, 2 Users, no dependencies)

Ua and Ub are working on La
Ub modified one module

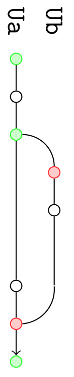


Ub creates new branch	Ub:La.0 ⊃ {Ua:La.1:Ma}
Ub saved changes to Ma	Ub:La.1 ⊃ {Ub:La.1:Ma}
Ub sends *request* to La's creator to updade La's HEAD	
Ua accepts the request by setting the HEAD to Ub's version	La ⟹ Ub:La.1

Result: La ⟹ Ub:La.1 ⊃ {Ub:La.1:Ma}

4.3 Scenario (2 Modules, 2 Users, no dependencies)

Ua and Ub are working on La
Ua created module Mb
Ub is working on Mb

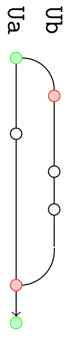


Ua adds a new module Mb to La	Ua:La.2 ⊃ {Ua:La.1:Ma, Ua:La.2:Mb}
Ua sets the HEAD	La ⟹ Ua:La.2
Ub creates new branch	Ub:La.0 ⊃ {Ua:La.1:Ma, Ua:La.2:Mb}
Ub modifies Mb	Ub:La.1 ⊃ {Ua:La.1:Ma, Ub:La.1:Mb}
Ub sends request to Ua to upgrade La	
Ua modifies Ma	Ua:La.3 ⊃ {Ua:La.3:Ma, Ua:La.2:Mb}
Ua acepts Ub's request, upgrades La	Ua:La.4 ⊃ {Ua:La.3:Ma, Ub:La.1:Mb}
Ua sets the HEAD	La ⟹ Ua:La.4

Result: La ⟹ Ua:La.4 ⊃ {Ua:La.3:Ma, Ub:La.1:Mb}

4.4 Scenario (2 Modules, 2 Users, no dependencies)

Ua and Ub are working on La
Ub created module Mb



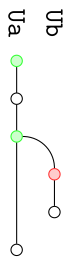
Ub creates new branch	Ub:La.0 ⊃ {Ua:La.1:Ma}
Ua modifies Ma	Ua:La.2 ⊃ {Ua:La.2:Ma}
Ub adds a new module Mb to La	Ub:La.1 ⊃ {Ua:La.1:Ma, Ua:La.1:Mb}
Ub modifies Mb	Ub:La.2 ⊃ {Ua:La.1:Ma, Ub:La.2:Mb}
Ub sends request to Ua to upgrade La	
Ua acepts Ub's request	Ua:La.3 ⊃ {Ua:La.2:Ma, Ub:La.2:Mb}
Ua sets the HEAD	La ⟹ Ua:La.3

Result: La ⟹ Ua:La.3 ⊃ {Ua:La.2:Ma, Ub:La.2:Mb}

4.5 Scenario with conflict (2 Modules, 2 Users, no dependencies)

`Ua` and `Ub` are working on `La`
`Ua` created module `Mb`
`Ua` and `Ub` are working on `Mb`
Conflict arises...

Steps leading to the conflict:

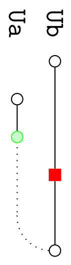


`Ua` adds a new module `Mb` to `La`	`Ua:La.2` $\supset$ {`Ua:La.1:Ma`, `Ua:La.2:Mb`}
`Ua` sets the HEAD	`La` $\Longrightarrow$ `Ua:La.2`
`Ub` creates new branch	`Ub:La.0` $\supset$ {`Ua:La.1:Ma`, `Ua:La.2:Mb`}
`Ub` modifies `Mb`	`Ub:La.1` $\supset$ {`Ua:La.1:Ma`, `Ub:La.1:Mb`}
`Ua` modifies `Mb`	`Ua:La.3` $\supset$ {`Ua:La.1:Ma`, `Ua:La.3:Mb`}

Libraries `Ub:La.1` and `Ua:La.3` are **conflicted** because `Ub:La.1:Mb` and `Ua:La.3:Mb` are both an evolution of the `Ua:La.2:Mb`. From that moment many scenarios may happen. Just a few of them will follow.

4.5.1 `Ua` sets HEAD and `Ub`'s revision is outdated

`La`'s manager — `Ua` has chosen the HEAD. At that moment he doesn't know about `Ub`'s changes to `Mb`.



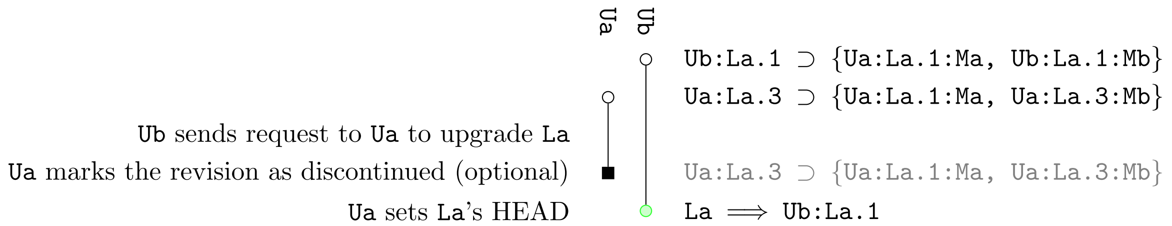
	`Ub:La.1` $\supset$ {`Ua:La.1:Ma`, `Ub:La.1:Mb`}
	`Ua:La.3` $\supset$ {`Ua:La.1:Ma`, `Ua:La.3:Mb`}
`Ua` sets the HEAD	`La` $\Longrightarrow$ `Ua:La.3`
`Ub:La.1` is marked as *conflicted*. Send update request disabled	`Ub:La.1` $\supset$ {`Ua:La.1:Ma`, `Ub:La.1:Mb`}
`Ub` receives info that his source is behind the HEAD	
`Ub` manually solves conflict by editing `Mb`	`Ub:La.2` $\supset$ {`Ua:La.1:Ma`, `Ub:La.2:Mb`}
`Ub` sets the conflict as resolved	

From that moment `Ub`:La.2 becomes a normal (not conflicted) PackageRevision. `Ub` may send Package manager an upgrade request which could end by switching `La`'s HEAD to `Ub`:La.2. It is important to note, that the `Ub:La.2` is not an evolution of `Ua:La.3`, it will not be originated from it.[5]

4.5.2 `Ub` sends update request, `Ua` decides to drop his changes

`Ub` thinks his change to `Mb` is finished and requests update of the Library from its manager — `Ua`. He accepts the request and marks his version of this module as discontinued. This mark prevents from the automatic set to conflicted revision.

[5]Decide if this is the right thing to do.



`Ub:La.1` $\supset$ {`Ua:La.1:Ma, Ub:La.1:Mb`}

`Ua:La.3` $\supset$ {`Ua:La.1:Ma, Ua:La.3:Mb`}

`Ub` sends request to `Ua` to upgrade `La`

`Ua` marks the revision as discontinued (optional) `Ua:La.3` $\supset$ {`Ua:La.1:Ma, Ua:La.3:Mb`}

`Ua` sets `La`'s HEAD `La` $\Longrightarrow$ `Ub:La.1`

Draft/Ideas

update Library if Library HEAD has been changed something should tell the User that an update is possible. It should then (on request) change the versions of all Modules which are not in conflict with updating Library. In essence, if
`Ua:La.1` $\supset$ {`Ua:La.1:Ma, Ub:La.2:Mb`} is a Library to be updated and
`La` $\Longrightarrow$ `Uc:La.3` $\supset$ {`Ub:La.1:Ma, Uc:La.3:Mb, Uc:La.1:Mc`} is current HEAD, then
`Ub:La.2:Mb` should be updated to `Uc:La.3:Mb` and `Uc:La.1:Mc` should be added.
User should receive a notification that `Ua:La.1:Ma` is not in sync with HEAD.

To be continued...