

Mozilla Addon Builder

Definition of the Package Building System

Piotr Zalewa

May 2, 2010

If in doubts, please take a look at the accompanied document:

<http://github.com/zalun/FlightDeck/raw/master/Docs/Addon%20Builder%20-%20Build%20System.pdf>.

1 Syntax

1.1 Objects

x, y, z — represents $[a..z]$

m, n — represents $[0..9]^+$

Ux is the specific User (identified by *User:name*)

Px is the specific Package (identified by *Package:name*)

It should always be used within its **type** context as Lx — Library or Ax — Addon

Every Package has an associated PackageRevision¹ (identified by a triplet $Ux:Py.n$ *User/Package/PackageRevision:revisionNumber*)

Mx is the Module (identified by $Ux:Py.n:Mz$ *PackageRevision/Module:name*²)

1.2 Object identification — revision numbers and HEAD

$Ux:Py.n$ defines revision of the Package.

$Ua:La.1$ — First revision of Library La saved by Ua .

$Ux:Py.n:Mz$ defines the precise Module revision — a Module inside the PackageRevision.

$Ua:La.1:Ma$ — Module Ma inside the first revision of Library La saved by Ua .

$Px \Rightarrow Uy:Px.n$ is the HEAD revision of the Package

$La \Rightarrow Ua:La.1$ — La 's HEAD points to the first revision of Library La saved by Ua .

$Ux:Py.n \supset \{Ux:Py.m:Mz, \dots\}$ Modules inside the Package revision.

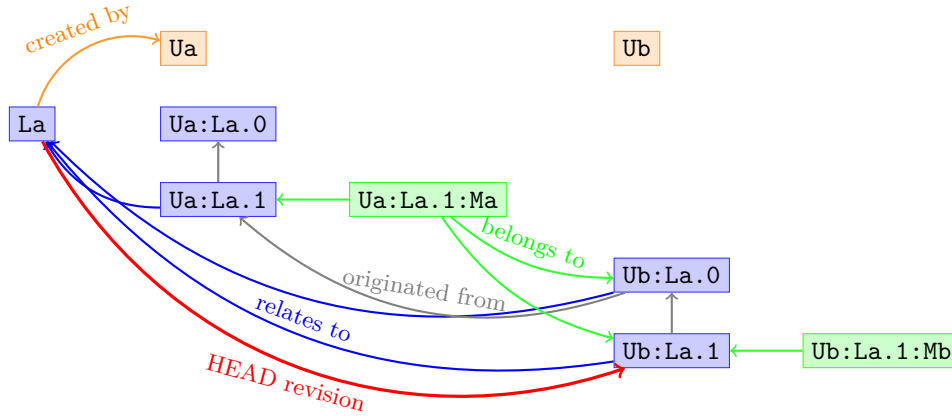
$Ua:La.2 \supset \{Ua:La.1:Ma, Ub:La.2:Mb\}$ — Second revision of Library La saved by Ua contains Ma saved by Ua in his La 's first revision and Mb saved by Ub in his second La 's revision.

¹PackageRevision is not the same as Package version. The latter is just meta-data, a text field of PackageRevision object used only in exported XPI. It will no longer be used for data identification.

²Every data object is identified by a PackageRevision. The concept is similar to *git*'s commits. In essence, for every saved Module change, a new PackageRevision is created.

2 Relations between database objects

Graph of a sample database stage for the $La \Rightarrow Ua:La.1 \supset \{Ua:La.1:Ma, Ub:La.1:Mb\}$. All objects relate to the appropriate Users.



Real world example will be more complicated. In essence a PackageRevision might (and most of the time will) be originated from more than one PackageRevisions.

3 Editing Library and its Modules

3.1 Starting point

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

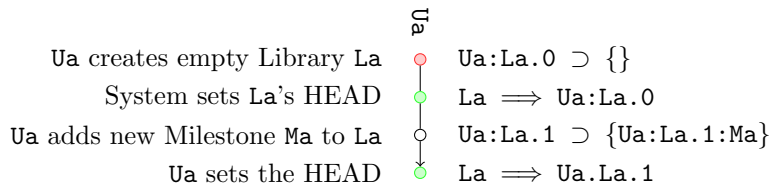
$$La \Rightarrow Ua:La.1 \supset \{Ua:La.1:Ma\}$$

Package La is created by User Ua .

La 's HEAD is PackageRevision identified as $Ua:La.1$

It contains only one module - Ma

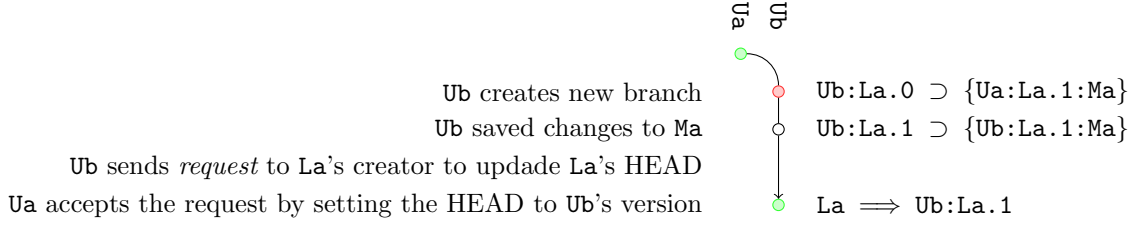
Following steps had to happen to achieve above status:



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

Ua and Ub are working on La

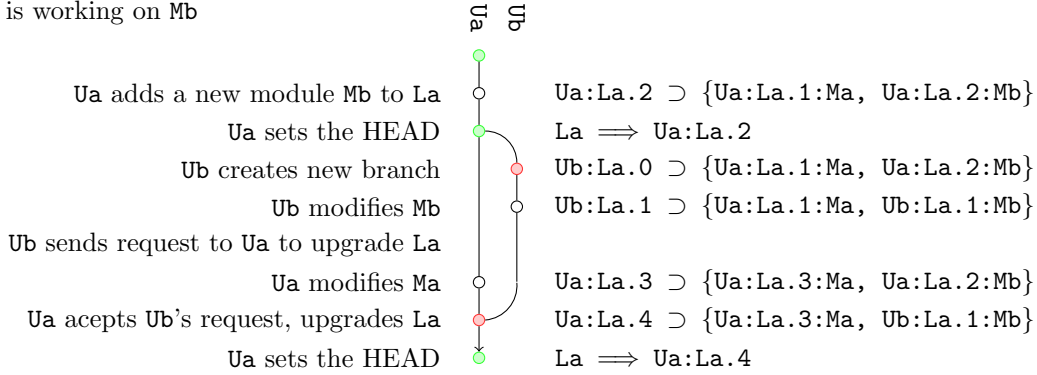
Ub modified one module



Result: La $\Rightarrow Ub:La.1 \supset \{Ub:La.1:Ma\}$

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

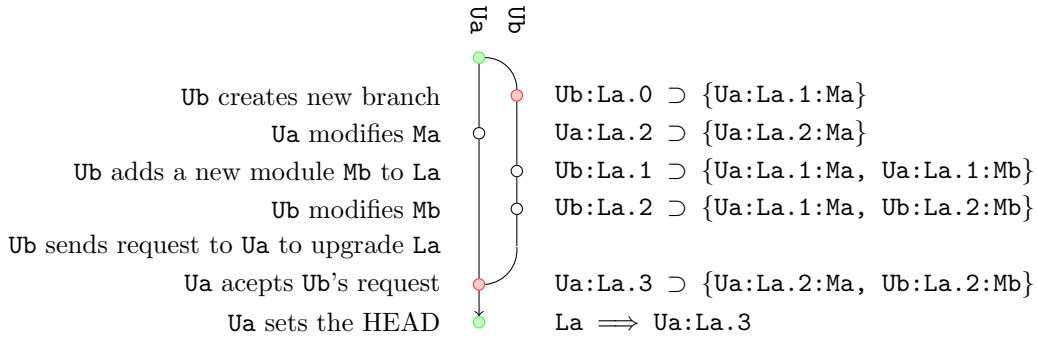
Ua and Ub are working on La
 Ua created module Mb
 Ub is working on Mb



Result: La $\Rightarrow Ua:La.4 \supset \{Ua:La.3:Ma, Ub:La.1:Mb\}$

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

Ua and Ub are working on La
 Ub created module Mb



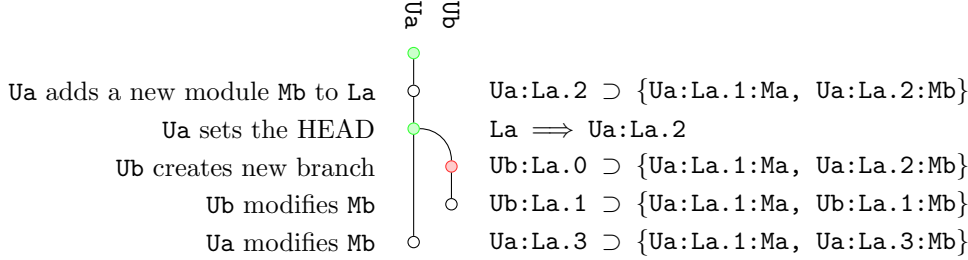
Result: La $\Rightarrow Ua:La.3 \supset \{Ua:La.2:Ma, Ub:La.2:Mb\}$

3.5 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...

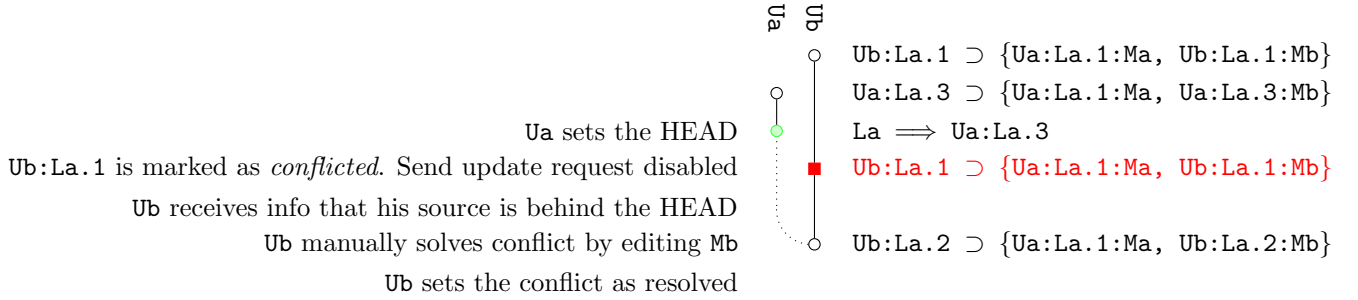
Steps leading to the conflict:



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.

3.5.1 Ua sets HEAD and Ub's revision is outdated

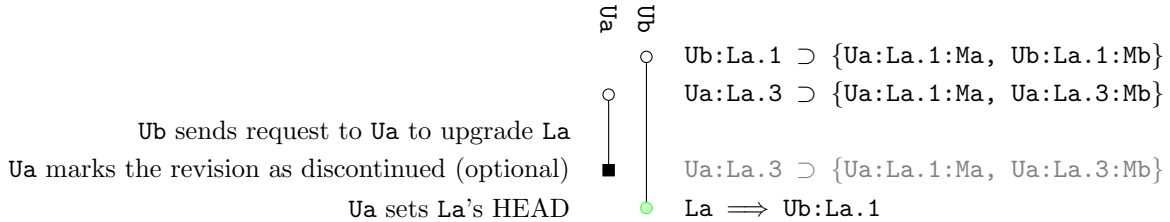
La's manager — Ua has chosen the HEAD. At that moment he doesn't know about Ub's changes to Mb.



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.³

3.5.2 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.



³Decide if this is the right thing to do.

4 Exporting XPI

Code used to export XPI⁴ may be unsaved. This happens when User will use the "Try in browser" functionality. In this case XPI may not be send to AMO⁵.

4.1 Creating directory structure

Directory structure should be as close as standard Jetpack SDK as possible.

4.2 Exporting Addon with Modules

4.3 Exporting dependencies

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

$La \implies Uc:La.3 \supset \{Ub:La.1:Ma, Uc:La.3:Mb, Uc:La.1:Mc\}$ 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...

⁴An 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.

⁵<http://addons.mozilla.org/>