Jira usage

Your Rating: Results: 102 rates

Jira usage for external developers

External developers, we welcome you to create tickets on our issue tracker at http://jira.magnolia-cms.com. You don't need to follow or read these guidelines, just describe the issue as best you can and leave any fields blank that you are uncertain of. We'll do the legwork and get back to you if we have questions.
The guidelines on this page are intended for developers working at Magnolia.

Meta

Search for an existing ticket before creating a new ticket.

Create the ticket before you start work on your task. This alerts others to the existance of the issue - and that you are working on it.

If a ticket is half-finished near the end of a sprint, but with useful and reviewable code:

- Update title if necessary to reflect work done, mark resolved and submit for review.
- Create follow up tickets for work still to do, and probably link them to this ticket.

Summary and description

It is very important that the **title** ("summary" in Jira terms) is short enough, and at the same time precise and long enough to convey all necessary information. Keep in mind that the summary is what appears in the **change log**, so it has to be clear. If anyone looking at the change log has to go to each issue's details to understand what changed and why, we missed our goal.

In most cases, the summary should not contain the technical details of the change, but rather the use-case or a short description of the problem. ("ability to do xyz", "abc does not work when def", rather than "fix class a.b.c to use d.e.f instead of u.v.w")

The **description** is where one can go at length describing the problem, a possible solution, and technical details about either.

The sum of the summary and the description should tell

- why things are changed (with a typical use-case)
- what is being changed, added, removed

When resolving an issue, it's always nice to leave a little comment telling what has changed where. Revision numbers are superfluous since jira is linked to svn, but specifying this has been applied to the trunk and/or to certain branch(es) is nice; if we're talking about a technical change, a 3-word sentence saying what class was impacted is also good.

It often happens that when reporting an issue, the conditions above can't all be met. That's entirely acceptable, since issues can be edited. When resolving, or as soon as the problem/solution has been identified, please edit and complete the summary and description.

Use the "in progress" state

If you start looking into an issue, even if no concrete work is involved, mark the issue as "in progress". This will help, when approaching release time, knowing what we can bump to the next version or what we should finish. In progress issues are saying "hey, we've started something here, please double-check me before bumping or releasing".

Leave the issue "in progress" even if you're not touching it for days or weeks. Someone might take over. If part of the issue is solved, you might want to consider splitting it into several issues: resolve the current one and bump the (linked) one to the next release, for example.

Link issues

Use and abuse the issue linking feature of Jira. Add links even if you refer to another issue in a comment. Use the appropriate relationship descriptor.

Having the links help quickly checking what's left to do, what should be done before fixing an issue, etc. Anyone who wants to know more about a certain issue can check the links without having to do a search and thus get a broader view of the problem at hand.

Another advantage of linking issues is that when **changing** behaviour, we can refer to the original behaviour's bugfix or improvement. If you modify the behaviour of a component, it's very likely that the original behaviour is document through another issue; link them, and explain why the behaviour needs to change.

Labels, Components & Versions

Labels

- **next** (for preselecting tickets in the backlog and maintenance backlog for the next version, candidates for the product management)
- **quickwin** (marker for tickets which are 'easy' to fix)
- **ux** (Andreas or UX team is involved)
- **architecture** (Gregory or Architecture team is involved)
- **maintenance** (tickets which are not part of the current maintenance version but are worked on by the maintenance team)
- **support** (tickets which are linked to support issues)

Components

- each ticket needs at least one component set
- Request the module owner to create a component if you feel one is missing

Versions

- 5.2.1
    - planned maintenance release
    - selected from the candidates
    - once in the version we do our best to ship it
- 5.2.x
    - maintenance backlog
    - label proposed candidates with next
- 5.3

    - planned major version
- Backlog
    - feature we want to add, mainly as stories and epics, label proposed candidates with next
- Ideas
    - proposals, ideas, still missing the OK of the product management
- no fix version (and open)
    - this is the inbox
    - needs to be processed
    - should be 0 tickets after the processing

Do not create versions for milestones, beta releases and release candidates, these are managed by Sprints.

Semantics of statuses

- **Resolve**: concrete work was involved and the issue is not an issue anymore. In this case, a "fix version" must be set.
- **Close**: issues are closed in 2 cases:

1. the resolution was "accepted" (by the original request, by a reviewer, by the team) and closing the issues states this. We usually don't do this at Magnolia, but it happens. Like a resolved issue, there must be a "fix version".
2. the issue is "invalid". We typically do this for "won't fix", "not an issue" and "duplicate" issues. In these cases, no concrete work (code) was involved. In these cases, there should not be a "fix version", since nothing was changed in the project for any release.

There are also cases where issues are solved by fixing another project. Typically, an issue is reported against a given module (either because the reporter didn't realize it wasn't specific to the module, or because it's only reproducible via this module). If the issue is moved to the appropriate module, nothing special. If on the other hand we create another issue in the impacted module (can be a good thing, to have a more precise description, for instance, etc), the issues 1) must be linked (duplication or dependency) 2) the original issue can usually be "closed" with "duplicate": in that case, no fix version for the original issue, but at least a comment saying "this was fixed in project X for version Y".

The **resolution** field's value is also very important, and one needs to pay attention that the combination of the status (resolved, closed) and resolution (fixed, won't fix, outdated, ...) makes sense.

Review-and-close an issue

When reviewing an issue for validation, there are multiple thing to take into account:

- Code review. This is essential, of course. The reviewer should make sure she understands the context of the change, and the change itself.
- Code quality. Is the new code tested ? Is the new code readable, clear, documented ?
- The Jira issue itself. Before closing an issue, the reviewer should make sure the issue itself is properly titled and described, such that it will be useful not only for further visits by developers, but also to non-developers looking for change logs.
- Documentation. Check the **Release notes required** box when the issue needs to be listed in release notes and documented. The Doc Team looks for this flag.

Screenshots

When adding screenshots to Jira:

Mac users:

- apple-shift-4: make a screenshot of a selected area
- apple-shift-4 then space: screenshot of the current window (no desktop, no other apps)
- apple-shift-3: make a screenshot of the whole screen(s) (only use this if really necessary)

Everyone:

- hide your bookmarks and other tabs, nobody needs/wants to know what your favorite porn sites are.

Project categories

We use a variety of project categories in JIRA, mainly for the purpose of search and issue filters. These are the most common categories, albeit more for devops purpose, upon Project Registration.

Category	official support	open-source	notes
Magnolia Community	✅	✅	

Magnolia Enterprise Edition	✅	-	
Magnolia Integration	✅	-	further limited to integrations with contribution from vendor
Forge	-	✅	
Internal	-	-	R&D, PoCs, projects for internal organization
Projects	-	-	Services, add-on development, integrations without official support

Possible improvements

Started this page as a placeholder for listing problems with and things we could do to improve our usage of Jira. Just rough ideas to be thought about, filtered, and maybe applied one day.

- how to know which issues haven't been treated yet
    - using the Bucket plugin
    - allowing unassigned issues (so assigning to yourself or someone would clearly mean this is being taken in charge)
    - having a "reviewed by dev" flag
    - systematically commenting on new issues when reviewed. that was we only need to regularly process issues where last participant is not a dev.
- having an "idea" or "todo" issue type (to avoid many of the "we should maybe do..." issues which are being forgotten