OME Contributing Developer

unknown

Feb 01, 2023

CONTENTS

This documentation is for developers who want to contribute code to OME consortium projects. It includes internal developer practices and workflows, standard procedures for tasks such as release, and other information which may be valuable to a wider audience.

CHAPTER

ONE

CHECKING OUT THE SOURCE CODE

This section is primarily designed for the core OME developers who want to check out the main code base using Git. If you need guidance in installing, configuring and using Git, see the *Using Git* page.

1.1 Code locations

OME code is stored in multiple git repositories, each of which is available from several locations.

1.1.1 OMERO

The main repository, known as ome.git, is available from:

- https://github.com/ome/openmicroscopy
- git://openmicroscopy.org/ome.git

1.1.2 Bio-Formats

The Bio-Formats repository is available from:

- https://github.com/ome/bioformats
- git://openmicroscopy.org/bioformats.git

1.1.3 Other repositories

Each member of the GitHub openmicroscopy organization, as well as anyone else who has clicked the "Fork" button, will have their own repository. These are listed here:

- https://github.com/ome/openmicroscopy/network/members
- https://github.com/ome/bioformats/network/members

1.2 Cloning the source code

Most OME development is currently happening on GitHub, therefore it is highly suggested that you become familiar with how it works, if not create an account for yourself.

Note: There is extensive guidance on the *Using Git* page and the following examples assume you have set up your account using "gh" for your personal repositories and "origin" as the official repositories as described there.

Start by cloning the official repository for the project you want to work with e.g.:

```
git clone https://github.com/ome/openmicroscopy.git
```

Since the openmicroscopy (OMERO) repository now makes use of submodules, you first need to initialize all the submodules:

```
cd openmicroscopy
git submodule update --init
```

Alternatively, with version 1.6.5 of Git and later, you can pass the `--recursive` option to git clone and initialize all submodules:

```
git clone --recursive https://github.com/ome/openmicroscopy.git
```

Note: The use of submodules does not apply to Bio-Formats, which has all code and documentation within a single repository at https://github.com/ome/bioformats.git

The natural workflow when using GitHub is not just to download someone else's repository, but also to create a personal working copy. Go to the repository page at https://github.com/ome/openmicroscopy or https://github.com/ome/bioformats and click on "Fork". This will create a copy of the repository in your own personal space e.g.:

```
https://github.com/YOURNAME/bioformats
```

which can be added to your local repository as another remote:

```
git remote add gh git@github.com:YOURNAME/bioformats.git
```

Note: For the SSH (Secure Shell) transport to work, you will need to follow some of the instructions under https://github.com/account/ssh

Depending on which repository you cloned first, either origin/develop or gh/develop will be the "develop" branch of your own fork of openmicroscopy/openmicroscopy or openmicroscopy/bioformats. The example below assumes that "gh" is your own personal GitHub repository, and "origin" is the official openmicroscopy repository.

You may even want to remove the "develop" branch from your fork since all branching should happen from the official develop branch. If you'd prefer to keep a copy of "develop" in "gh", that is fine, but you may then need to keep your develop up-to-date with the official develop:

```
git checkout develop
git reset --hard origin/develop   # Warning: This will delete any unsaved changes and␣
↪commits to develop!
```

(continues on next page)

(continued from previous page)

```
git push -f gh develop          # Warning: This will replace gh/develop with the␣
↪official version remotely.
```

CHAPTER

TWO

USING GIT

The following is primarily designed for the core OME developers who are contributing to our code base using Git. It should contain all the useful commands and configuration you need for doing most Git tasks.

Note: This section assumes that “gh” is your own personal GitHub repository, and “origin” is one of the official openmicroscopy repositories.

2.1 Installing Git

In general, see the Git downloads page for installation options.

2.1.1 Linux

Most flavors of Linux have git available through the package manager. For example, on Debian or Ubuntu:

```
sudo apt-get install git
```

2.1.2 Mac OS X

You can install Git using Homebrew:

```
brew install git
```

Or you can use the binary installer.

2.1.3 Windows

We recommend using either Git for Windows for a basic Git installation, or Cygwin for a full-featured Unix-style environment that includes Git. You can also use TortoiseGit for Git shell integration. You may also want to consider installing VirtualBox with a Linux guest OS to make your life easier. Lastly, when using Git on Windows, please be aware of the CRLF conversion issue.

2.2 Git configuration

If you are looking to get started as quickly as possible, the minimum you will need is to have Git installed and then:

```
git config --global user.name "Full name"
git config --global user.email YOUR_EMAIL
git clone --recursive https://github.com/ome/REPOSITORY_NAME
cd REPOSITORY_NAME
```

You will not be able to push back to this repository, but you will at least have something you can start looking at.

Git provides a number of options which can make working with it considerably more pleasant. These configuration options are available either globally in `$HOME/.gitconfig` or in the `.git` directory of your repository. The file is in INI-format, but can also be modified using the `git config` command, as illustrated in the examples following.

The most important thing is to update your 'global' credentials that are used in your commits. These values are saved in `~/.gitconfig`:

```
git config --global user.name "Full name"
git config --global user.email YOUR_EMAIL
```

If you have a PGP key for signing commits and tags, you may want to add it as well:

```
git config --global user.signingkey YOUR_PGP_KEY_ID
```

Color and display options make log and diff output much more friendly:

```
git config --global color.ui true
git config --global color.diff auto
git config --global color.graph auto
git config --global color.status auto
git config --global color.branch auto

git config --global core.ui always
git config --global core.editor mate_wait
```

Aliases provide a way to make shortcuts for longer Git commands. One that is often used among the OME team is `graph`:

```
git config --global alias.graph "log --date-order --graph --decorate --oneline"
```

See Helpful command aliases for more examples.

2.3 Interacting with GitHub

2.3.1 Cloning the repositories

You can fork any of the openmicroscopy repositories you will be working on by clicking the fork icon in the top righthand corner of each repo's homepage on GitHub. This will give you your own copy of the repo on GitHub. To set this up from the command line so you can push to it and open PRs, you need to clone the repo. The following example uses the documentation repo:

```
git clone https://github.com/ome/ome-documentation
cd ome-documentation
git remote add gh git@github.com:YOUR_USERNAME/ome-documentation.git
```

To clone private repositories you need to use the SSH protocol:

```
git clone git@github.com:openmicroscopy/REPO_NAME.git
```

2.3.2 GitHub remotes

You can add the other members of the OME network as remotes, so you can follow what they are doing:

```
git remote add SOMEUSER git://github.com/SOMEUSER/openmicroscopy.git
git fetch SOMEUSER
```

If you would like to work more closely with someone, via pushing directly to their branch or they from yours, then you can have them add you as a collaborator on their repository or do the same for them on yours. This is done under https://github.com/account/repositories

If you have not made such a repository yet as a remote, then you should do so using the SSH protocol:

```
git remote add SOMEUSER git@github.com:SOMEUSER/openmicroscopy.git
```

Otherwise, you will need to modify its URL

```
git remote set-url SOMEUSER git@github.com:SOMEUSER/openmicroscopy.git
```

If you would like to be kept up-to-date on what several users are doing on GitHub, you can set the "default remotes" value to the list of people you would like to check in .git/config:

```
git config remotes.default "ome team origin gh official chris ola will jm colin"
```

Now, `git remote update` will check the above list of repositories.

2.3.3 Pushing to GitHub

When you have work which you want to share with the rest of the team, it is vital that you push it to your GitHub fork.

```
git push gh your-branch
```

This will create a new branch, and the same command can be used to subsequently update that branch.

If you NEED to use a different name for the branch on GitHub, you can do:

```
git push gh your-branch:refs/heads/branch-name-on-gh
```

As mentioned elsewhere, the "refs/heads/" prefix only needs to be used to create a new branch, and can be dropped for subsequent pushes.

2.3.4 Tracking others' branches

The flip-side of pushing your own branches is being aware that other OME developers will also be pushing theirs. GitHub provides a number of ways of monitoring either a user or a repository. Notifications about what watched users and repositories are doing can be seen in your GitHub inbox or via RSS feeds. See Be social for more information.

Even if you do not feel able to watch the everyone's repository, you will likely want to periodically check in on the current Pull Requests (PRs). These will contain screenshots and other updates about what the team is working on. When the PRs have been sufficiently reviewed, they will be merged into the develop branch so that others' work will start to be based on it.

2.3.5 Cleaning up your GitHub branches

Once your branches have been merged into the mainline ("develop" of openmicroscopy/openmicroscopy) you should delete them from your repository.

```
git branch -d your-branch
git push gh :your-branch
```

This way, anyone looking at your fork clearly sees what is currently in progress or upcoming.

2.4 Common Git Commands

Although everyone has a slightly different way of working, the following command examples should show you much of what you will want to do on a daily or weekly basis when working with OME via Git.

See if you have any changes that you might need to commit. This also displays some useful tips on how to add and remove files:

```
git status
```

Create a branch from the "develop" branch:

```
git checkout -b feature/foo origin/develop
```

At this point, you are ready to do some work:

```
git checkout my-work  # Just to be sure.
vim README.txt        # edit files
git merge anotheruser/some-work
git status            # See what you have done
```

You can also add files or directories to the 'cache' with interactive choice of which 'chunks' to accept or decline (useful for checking that you are not adding any unintended changes, print statements etc.):

```
git add -p path/to/dir/or/file
```

Check the status again - to see summary of what you are about to commit:

```
git status
```

Any remaining changes that you want to discard can be reverted by:

```
git checkout -- path/to/file.txt
```

When committing the code you have just modified/merged your commit message should refer to related tickets. E.g. “See #1111” will link the commit to the ticket on trac, and “Fixes #2222”” will link and close the ticket on trac.

```
git commit -m "Add message here and refer to the ticket number. See #1234. Fixes #5678"
```

Note: If you want to add more than a short one-line message, you can omit the -m “message” and Git will open your specified editor, where you should add a single line summary followed by line space and then a paragraph of more text. See *Commit messages* for more discussion.

After you have committed, you can keep working and committing as above - the changes are only saved to your local git.

For example, you can move to another branch to continue work on a different feature. To see a list branches:

```
git branch
```

Add the `-a` to list remote branches too.

To simply move between branches, use `git checkout`. All the files on your file-system will be updated to the new branch:

```
git checkout dev_4_2
```

Note: Make sure they are refreshed if you have files open in an editor or IDE

If you have forgotten what you did on a particular branch, you can use `git log`. Add the `-p` flag to see the actual diff for each commit.

You can use the first 5 characters of a commit’s hash key to begin the log at a certain commit. E.g. show diff for commit 83dad:

```
git log 83dad -p
```

Or to display a nice graph:

```
git log --graph --decorate --oneline
```

If an alias has been set-up as described in the configuration section above, you can just do:

```
git graph
```

This is most useful when showing how two branches are related:

```
git graph origin/develop develop
```

When you are ready, you will need to push your local changes to your own forked repository in order to share with others. If the branch does not yet exist in your repository, you will need to prefix the push command with `refs/heads`:

```
git push gh my_fix_123:refs/heads/my_fix_123
```

After that initial push, the following will suffice as long as you are on the my_fix_123 branch:

```
git push gh my_fix_123
```

You will find it easier if you name remote branches the same as local branch though it is not a requirement:

```
git push gh name/of/branch:refs/heads/name/of/branch
# E.g:
git push gh feature/export:refs/heads/feature/export
```

Once you have pushed, you can open a "Pull Request" to inform the team about the changes. More on that below.

You can also create a local branch from a remote branch, whether it is your own or belongs to someone else on the team. These will be 'tracked' so that commits you push automatically go to the corresponding remote branch:

```
git fetch SOMEUSER && git checkout -b name/of/branch SOMEUSER/name/of/branch
# work on the branch then:
git push SOMEUSER name/of/branch
```

2.4.1 Collaborating via git rebase

If you have been permitted write access to someone else's forked repository, or you have granted someone else write access to your repository, then there is a further aspect that you need to be aware of.

If both of you are working on the my_fix_123 branch from above, then when it is time to push, your version may not represent the latest state. To prevent losing any commits or introducing unnecessary merge messages, you will first need to access the latest remote changes:

```
git fetch gh
```

To see the differences between your local changes ('my_fix_123') and the remote changes ('gh/my_fix_123'), you can use the log command:

```
git log --graph --date-order gh/my_fix_123 my_fix_123
```

If the remote branch ('gh/my_fix_123') have moved ahead of yours, then you will want to rebase your work on top of this new work:

```
git rebase gh/my_fix_123
```

Now your local changes will follow the remote changes in the log. You can check how this looks by viewing the graph again:

```
git log --graph --date-order gh/my_fix_123 my_fix_123
```

Now you can push your changes on the 'my_fix_123' branch to the remote repository:

```
git push gh my_fix_123
```

Rebasing allows you to update the 'base' point at which you branched from another branch (as described above). You can also use 'rebase' to organize your commits before merging.

It can strip whitespace, since it is good practice not to commit extra whitespace at the end of lines or files. Git allows you to remove all extra whitespace during rebase e.g. to origin/develop branch

```
git rebase --whitespace=strip origin/develop
```

Rebase "interactive" using the `-i` flag allows you to remove, edit, combine etc commits. Git will open an editor to allow you to edit the commit summary along with instructions on how to omit, modify commits. For example, to rebase onto origin/develop branch:

```
git rebase -i origin/develop
```

2.5 Working with submodules

Since submodules are git repositories, all the tools described previously (add remotes, edit/merge, commit…) can be used within each submodule repository:

```
$ cd components/bioformats
$ git remote add melissalinkert git@github.com:melissalinkert/bioformats.git
$ git remote
origin
melissalinkert
sbesson
$ git checkout -b new_branch origin/develop
$ vim Readme.txt
$ git merge melissalinkert/branch
$ git commit -m "Merge branch"
$ git push sbesson new_branch
```

Additionally, you can perform an update of the submodule from the parent project, i.e. checkout a specific commit. After updating, the submodule ends up in a detached HEAD state:

```
$ cd code/openmicroscopy
$ git submodule update
Submodule path 'components/bioformats': checked out
↪'9328b869b9ba61851adaa3db428ce25f0ca56845'
$ cd components/bioformats
$ git branch
* (no branch)
  develop
```

If you move between branches in the project, you may end up in a different state of the submodule:

```
$ cd ../..
$ git checkout my-branch
M   components/bioformats
Switched to branch 'my-branch'

$ git status
# On branch my-branch
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   components/bioformats (new commits)
#
# no changes added to commit (use "git add" and/or "git commit -a")
```

If you do not want to modify the submodule state, run **git submodule update**. Be careful though, the **git subdmodule update** command will silently delete all local changes under the submodule. If you want to keep your changes, make sure you have pushed them to GitHub.

To advance the submodule to another commit, you can run the **git add** command:

```
cd components/bioformats
git merge gh/branch
git commit -m "Merged branch"
git push

cd ..
git add bioformats
git commit -m "Move to latest bioformats"
```

Warning: Be careful NOT to add a trailing slash when adding the submodule, the following command would want to delete the submodule and add all the files in the submodule directory:

```
git add components/bioformats/
```

There are Git hooks available to make working with submodules safer. See post-merge-checkout for an example.

2.6 Commit messages

All commit messages in git should start with a single line of 72 characters or less, following by a blank line, followed by any other text.

```
Add feature X (See #123, Fix #321)
<this line left blank>
More description about X. It's really great ...
```

Many git tools expect exactly this format, not the least of which is GitHub. If you would like to see how these commit messages are rendered on GitHub, take a look at the repository https://github.com/kneath/commits

You can read more about commit messages at A Note About Git Commit Messages.

2.7 Rebasing to keep code clean

Rebasing allows you to update the 'base' point at which you branched from another branch (as described above). You can also use 'rebase' to organize your commits before merging.

- Strip whitespace: It is good practice not to commit extra whitespace at the end of lines or files. Git allows you to remove all extra whitespace during rebase - E.g. to origin/develop branch

  ```
  git rebase --whitespace=strip origin/develop
  ```

- If you used the set-up script above, the alias 'ws' was added to allow you to achieve the same action with:

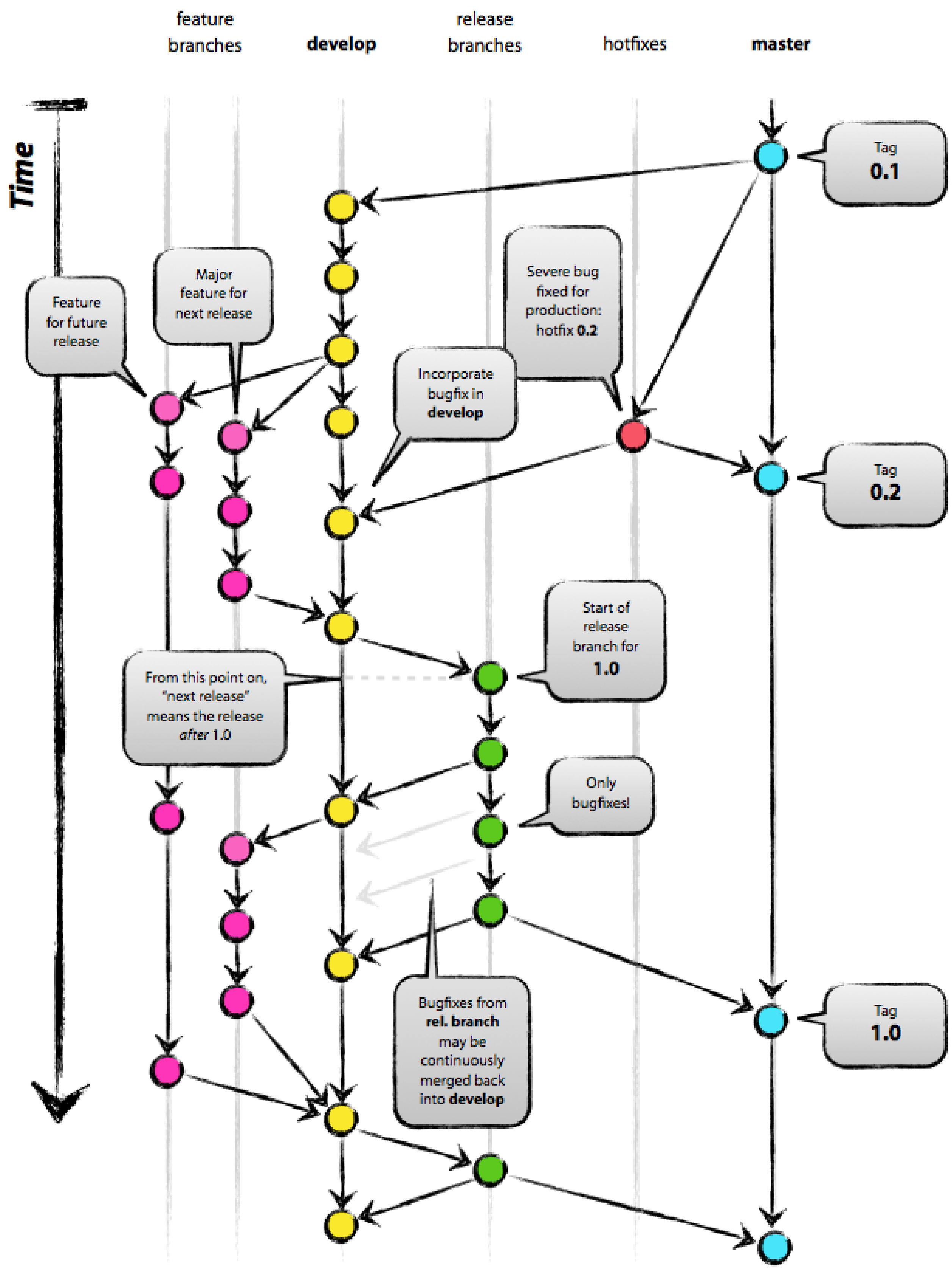
  ```
  git ws origin/develop
  ```

- Rebase "interactive": To remove, edit, combine etc. your commits, use the `-i` flag. Git will open an editor to allow you to edit the commit summary (gives instructions too). For example, to rebase onto origin/develop branch:

```
git rebase -i origin/develop
```

2.8 Branch naming

We roughly follow the git-flow style of naming and managing branch. Info about the idea can be found under A successful Git branching model. There is also a screencast available on Vimeo.


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Severe bug fixed for production: hotfix 0.2
Major feature for next release
Feature for future release
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

The master branch is always “releasable”, almost always by having a tagged version merged into it. The develop branch is where unstable work takes place. At times, another stable branch with the version name appended ("dev_x_y") is also active. PRs merged into this stable branch are also rebased onto develop.

For more information about how multiple branches are being maintained currently, see *Continuous integration*.

2.9 Advanced: Branch management

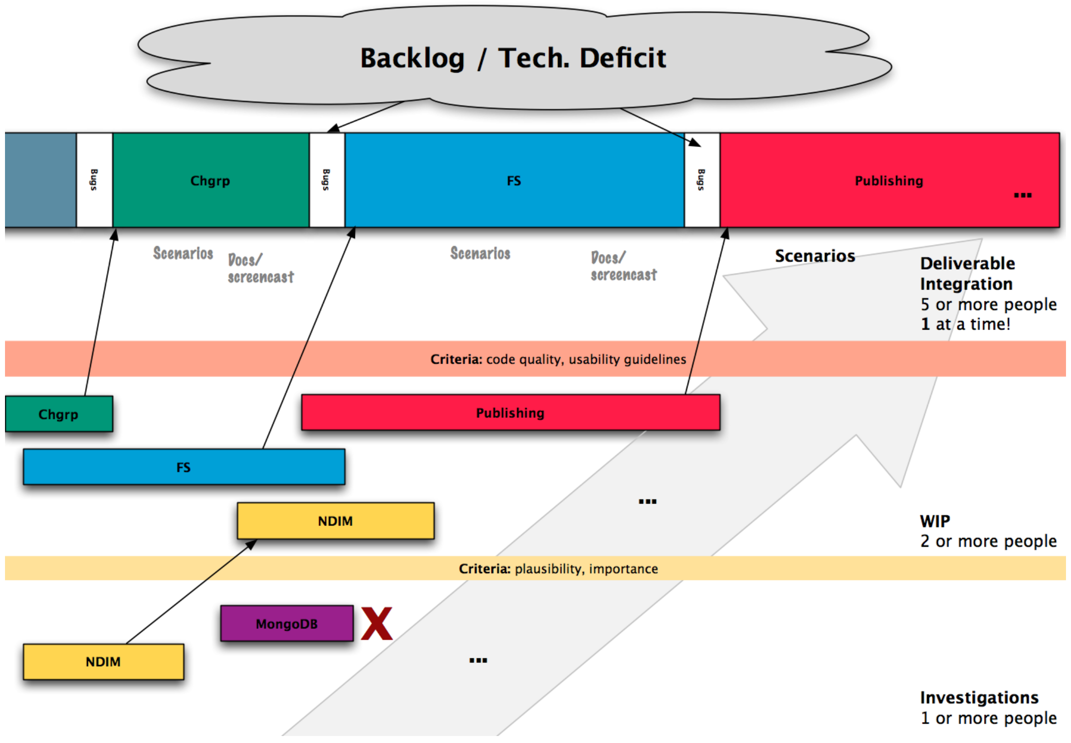
One large goal of the work with the forked repository model is to have both team members as well as external collaborators be aware of upcoming features as they happen, and have them be able to comment on the work as quickly and easily.

There is a danger of some members of the community not being aware of which branches are active and applicable, but if our weekly meetings contain a summary of what work is happening in which branch as opposed to just which tickets are in progress on the whiteboard, then it should be fairly easy for someone from the outside to get involved.

What follows is an explanation of the overarching way we categorize and review our branches. This is not required reading for everyone.

2.9.1 Branch types

To make working with a larger number of branches easily, we will initially introduce some terminology. Branches should typically be in one of three states: investigations, works-in-progress (WIP), or deliverables.



Investigations

At the bottom of the figure above are the investigation branches. These are efforts which are being driven possibly by a single individual and which are possibly not a part of the current milestone. They may not lead to released code, or they may be put on hold for some period of time while other avenues are also investigated.

WIPs

For an investigation to move up to being a work-in-progress, it should have more involvement from the rest of team and have been discussed and documented via stories, mini-group meetings, etc. Where necessary – which will usually be the case – the major components (Bio-Formats, the model, the database, the server, at least one client) should be under way.

Deliverables

Finally, deliverable branches are intended for inclusion in the upcoming milestone. They have all the necessary “paper work” – requirements, stories, tasks, scenarios, tests, screencasts, etc. Where support is needed to get all of the pieces in place, the rest of the team can be involved. And when ready a small number (mostly likely just one) will be finalized and merged into “develop” at a time. This represents the post-sprint “demo” concept that has been discussed elsewhere.

The backlog

One non-branch type that should also be kept in mind is the backlog. Between major deliverables and while a WIP is being ramped up to a deliverable, the backlog should be continually worked on and the fix branches also merged in once tested and verified.

2.9.2 Branch workflow

With the definitions, we can walk through the progression of a branch from inception to delivered code.

First, someone, perhaps even an external collaborator, creates a branch, typically starting from master or develop (having them branch from the mainline should hopefully makes things easier later on). Work is first done locally, and then eventually pushed to github.com/YOURUSER/openmicroscopy. If you have given access to particular members of the team, then they may want to work directly on that branch. Alternatively, they may create branches from your branch, and send you commits – either via Pull Request or as patches – for you to include in your work.

It is advisable to keep the OME team in the loop about your work as it progresses, e.g., by tagging ome on the forum or by opening a Pull Request.

After it is clear that there is some interest in your investigation branch, then the related stories and possibly requirement should be flushed out. The design of the work should be checked against the other parts of OME. For example, a GUI addition should fit into other existing workflows, and the implications on the other client (i.e. OMERO.web’s impact on OMERO.insight, or the other way around) should be evaluated.

At this point, the branch will most likely be considered a work-in-progress and will need to start getting ready for release. The various related branches will need to be kept in sync. Whether through a rebase or a merge workflow, all involved parties will need to make sure they regularly have an up-to-date view of the work going on.

For example, the “remotes.default” has been configured as above, a sensible thing to do every morning on coming to work is to run:

```
git remote update
```

and see all changes that the team have made:

```
~/git $ git remote update
Fetching team
Fetching origin
remote: Counting objects: 22, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 7), reused 0 (delta 0)
Unpacking objects: 100% (8/8), done.
From ssh://lust/home/git/omero
3f2ab6f..f80cbc4  dev_4_1_jcb -> origin/dev_4_1_jcb
Fetching gh
...
Fetching jm
remote: Counting objects: 46, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 24 (delta 19), reused 24 (delta 19)
Unpacking objects: 100% (24/24), done.
From git://github.com/jburel/openmicroscopy
* [new branch]      feature/plateAcquisitionAnnotation -> jm/feature/
↪plateAcquisitionAnnotation
Fetching colin
From git://github.com/ximenesuk/openmicroscopy
* [new branch]      909-Proposal2 -> colin/909-Proposal2
```

If you want to get the changes for all submodules, you can use:

```
git submodule foreach --recursive git remote update
```

At this point, you may need to "merge –ff-only" or just "rebase" your work to incorporate the new commits:

```
git checkout 909-Proposal2
git show-branch 909-Proposal2 colin/909-Proposal2
git rebase colin/909-Proposal2
```

Finally, the WIP branch will have advanced far enough that it should be made release-ready, which will need to be discussed at a weekly meeting. Often at this point, the involved developers will need help from others getting the documentation, the testing, the screencasts, the scenarios, and all the other bits and bobs (the "paper work") ready for release.

One at a time (at least initially), WIP branches will be picked and made into a deliverable. At this point, several people will have looked over the code and all the paper work, and the whole team should feel comfortable with the release-state of the branch. At this point, a Pull Request should be issued to the official openmicroscopy/openmicroscopy repository for the final merge. All the related branches in each individual's repository can now be deleted.

A major benefit of having the paper work per deliverable done immediately is that if it becomes necessary the mainline, i.e. the "develop" branch of openmicroscopy/openmicroscopy, could be released far more quickly than if we have several deliverables simultaneously in the air.

Merge branches

A significant disadvantage to having separate lines of inquiry in separate branches is the possibility that there will be negative interactions between 2 or more branches when merged, and that these problems won't be found until late in development. To offset this risk, it is possible and advisable to begin creating "temporary merge branches" earlier in development.

For example, if we assume that two of the branches from the `git remote update` command from above are intended for release fairly soon:

- jm/feature/plateAcquisitionAnnotation
- colin/909-Proposal2

Then we can create a temporary test branch:

```
git checkout -b test-909-and-plate origin/develop
git merge --no-ff jm/feature/plateAcquisitionAnnotation
git merge --no-ff colin/909-Proposal2
```

and build and test this composite. This need not be done manually, but assuming there's a convention like "all branches for immediate release are prefixed with 'deliverable/' ", then a jenkins job can attempt the merge, failing if it is not possible, and run all tests if it succeeds. Any weekly testing we do can use the artifacts generated by this build to be as sure as possible that nothing unexpected has leaked in.

Code reviews and comments

On the flip-side, a major advantage to having the above branching workflow is that is far easier to review the entire impact and style of a deliverable before it is integrated into the mainline. Any commit or even line which is being proposed for release can be commented as shown on https://github.com/features/code-review

If you would like to include other users beyond just the branch owner in the discussion, you can use a twitter-style name to invite them ("@SOMEUSER"): https://github.com/blog/821-mention-somebody-they-re-notified

Pull Requests

Several times above "Pull Requests" (PR) have been mentioned. A Pull Request is a way to invite someone to merge from one repository to another. If the commits included in the PR can be seamlessly merged, then the target user need only click on a button. If not, then there may be some back-and-forth on the work done, similar to the code reviews of a deliverable branch. For background, see

- About Pull Requests
- Pull Requests 2.0

If you have discovered that something in the proposed branch needs changing (and you do not have write access to the branch itself), then you can checkout the branch, make the fixes, push the branch, and open a Pull Request.

```
git checkout -b new_stuff SOMEUSER/new_stuff
# Modifications
git commit -a -m "My fix of the new_stuff"
git push gh new_stuff
# Now go to the new_stuff branch on github.com and open the PR
```

GitHub's "Open a pull request" page invites you to leave a comment under the PR title: we use this comment to describe the PR. A good choice of PR title and description are both helpful to reviewers of your work. For the PR description there may be template text already provided for you to edit. If so then do consider what it says but also feel free to change that template as much as makes sense for describing your PR.

Pull Request conflicts

When issuing a pull request, usually you will the following message “This pull request can be automatically merged”. If this is not the case, follow a possible workflow to fix the problem. For the sake of this example, *bugs* is the branch we are working on:

```
# push the branch to GitHub
git push gh bugs:refs/heads/bugs
# issue a pull request, not possible to merge due to a conflict.
```

Now we need to fix the conflict:

```
# checkout your local branch
git checkout bugs
# fetch and merge origin/develop
git fetch origin
git merge origin/develop            # Any conflicts will be listed
# Edit the conflicting files to fix conflicts, then
git add path/to/file
git commit                          # Use the suggested 'merging...' message
git push gh bugs
```

Your branch should now be able to merge back into develop. This should only be done at the very end of a pull request just before it is merged into origin/develop. Multiple “pull origin/develop” messages in a branch would be very bad style.

2.10 Git resources

- Pro Git book
- https://git-scm.com/book/ch3-6.html
- A successful Git branching model

CHAPTER

THREE

CODE CONTRIBUTIONS

In order to expedite the contribution of code to the OME project, whether individual files or entire modules such as a service or web application, we have put together the following guidelines. If you have issues with any of the below, please let us know.

3.1 File headers

The official header templates for each file type (Java, Python, HTML, etc.) can be found in the docs/headers.txt file of the source repository. The correct template should be applied at the top of all newly created files. The header of existing files should not be modified without previous discussion except with regard to keeping the year line up to date, for example changing “2008-2011” to “2008-2013”.

3.2 Character encoding

OME Python and Java source files are all encoded in UTF-8.

3.3 Code style and linting

Code styling can be a matter of intense debate. We are in the process of introducing auto-formatters to most of our repositories to reduce the time wasted on formatting code or discussing code styles. Where possible pre-commit is used to manage auto-formatters such as black (Python), as well as linters such as flake8 (Python).

3.4 Copyrights

The copyright line for a newly created file is based on the institution of the creator of the file and will remain unchanged even if copied or moved. Before redistribution of code can take place, an agreement must be reached between the OME team and the copyright holder.

3.5 Licenses

The licenses of any files intended for redistribution with OME must be compatible with the GPL and more restrictively for the web components with the AGPL. Some files in the code-base (the schema, etc.) are released under more liberal licenses but are still compatible with the GPL.

3.6 Distribution

For a block of work to be considered for redistribution with OME, the code must further be made available in one of the following formats.

3.6.1 Patches/Pull requests

Smaller changes to the existing code base can be submitted to the team either as patches, or preferably as pull requests on GitHub. You can read more about pull requests on the *Using Git* page. The idea is that such smaller changes are reviewed line-by-line and then maintained by the core team.

3.6.2 Submodules

Larger submissions, which cannot be effectively reviewed so intensively, should be submitted as git submodules. Such submodules provide a unique way to describe to a component version, which becomes linked into the main codebase. During checkout, all submodules are downloaded into the OME directory; and during the build process, submodules are compiled into the official distribution.

The OME team cannot maintain or ship code which is only available as a long-living branch (a fork) of the code base, and we would encourage submitters to use one of the above methods.

3.7 Procedure for accepting code contributions

External contributors will need to sign our *Contributor License Agreement* in order to get their pull requests reviewed.

External pull requests will get an initial review to identify if they are suitable to pass into our *continuous integration system* for building and testing. We try to do this within 2 days of submission but please be patient if we are busy and it takes longer.

If there are any obvious issues, we will comment and wait for you to fix them. Once we are confident the PR contains no obvious errors, an "include" label will be added which means the PR will be included in the merge build jobs for the appropriate branch.

Build failures will then be noted on the PR and we will either submit a patch or provide sufficient information for you to fix the problem yourself. The "include" label will be removed until this is completed. The PR will be merged once all the builds are green with the "include" label added.

If the code you wish to submit is large enough to require its own submodule, you should contact us to discuss how we might incorporate your work into the official distribution.

3.8 Examples of contribution templates

There are any number of other projects which have set up similar practices for code contributions. If you would like to read more on the rationale, please see:

- https://incubator.apache.org/
- https://www.apache.org/foundation/how-it-works.html

See also:

https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html
Best practices for git commit message formatting

https://en.wikipedia.org/wiki/Technical_debt
Wikipedia article on Technical debt

https://prettier.io/docs/en/why-prettier.html
Benefits of using an auto-formatter to avoid debates on style

CHAPTER

FOUR

CONTRIBUTOR LICENSE AGREEMENT

Similarly to other projects like the Apache Software Foundation or the Python Software Foundation, OME uses a Contributor License Agreement (CLA) to accept code contributions. This is a legal document in which a contributor states that they are entitled to contribute their work to the project and are willing to have it used in distributions and derivative works. The CLA also ensures that once a contributor has provided a contribution, they cannot try to withdraw permission for its use at a later date. People and companies can therefore use that software, confident that they will not be asked to stop using pieces of the code at a later date.

To enter the agreement, please `the CLA form`, fill your full name, email address, the GitHub username under which you will make your contributions, date and sign the document. Finally, send the completed form to *contributors@openmicroscopy.org*.

CHAPTER

FIVE

TEAM COMMUNICATION

For anyone completely new to the project, it is most important to know how to get plugged in. There is a fairly extensive amount of communication flying around related to the project, and being able to find and track it may take some time.

5.1 Instant messaging and video conferencing

On a day-to-day level, the team meets in a Slack chatroom. Slack can be used in your internet browser or via an app; you will be invited to join the team by an admin.

The daily stand-up meeting is managed via the '#general' channel, with notes in google docs that are edited throughout the day as people complete the tasks assigned to them.

Slightly less frequently, members of the team meet on Zoom for voice discussions. These meetings are organized as needed, but should provide feedback where appropriate (tickets, notes, etc).

5.1.1 Other IM tools

Slack is the only IM tool used by the entire OME team. Some team members do also use IRC (#ome on irc.freenode.net) and **may** provide support via that channel but in general, all external requests for help are best submitted and dealt with via the forums so they are available for the whole community. In particular, the various Gitter channels associated with OME projects on GitHub are not routinely monitored and responded to.

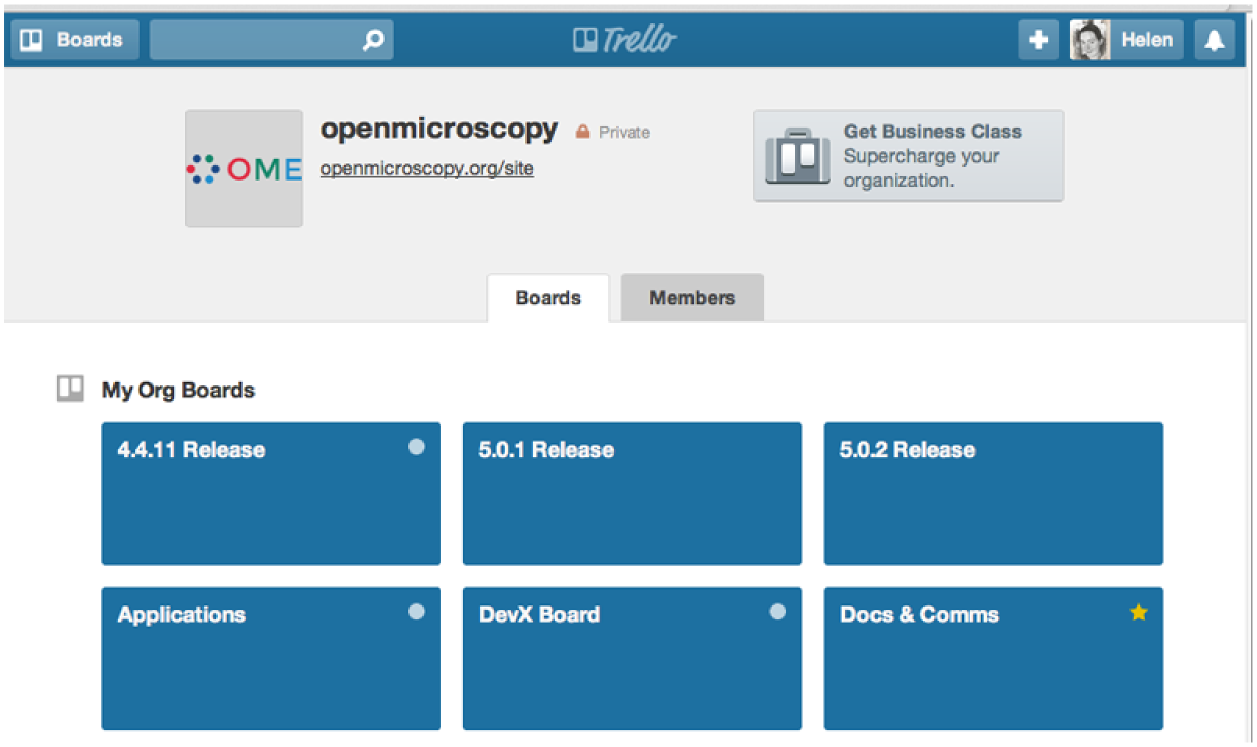
5.2 Trac

Note: The team is increasingly moving away from Trac and towards using Trello, especially for managing 'story'-level items, documentation and testing.

The Trac server is available under https://trac.openmicroscopy.org/ome and uses your LDAP account for authentication. Trac was used to record all tickets, but today is no longer actively used for new tasks and is mainly a record of older tasks.

5.3 Trello

Trello is an online organizational tool used to manage "cards" arranged in "lists" on various subject-themed "boards". This is currently the team's main internal planning tool for higher level development goals and for managing documentation, testing, and the maintenance of our continuous integration tools.



You can request access to the openmicroscopy boards as an external collaborator. Sign up for a free account and then get in touch with us to be added. We have now added a public OME organization to allow anyone to follow our development progress (see *Public-facing workflow* for more information).

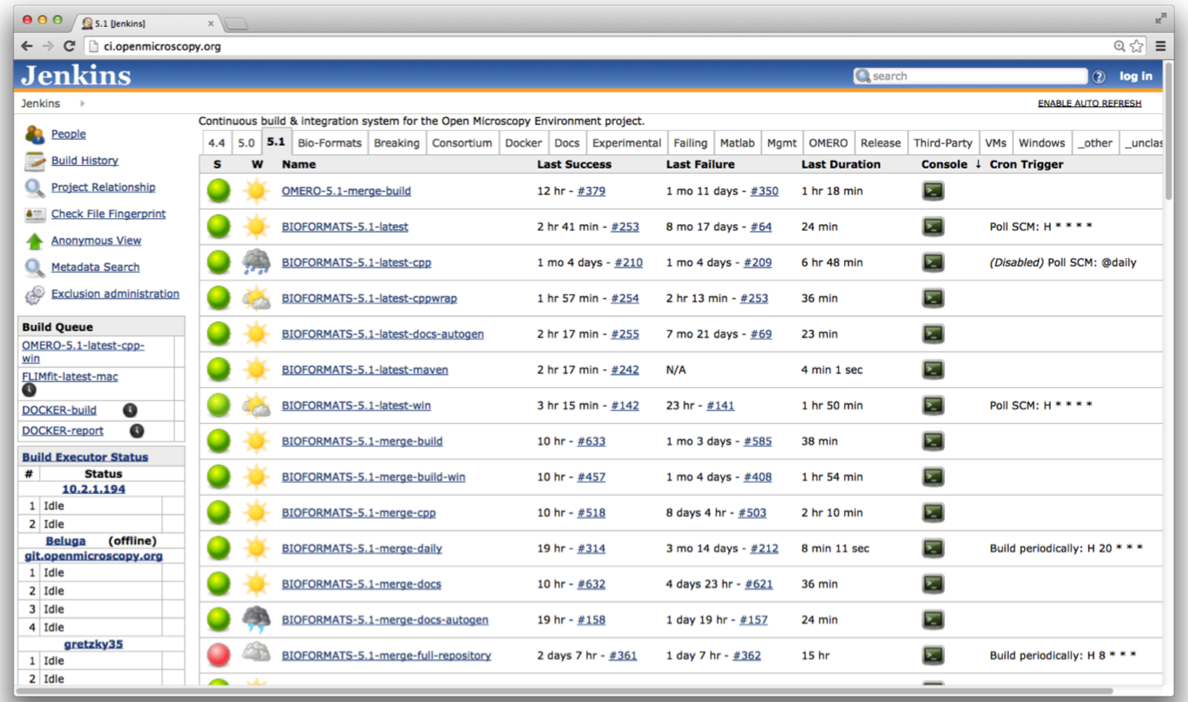
5.4 Developer documentation

The developer documentation is maintained under version control, generated using Sphinx and hosted on the OME website.

Each section of the code base (OMERO, Bio-Formats) has a landing page that will direct you to all the developer documentation that you might need. For example, the Developer page for OMERO is here.

5.5 Jenkins: Continuous integration

Our Jenkins server is available here and also uses LDAP authentication. Jenkins provides a mechanism to run arbitrary tasks ("jobs") on one or more platforms after particular events (time of day, git push, etc.) These jobs build all of the binaries released by the team, and also run automated testing.



Continuous build & integration system for the Open Microscopy Environment project.

Build Queue
OMERO-5.1-latest-cpp-win
FLIMfit-latest-mac
DOCKER-build
DOCKER-report

Build Executor Status	
#	Status
10.2.1.194	
1	Idle
2	Idle
Beluga (offline)	
git.openmicroscopy.org	
1	Idle
2	Idle
3	Idle
4	Idle
gretzky35	
1	Idle
2	Idle

S	W	Name	Last Success	Last Failure	Last Duration	Console ↓	Cron Trigger
🟢	☀	OMERO-5.1-merge-build	12 hr - #379	1 mo 11 days - #350	1 hr 18 min	▣	
🟢	☀	BIOFORMATS-5.1-latest	2 hr 41 min - #253	8 mo 17 days - #64	24 min	▣	Poll SCM: H * * * *
🟢	🌧	BIOFORMATS-5.1-latest-cpp	1 mo 4 days - #210	1 mo 4 days - #209	6 hr 48 min	▣	*(Disabled)* Poll SCM: @daily
🟢	⛅	BIOFORMATS-5.1-latest-cppwrap	1 hr 57 min - #254	2 hr 13 min - #253	36 min	▣	
🟢	☀	BIOFORMATS-5.1-latest-docs-autogen	2 hr 17 min - #255	7 mo 21 days - #69	23 min	▣	
🟢	☀	BIOFORMATS-5.1-latest-maven	2 hr 17 min - #242	N/A	4 min 1 sec	▣	
🟢	⛅	BIOFORMATS-5.1-latest-win	3 hr 15 min - #142	23 hr - #141	1 hr 50 min	▣	Poll SCM: H * * * *
🟢	☀	BIOFORMATS-5.1-merge-build	10 hr - #633	1 mo 3 days - #585	38 min	▣	
🟢	☀	BIOFORMATS-5.1-merge-build-win	10 hr - #457	1 mo 4 days - #408	1 hr 54 min	▣	
🟢	☀	BIOFORMATS-5.1-merge-cpp	10 hr - #518	8 days 4 hr - #503	2 hr 10 min	▣	
🟢	☀	BIOFORMATS-5.1-merge-daily	19 hr - #314	3 mo 14 days - #212	8 min 11 sec	▣	Build periodically: H 20 * * *
🟢	☀	BIOFORMATS-5.1-merge-docs	10 hr - #632	4 days 23 hr - #621	36 min	▣	
🟢	⛈	BIOFORMATS-5.1-merge-docs-autogen	19 hr - #158	1 day 19 hr - #157	24 min	▣	
🔴	☁	BIOFORMATS-5.1-merge-full-repository	2 days 7 hr - #361	1 day 7 hr - #362	15 hr	▣	Build periodically: H 8 * * *

5.6 Git and GitHub: Source code

Commits take place primarily on GitHub currently. To be aware of what is really going on, your best option is to become familiar with Git, GitHub, and the repositories of all the team members. Information on doing that is available in the *Checking out the source code* and *Using Git*.

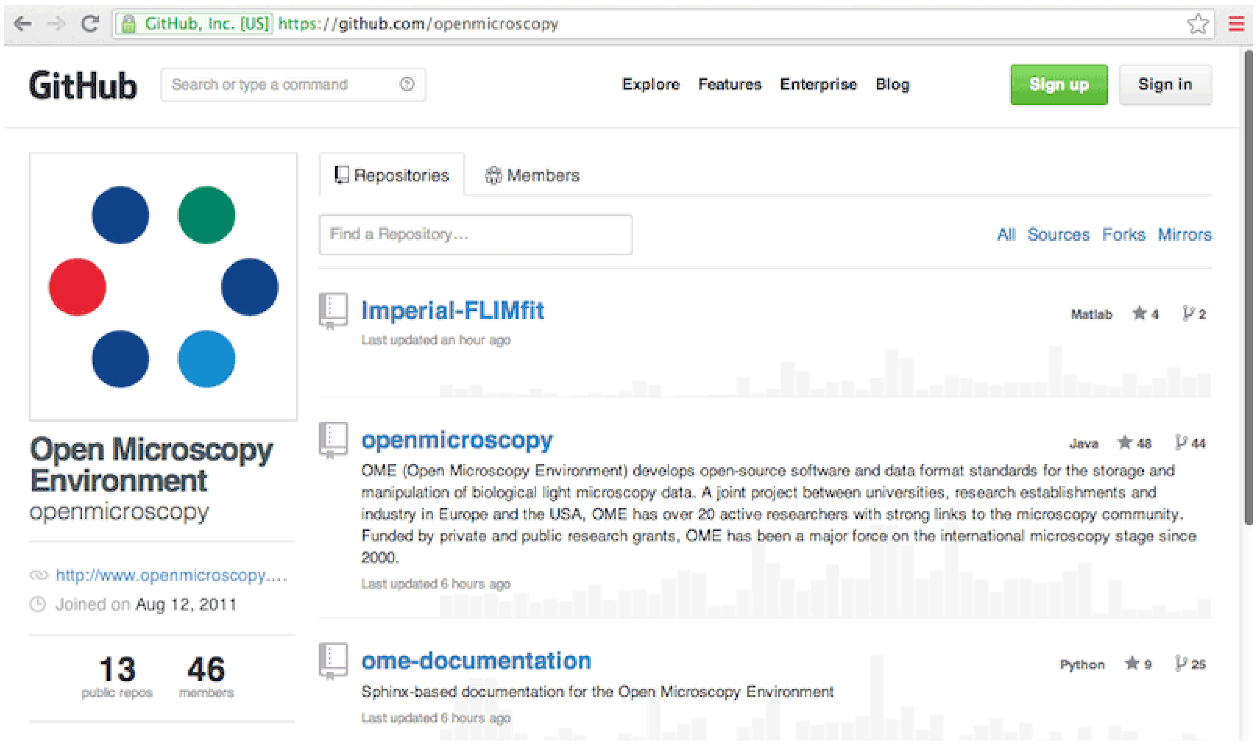
5.7 Forums and mailing list

Feedback from the OME community happens primarily on the forum as well as GitHub.

You should be aware of and scan all threads on a fairly regular basis. The general rule is that requests from the community will be responded to by the next working day, where to the best of our ability, we keep the 'working days' and time zones of the community in mind.

Where possible, the task of monitoring feedback is spread across the team. Forum questions are listed at the morning stand-up meeting and can be checked off in the accompanying notes when dealt with to ensure nothing is ignored or forgotten.

Anyone on the team should feel free to speak up to answer questions, but do try to verify the correctness of answers, code samples, etc. before posting.



Open Microscopy Environment
openmicroscopy

http://www.openmicroscopy....
Joined on Aug 12, 2011

13 public repos
46 members

Imperial-FLIMfit — Matlab ★ 4, 2
Last updated an hour ago

openmicroscopy — Java ★ 48, 44
OME (Open Microscopy Environment) develops open-source software and data format standards for the storage and manipulation of biological light microscopy data. A joint project between universities, research establishments and industry in Europe and the USA, OME has over 20 active researchers with strong links to the microscopy community. Funded by private and public research grants, OME has been a major force on the international microscopy stage since 2000.
Last updated 6 hours ago

ome-documentation — Python ★ 9, 25
Sphinx-based documentation for the Open Microscopy Environment
Last updated 6 hours ago

As much information about our activities and decision processes should be made public as possible. For many items, there is no reason to hide our process, but we do not go out of our way to make them public. For example, internally the team often uses OmniGraffle documents to illustrate concepts, but these are kept privately to prevent any confusion.

5.8 Internal servers

There are a number of servers and services inside of the University of Dundee system that are used by the entire team. You may not need access to all of them immediately, but it is good to know what is available in case you do.

- **vpn.lifesci.dundee.ac.uk** (LDAP-based) is necessary for securely accessing some of the following resources (e.g. squig, jenkins)
- **squig.openmicroscopy.org** is the shared, team-wide repository for data which can be mounted if you are on VPN or within the UoD system. It contains test data for various file formats.
- The OME QA system is an in-house system for collecting feedback from users, including failing files, stack traces, etc. Like our community feedback, QA feedback should be turned into a ticket in a timely manner.
- Home directory / data repository on necromancer (SSH-based)

Note: For anyone who has been hired to work at the University of Dundee, you will be provided with a new start tasklist which itemizes all the things that need to be done to get you set up in RL (building access, a chair, etc.).

5.9 Google Docs

In addition to the services hosted in Dundee, the team also makes use of several Google resources due to the improved real-time collaboration that they provide. A single Google collection “OME Docs” is made available to all team members. Anything placed in the collection is automatically editable by everyone.

For example, the primary contact information for all team members is available in the DevContactList spreadsheet.

You can enable notifications on the spreadsheet so that you receive an email if any changes are made.

5.10 Meetings

Weekly meetings are held online with all members of the team. Notes are taken collaboratively in a **public** Google doc in the “OME Docs > Notes > Tuesday meetings” collection. Anyone who missed the meeting is expected to review the notes and raise any issues during the next meeting.

Periodically, a technical presentation is held during the weekly meeting. This can be used to either introduce an external tool for suggested use by the team or as a peer review of in-progress work.

Mini group meetings can either be regularly scheduled (e.g. weekly) or on an as-needed basis. Notes from such meetings should be recorded in gdocs or on Trello as appropriate and if necessary matters arising should be covered in the weekly meeting for the rest of the team.

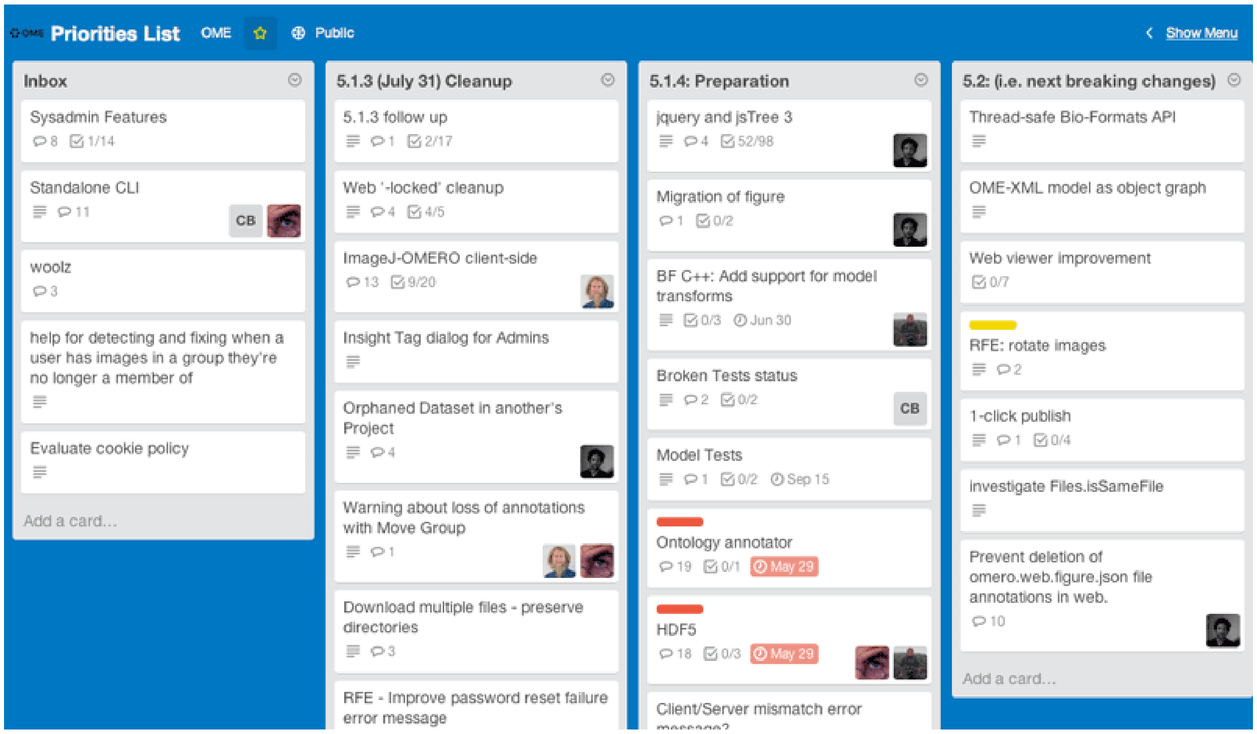
CHAPTER

SIX

TEAM WORKFLOW SUMMARY

6.1 Development management

The OME team uses a Kanban-based approach to manage development work. New features, or bug fixes which require significant amounts of work, are identified as priorities, listed in the 'Priorities List' board on Trello and then assigned to a target release. As the particular release becomes imminent, cards assigned to it are moved to a dedicated release board to allow all the current work to be tracked in one place. 'Epic' bodies of work may be managed via topic-based boards.


Priorities List   OME   Public   Show Menu

Inbox
- Sysadmin Features 8 1/14
- Standalone CLI 11 CB
- woolz 3
- help for detecting and fixing when a user has images in a group they're no longer a member of
- Evaluate cookie policy

Add a card...

5.1.3 (July 31) Cleanup
- 5.1.3 follow up 1 2/17
- Web '-locked' cleanup 4 4/5
- ImageJ-OMERO client-side 13 9/20
- Insight Tag dialog for Admins
- Orphaned Dataset in another's Project 4
- Warning about loss of annotations with Move Group 1
- Download multiple files - preserve directories 3
- RFE - Improve password reset failure error message

5.1.4: Preparation
- jquery and jsTree 3 4 52/98
- Migration of figure 1 0/2
- BF C++: Add support for model transforms 0/3 Jun 30
- Broken Tests status 2 0/2 CB
- Model Tests 1 0/2 Sep 15
- Ontology annotator 19 0/1 May 29
- HDF5 18 0/3 May 29
- Client/Server mismatch error message?

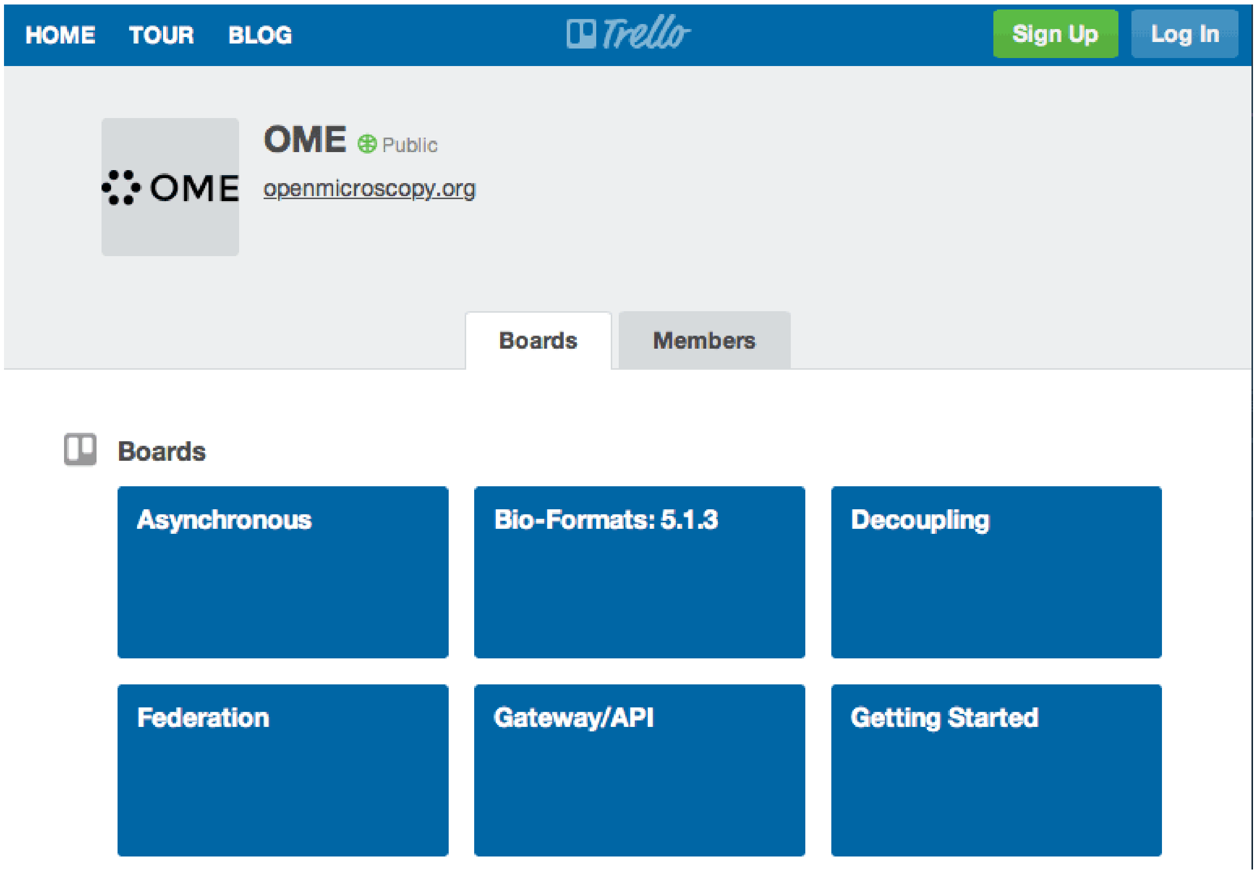
5.2: (i.e. next breaking changes)
- Thread-safe Bio-Formats API
- OME-XML model as object graph
- Web viewer improvement 0/7
- RFE: rotate images 2
- 1-click publish 1 0/4
- investigate Files.isSameFile
- Prevent deletion of omero.web.figure.json file annotations in web. 10

Add a card...

Developers should focus on the tasks identified as priorities for the next point release or longer-term work towards the next major version based on the outcome of team meetings. Constant communication is vital to ensure everyone is working together to the same goals.

6.2 Public-facing workflow

To follow the development process from outside the project, you can view the milestones on the Trac Roadmap page and Trac tickets for bugs and/or follow our new public Trello OME organization. Trello is now used for higher level planning and will be of most interest to those concerned with new features and functionality (Trac is still used for individual bug tracking). The 'Getting Started' board provides full instructions and an index of current boards, allowing you to browse upcoming work for the next releases and topic-based 'epic' work plans.



You can sign up for a free Trello account to add comments and you need to be added to the organization (by commenting on the 'Add me, please' card) to gain extra permissions e.g. to vote on or add cards.

For information about keeping up to date with OME projects, refer to the *Team communication* guide.

CHAPTER

SEVEN

ANSIBLE ROLES DEVELOPMENT

This document describes the conventions and process used by the OME team for developing, maintaining and releasing its Ansible roles.

The set of rules and procedures described below applies to the official OME roles registered in https://github.com/ome/ansible-roles.

7.1 Source code

The source code of an Ansible role should be maintained under version control using Git and hosted on GitHub under the ome organization. The Git repositories should be named as *ansible-role-<ROLENAME>*.

Each directory layout should minimally follow the standard Ansible role layout including other files and folders for testing and deployment. A typical role structure is shown below:

```
defaults/           # Default variables
handlers/           # Handlers
meta/               # Role metadata
    main.yml        # Dependencies and Galaxy metadata
molecule/           # Test
tasks/              # Main list of tasks to be executed
    main.yml
templates/          # Role templates
.travis.yml         # CI/deployment configuration file
README.md
```

7.2 Versioning

Ansible roles must follow the PEP440 scheme for versioning. Final releases must also be compliant with Semantic Versioning i.e. a final version must be expressed as *MAJOR.MINOR.PATCH* where:

- the MAJOR version must be incremented when incompatible API changes are made,
- the MINOR version must be incremented when functionality is added in a backwards-compatible manner, and
- the PATCH version must be incremented when backwards-compatible bug fixes are made.

Final releases must be tagged with a tag matching the version i.e. *MAJOR.MINOR.PATCH* with no prefix.

7.3 Testing and Continuous Integration

For each Ansible role, a `molecule` folder should be configured allowing the testing to be tested using Molecule. One or more scenarios should be configured using at least a Docker driver if possible. A generic `molecule` folder can be initialized using the following command:

```
molecule init scenario -r ansible-role-<NAME> -s default -d docker
```

Continuous Integration of Ansible roles is performed using Travis CI.

Note: OME Ansible roles are getting progressively upgraded from Molecule 1.x to Molecule 2.x. New roles must be configured using Molecule 2.x.

7.4 Distribution and support

All core OME Ansible roles should be deployed to Ansible Galaxy under the openmicroscopy organization. All roles must support RHEL/CentOS 7 as a primary platform. New roles should also include Ubuntu 18.04 as a supported platform whenever possible.

The Galaxy role name should be *openmicroscopy.<ROLENAME>*. For overriding the default name derived from the GitHub repository name, the *role_name* variable should be set in `meta/main.yml`. For role names composed of multiple words, note that the Galaxy import process will convert hyphens to underscores.

Ansible playbooks can consume these roles using a **`requirements.yml`** file - see https://github.com/ome/prod-playbooks/blob/master/requirements.yml and https://github.com/IDR/deployment/blob/master/ansible/requirements.yml for examples of such files.

The release of an Ansible role and its deployment to Galaxy release happens by triggering a role import in Galaxy using the Travis integration on each release tag.

A PGP-signed tag of form *x.y.z* should be created for the released version using **`scc tag-release`** or **`git tag -s`** and pushed to the upstream repository:

```
$ git tag -s x.y.z -m "<tag message>"
$ git push origin x.y.z
```

CHAPTER EIGHT

JAVA COMPONENTS (MAVEN)

This document describes the conventions and process used by the OME team for developing, maintaining and releasing its Java components using Maven as their build system.

The set of rules and procedures described below applies to all the following Java libraries.

Component name	GitHub URL	groupId:artifactId
OME Common Java libary	https://github.com/ome/ome-common-java	*org.openmicroscopy:ome-common*
OME Data model	https://github.com/ome/ome-model	*org.openmicroscopy:ome-model* *org.openmicroscopy:ome-xml* *org.openmicroscopy:specification* *org.openmicroscopy:ome-model-doc*
OME POI	https://github.com/ome/ome-poi	*org.openmicroscopy:ome-poi*
OME MDB Tools	https://github.com/ome/ome-mdbtools	*org.openmicroscopy:ome-mdbtools*
OME Stubs	https://github.com/ome/ome-stubs	*org.openmicroscopy:ome-stubs* *org.openmicroscopy:lwf-stubs* *org.openmicroscopy:mipav-stubs*
OME Metakit	https://github.com/ome/ome-metakit	*org.openmicroscopy:metakit*

Note: Some of the historical monolithic Java projects, including Bio-Formats and OMERO, do not strictly comply with these guidelines yet. As the project evolves and components are migrated, any new Java repository should follow this set of rules.

8.1 Conventions

8.1.1 Source code and build system

The source code of a Java library should be maintained under version control using Git and hosted on GitHub.

Maven should be used as the primary build system. The directory layout should follow the standard Maven layout i.e. in the case of a single-module project:

```
pom.xml
src/
  main/
    java/
      <package>
test/
  main/
    java/
      <package>
```

Additionally, the top-level `pom.xml` should be structured according to the Maven guidelines.

See also:

Using Git

8.1.2 Development

Java components use Semantic Versioning i.e. given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes,
- MINOR version when you add functionality in a backwards-compatible manner, and
- PATCH version when you make backwards-compatible bug fixes.

In between releases the version is bumped to the next SNAPSHOT version.

Code contributions should follow the guidelines highlighted in *Code contributions*.

8.1.3 Distribution

All the release artifacts for the repositories listed above should be deployed to the Central Repository according to the process described in the next section.

8.2 Release process

8.2.1 Maintainer prerequisites

It is important to get familiar with the OSSRH guide and especially the Maven section for performing a release deployment.

To be able to maintain a Java component, a developer must:

- have a GitHub account and have push rights to the GitHub source code repository

- have a Sonatype account and be registered as a maintainer of the *org.openmicroscopy* repository (JIRA issues should be opened for each developer)
- have a valid PGP key for signing the tags and the JARs

8.2.2 Release strategies

There are different strategies to release a Maven component. At the moment we are pushing 2 successive commits (or Pull Requests) to the master branch. The first commit/Pull Request bumps the version number to the release version and is used for generating the release while the second commit bumps the version to the next development cycle.

See also:

https://imagej.net/Development_Lifecycle
A section describing approaches which OME might be considering.

8.2.3 Release preparation

The first step of the Java component release is to prepare a release candidate on the GitHub and Sonatype repositories.

The first operation to perform a Maven release is to bump the version out of SNAPSHOT either via editing the `pom.xml` manually or using the Maven versions plugin:

```
$ mvn versions:set -DnewVersion=x.y.z -DgenerateBackupPoms=false
$ git add -u .
$ git commit -m “Bump release version to x.y.z”
```

Additionally, a PGP-signed tag should be created for the released version e.g. using **`scc tag-release`** or more simply **`git tag -s`**:

```
$ scc tag-release -s x.y.z --prefix v
```

Push the master branch and the tag to your fork for validation by another member of the team:

```
$ git push <fork_name> master
$ git push <fork_name> vx.y.z
```

Once you have updated all the versions and ensured that your build passes without deployment you can perform the deployment by using the release profile with:

```
$ mvn clean deploy -P release
# Potentially add -D gpg.keyname=keyname if desired.
```

This will upload the artifacts to a staging Sonatype repository and perform all the validation steps. The uploaded artifacts can be examined at https://oss.sonatype.org/content/repositories/orgopenmicroscopy-xxxx/ where xxxx is an number incremented for each release.

8.2.4 Release promotion

At the moment all Java components use the Nexus Staging Maven plugin with the *autoReleaseAfterClose* option set to *false*. A separate promotion step is necessary for releasing the component to the Sonatype releases repository. This promotion can happen either via the Sonatype UI using the Release button or using the release phase of the nexus-staging plugin:

```
$ mvn nexus-staging:release -P release
```

See the 'Manually Releasing the Deployment to the Central Repository' section of the Apache Maven guide for more instructions. You should be able to find the staged repository by visiting https://oss.sonatype.org/#stagingRepositories and searching for "org.openmicroscopy".

The rsync to Central Maven and the update of Maven search usually happen within a couple of hours but the components are accessible beforehand.

Once the tag is validated, the master branch and the tag can also be pushed to the organization repository together:

```
$ git push origin vx.y.z
$ git push origin master
```

8.2.5 Next development version

Then finally restore the new development version using e.g. the Maven versions plugin again:

```
# Where w == z+1
$ mvn versions:set -DnewVersion=x.y.w-SNAPSHOT -DgenerateBackupPoms=false
$ git add -u .
$ git commit -m "Bump release version to x.y.w-SNAPSHOT"
$ git push origin master
```

8.2.6 Javadoc

At the moment, we use the service provided https://javadoc.io/ for public hosting of the Javadoc. For each release to Maven Central, the new Javadoc should be automatically deployed within 24h. It is possible to trigger the generation of the Javadoc by visiting the URL.

CHAPTER

NINE

JAVA COMPONENTS (GRADLE)

This document describes the conventions and process used by the OME team for developing, maintaining and releasing its Java components using Gradle as their build system. The set of rules and procedures described below applies to all the submodules of https://github.com/ome/omero-build as well as https://github.com/ome/omero-gradle-plugins/.

See also:

Java components (Maven)
Conventions and process for maintaining the OME Java Components using Maven

9.1 Conventions

9.1.1 Source code and build system

The source code of the components is maintained under version control using Git and hosted on GitHub.

Gradle is the primary build system. The directory layout should follow the standard Maven layout i.e. in the case of a single-module project:

```
 .github/
   workflows/     # GitHub actions workflow
 src/
   main/          # Component source
 test/
   main/
CHANGELOG.md
LICENSE.txt
README.md
build.gradle
settings.gradle
```

Additionally, a `publish.gradle` might exist allowing to declare some publishing tasks.

See also:

Using Git

9.1.2 Development

Server components follow Semantic Versioning i.e. given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes,
- MINOR version when you add functionality in a backwards-compatible manner, and
- PATCH version when you make backwards-compatible bug fixes.

In between releases the version is bumped to the next SNAPSHOT version.

Code contributions should follow the guidelines highlighted in *Code contributions*.

9.1.3 Distribution

All the release artifacts for the repositories listed above should be deployed to the OME Artifactory according to the process described in the next section.

9.2 Release process

To make a new component release, after merging all contributions, on the *master* branch, the version defined in `build.gradle` must be moved out of SNAPSHOT and set to the target version. Release notes described the major changes should also be added to `CHANGELOG.md`.

After committing the changes, a PGP-signed tag must be created for the released version using **`git tag -s`**:

```
$ git tag -s -m "Tag version x.y.z" vx.y.z:
```

The version should then be set to the next SNAPSHOT version in `build.gradle` and the changes committed to the *master* branch.

Both the master branch as well as the tag must be pushed upstream:

```
$ git push origin master vx.y.z
```

This will trigger two GitHub Actions builds and the generated artifacts will be uploaded to the OME Artifactory. All builds from the *master* branch are expected to be snapshots and uploaded to the *ome.snapshots* repository. All tag builds are expected to be full releases and uploaded to the *ome.staging* repository.

Once the artifacts are uploaded, the release artifacts need to be promoted from *ome.staging* to *ome.releases* by logging into https://artifacts.openmicroscopy.org/, going to the Artifacts tab, selecting the *ome.staging* repository and choosing the *Move Content* action.

CHAPTER

TEN

C++ COMPONENTS

This document describes the conventions and process used by the OME team for developing, maintaining and releasing its C++ components.

The set of rules and procedures described below applies to all the following C++ libraries.

Component name	GitHub URL	CMake project name
OME Common C++	https://github.com/ome/ome-common-cpp	*ome-common*
OME Data Model*	https://github.com/ome/ome-model	*ome-model*
OME Files C++	https://github.com/ome/ome-files-cpp	*ome-files-cpp*
OME Qt widgets	https://github.com/ome/ome-qtwidgets	*ome-qtwidgets*
OME CMake Superbuild	https://github.com/ome/ome-cmake-superbuild	*ome-cmake-superbuild*
OME Files Performance	https://github.com/ome/ome-files-performance	*ome-files-performance*
OME Files Python bindings†	https://github.com/ome/ome-files-py	*ome-files-py*

*

Contains both Java and C++ code - see *Java components (Maven)*

†

Contains both Python and C++ code

10.1 Conventions

10.1.1 Source code and build system

The source code of a C++ library should be maintained under version control using Git and hosted on GitHub.

CMake should be used as the primary build system. There is no standard CMake directory layout. C++-only components like ome-common-cpp use a flattened directory layout:

```
cmake/
CMakeLists.txt
docs/                    If applicable
  sphinx/
  doxygen/
lib/
test/
```

Components containing both Java and C++ code like ome-model organize the C++ sources according to the Maven-recommended layout i.e.:

```
<module>/src/main/cpp            Contains the C++ code
<module>/src/main/java           Contains the Java code
```

Additionally, header files should be maintained alongside the source files.

10.1.2 Development

C++ components use Semantic Versioning i.e. given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes,
- MINOR version when you add functionality in a backwards-compatible manner, and
- PATCH version when you make backwards-compatible bug fixes.

Code contributions should follow the guidelines highlighted in *Code contributions*.

10.1.3 Distribution

All the C++ sources and binaries are hosted on the OME downloads according to the process described in the next section.

10.2 Release process

10.2.1 Maintainer prerequisites

To be able to maintain a C++ component, a developer must:

- have a GitHub account and have push rights to the GitHub source code repository
- have a valid PGP key for signing the tags

10.2.2 Source release

The first step of the C++ component release is to prepare a source release from the Git repository.

Prior to a source release, a PR should be opened and merged to:

- review the `release-version` variable in `CMakeLists.txt` and drop the `# unreleased` comment
- update the top-level `NEWS.md` if it exists with the list of changes and the release date

A PGP-signed tag should be created for the released version e.g. using **`scc tag-release`** or more simply **`git tag -s`**:

```
$ scc tag-release -s x.y.z --prefix v
```

Push the master branch and the tag to your fork for validation by another member of the team:

```
$ git push <fork_name> master
$ git push <fork_name> vx.y.z
```

Once the tag is created, run the `<COMPONENT>-release` job under the https://ci.openmicroscopy.org/view/Release view tab. This job will create an archive of the repository using **`git archive`**:

```
$ git archive -v --format=tar "--prefix=${project}-${version}/" -o "${dest}/${project}-$
↪{version}.tar" "${tag}"
$ xz "{dest}/${project}-${version}.tar"
$ git archive -v --format=zip "--prefix=${project}-${version}/" -o "${dest}/${project}-$
↪{version}.zip" "${tag}"
```

and copy the source archives under https://downloads.openmicroscopy.org/<component>/<version>.

10.2.3 Next development version

Once the release is accepted, the version number of *release-version* in `CMakeLists.txt` should be incremented to the next patch number i.e. `x.y.z+1` and a suffixed with an `# unreleased` comment. If a top-level `NEWS.md` file exists, an entry should be added for the next patch release.

See also:

https://github.com/ome/ome-common-cpp/pull/55
Example Pull Request incrementing the patch number of ome-common-cpp and updating `NEWS.md` following the 5.5.0 source release

CHAPTER ELEVEN

OME DEVELOPMENT TOOLS

The following sections summarize the main tools maintained and used internally for the development of OMERO and Bio-Formats. Note this is not an exhaustive list of all tools used in the project.

11.1 Development tools: maintenance

Most of the development tools used internally by the OME project are Python-based and hosted on PyPI. Their source code is on GitHub.

11.1.1 Maintainer prerequisites

To be able to maintain a Python development tool, a developer must:

- have a GitHub account and have push rights to the GitHub source code repository
- have a valid PGP key for signing the tags

11.1.2 Release process

The first operation to perform while releasing a project is to create a signed tag locally:

```
$ cd <project>
$ git tag -s vx.y.z
```

The last command will create a tag with the default *v* prefix, i.e. *vx.y.z*.

Push the signed tag to the source code repository to trigger the upload to PyPI via GitHub action:

```
$ git push origin vx.y.z
```

11.2 Code management: scc

scc is a Python library with a set of utility commands used for code management and used in the OME *Continuous integration*. More information can be found in the Python package page or in the source code page.

If you find a bug or if you want an additional feature to be implemented, please open an issue.

11.2.1 Installation

The scc tools are a set of Python based utility programs. The tools suite can be installed using **`pip`**:

```
$ pip install -U scc
```

This command will install and/or upgrade the **PyGithub** and **yaclifw** package dependencies. If the version of Python installed is older than 2.7, this may also install the **argparse** package.

11.2.2 Github connection

Most of the scc commands instantiate a Github connection using the PyGithub package. GitHub strongly recommends to turn on two-factor authentification (2FA), see About Two-Factor Authentication for more details. If 2FA is activated, the only way to use scc commands creating a GitHub connection is to create an OAuth token, see Creating an access token for command-line use for details on how to create Personal Access Tokens via the GitHub interface. This token can then be stored in the global Git configuration file:

```
git config --global github.token REPLACE_BY_PERSONAL_ACCESS_TOKEN
```

Unless the `--token` option is passed to the scc command, the command first looks for the `github.token` specified in the git config file and, if found, uses this token to connect to GitHub:

```
$ scc merge master --info -v
2013-01-16 22:03:49,633 [  scc.config] DEBUG Found github.token
...
```

If no token is found, the command looks for a `github.user` in the git config file and, if found, uses this username to connect to Github:

```
$ scc merge master --info -v
2013-01-16 22:06:00,256 [  scc.config] DEBUG Found github.user
Enter password for https://github.com/sbesson:
```

Note: The password to be entered here is the GitHub password. Connecting using the GitHub username/password is NOT possible if 2FA has been activated.

Finally, if no token or user is found, both the GitHub username and password are queried at the prompt:

```
$ scc merge master --info -v
# github.token and github.user not found.
# See `scc token` for simpifying use.
Username or token: sbesson
Enter password for https://github.com/sbesson:
```

11.2.3 scc merge

Merge all the PRs based on specified branch matching the input filters including all submodules.

Description

Filters of different types can be specified and combined to include and exclude a set of PRs in the merge command. Each filter needs to be formatted as `key:value`. If no key but only a value is specified, it is assumed the filter is a label filter (see below). These filters can be passed to an *scc merge --include* or *scc merge --exclude* option.

The available filter types are described below:

- Label filters can be specified using the `label` key i.e. `label:<LABEL>`. This filter type will match a Pull Request if one of the following conditions is met:
  1. a label named `<LABEL>` is applied to the Pull Request
  2. the Pull Request description contains a line starting with `--<LABEL>`
  3. one of the Pull Request comments contains a line starting with `--<LABEL>`. Note this comment needs to be written by one of the public members of the organization owning the upstream repository.
- User filters can be specified using the `user` key i.e. `user:<USER>`. This filter type will select a Pull Request if it has been opened by the user `USER`. Additionally, two special user values are allowed:
  1. the `#org` value will match all PRs opened by public members of the organization of the upstream repository
  2. the `#all` value will match all PRs opened by any user
- PR filters can be specified using the `pr` key i.e. `pr:<NUMBER>`. This will select Pull Requests whose ID matches the input number. The form `#number` is also recognized as a PR filter. For repositories containing submodules, it is possible to filter submodule PRs using `user/repo#number`.

Arguments

The first argument is the name of the base branch of origin, e.g.:

```
$ scc merge develop
```

--comment

Add a comment to the PR if there is a conflict while merging the PR

```
$ scc merge develop --comment
```

--default `<filterset>`, **-D** `<filterset>`

Specify the default set of filters to use

Three filter sets are currently implemented: `none`, `org` and `all`. The `none` filter set has no preset filter. The `org` filter set uses `user:#org` and `label:include` as the default include filters and `label:exclude` and `label:breaking` as the default exclude filters. The `all` filter set uses `user:#all` as the default include filters.

Default: `org`

--exclude `<filter>`, **-E** `<filter>`

Exclude PR by filter (see filter semantics above):

```
$ scc merge develop -E label:l1 -E user:u1 -E #45 -E org/repo#40
```

--include <filter>, **-I** <filter>

Include PR by filter (see filter semantics above):

```
$ scc merge develop -I label:l1 -I user:u1 -I #45 -I org/repo#40
```

--check-commit-status <status>, **-S** <status>

Exclude PR based on the status of the last commit

Three options are currently implemented: `none`, `no-error` and `success-only`. By default (`none`), the status of the last commit on the PR is not taken into account. To include PRs which have a successful status only, e.g. PRs where the Travis build is green, use the `success-only` option:

```
$ scc merge develop -S success-only
```

To exclude all PRs with an `error` or `failure` status, use the `no-error` option:

```
$ scc merge develop -S no-error
```

--info

Display the candidate PRs to merge but do not merge them

```
$ scc merge develop --info
```

--push <branchname>

Push the locally merged branch to Github

```
$ scc merge develop --push my-merged-branch
```

--reset

Recursively reset each repository to the HEAD of the base branch

```
$ scc merge develop --reset
```

--shallow

Merge the PRs for the top-level directory only, excluding submodules:

```
$ scc merge develop --shallow
```

--remote <remote>

Specify the name of the remote to use as the origin. Default: origin:

```
$ scc merge develop --remote gh
```

As a concrete example, the first step of a merge job is calling the following merge command:

```
$ scc merge master --no-ask --reset --comment --push merge_ci
```

Use cases

The basic command will use the default filters and merge all PRs opened against `master` by any public members of the organization, include any PR labeled as `include` and exclude any PR labeled as `breaking` or `exclude`:

```
$ scc merge master
```

The following command overrides the default set of filters and will only merge PRs opened against `master` labeled as `my_label`:

```
$ scc merge master -Dnone -Ilabel:my_label
```

The following command overrides the default set of filters and will merge all PRs opened against `master` by public members of the organization, include any PR labeled with `my_label` and exclude any PR labeled as `exclude`:

```
$ scc merge master -Dnone -Iuser#org -Ilabel:my_label -Elabel:exclude
```

Changed in version 0.3.0: Added default values for *`--include`* and *`--exclude`* options.

Changed in version 0.3.8: Added *`--shallow`* and *`--remote`* options.

Changed in version 0.4.0: Added *`--check-commit-status`* option.

11.2.4 scc travis-merge

Merge PRs in a Travis environment, using the PR comments to generate the merge filters.

```
$ scc travis-merge
```

This command internally defines all the filter options exposed in **`scc merge`**.

The target branch is read from the base of the PR, the *`scc merge --default`* option is set to `none` meaning no PR is merged by default and no default *`scc merge --exclude`* option is defined.

The *`scc merge --include`* filter is determined by parsing all the PR comments lines starting with `--depends-on`.

To include a PR from the same GitHub repository, use the PR number prepended by `#`. For instance, to include PR 67 in the Travis build, add a comment line starting with `--depends-on #67` to the PR.

To include a PR from a submodule, use the PR number prepended by `submodule_user/submodule_name#`. For instance, to include PR 60 of bioformats in the Travis build, add a comment line starting with `--depends-on openmicroscopy/bioformats#60` to the openmicroscopy PR.

Note: The **`scc travis-merge`** command works solely for Pull Requests' Travis builds.

11.2.5 scc update-submodules

Update the pointer of all submodules based on specified branch.

The first argument is the name of the base branch of origin, e.g.:

```
$ scc update-submodules develop
```

--push <branchname>

Push the locally merged branch to Github and open a PR against the base branch:

```
$ scc merge develop --push submodules_branch
```

--no-pr

Combined with *--push* option, push the locally merged branch to Github but skip PR opening:

```
$ scc merge develop --push submodules_branch --no-pr
```

--remote <remote>

Specify the name of the remote to use as the origin (default: origin):

```
$ scc update-submodules develop --remote gh
```

11.2.6 scc rebase

Rebase a PR (open or closed) onto another branch and open a new PR.

The first argument is the number of the PR to rebase and the second argument is the name of the branch onto which the PR should be rebased:

```
$ scc rebase 142 develop
```

Assuming the head branch used to open the PR 142 was called `branch_142`, this command will rebase the tip of `branch_142` onto origin/develop, create a new local branch called `rebased/develop/branch_142`, push this branch to Github and open a new PR. Assuming the command opens PR 150, to facilitate the integration with **scc check-prs**, a `--rebased-to #150` comment is added to PR 142 and a `--rebased-from #142` comment is added to the PR 150. Finally, the command will switch back to the original branch prior to rebasing and delete the local `rebased/develop/branch_142`.

Note: By default, **scc rebase** uses the branches of the `origin` remote to rebase the PR. To specify another remote, use the *--remote* option.

--no-pr

Skip the opening of the PR

```
$ scc rebase 142 develop --no-pr
```

--no-delete

Do not delete the local rebased branch

```
$ scc rebase 142 develop --no-delete
```

--remote <remote>

Specify the name of the remote to use for the rebase (default: origin)

```
$ scc rebase 142 develop --remote snoopycrimecop
```

--continue

Re-run the command after manually fixing conflicts

If **scc rebase** fails due to conflict during the rebase, you will end up in a detached HEAD state.

If you want to continue the rebase operation, you will need to manually fix the conflicts:

```
# fix files locally
$ git add conflicting_files # add conflicting files
$ git rebase --continue
```

This conflict solving operation may need to be repeated multiple times until the branch is fully rebased.

Once all the conflicts are resolved, call **scc rebase** with the *--continue* option:

```
$ scc rebase --continue 142 develop
```

Depending on the input options, this command will perform all the steps of the rebase command (Github pushing, PR opening) skipping the rebase part.

Alternatively, you can abort the rebase and switch to your previous branch:

```
$ git rebase --abort
$ git checkout old_branch
```

Changed in version 0.3.10: Automatically add `--rebased-to` and `--rebased-from` comments to the source and target PRs.

11.2.7 scc check-prs

Compare two development branches and check that PRs merged in one branch have been merged to the other.

The basic workflow of the **scc check-prs** command is the following:

- list all first-parent merge commits for each branch including git notes referenced as `see_also/other_branch` where other_branch is the name of the branch to check against.
- exclude all merge commits with a note containing either "See gh-" or "n/a"
- for each remaining merge commit, parse the PR number and look into the PR body/comments for lines starting with `--rebased-to`, `--rebased-from` or `--no-rebase`.

Additionally, for each line of each PR starting with `--rebased-to` or `--rebased-from`, the existence of a matching line is checked in the corresponding source/target PR. For instance, if PR 70 has a `--rebased-from #67` line and a `--rebased-from #66` line, then both PRs 66 and 67 should have a `--rebased-to #70` line.

This command requires two positional arguments corresponding to the name of the branch of origin to compare:

```
$ scc check-prs dev_4_4 develop
```

--shallow

Check PRs in the top-level directory only, excluding submodules:

```
$ scc check-prs dev_4_4 develop --shallow
```

--remote <remote>

Specify the name of the remote to use as the origin (default: origin):

```
$ scc check-prs dev_4_4 develop --remote gh
```

--no-check

Do not check links between rebased comments:

```
$ scc check-prs dev_4_4 develop --no-check
```

New in version 0.3.10: Added support for body/comment parsing and `--rebased-to/from` linkcheck

Changed in version 0.4.0: Improved command output and added support for submodule processing

Changed in version 0.5.0: Renamed command

11.2.8 scc version

Return the version of the scc tools:

```
$ scc version
0.3.0
```

New in version 0.2.0.

11.2.9 scc deploy

Deploy a website update using file symlink replacement:

```
$ scc deploy folder
```

The goal of this command is to enable overwriting of deployed doc content and allow for "hot-swapping" content served by Apache without downtime and HTTP 404s.

--init

Prepare folder for symlink replacement. Should only be run once

```
$ scc deploy folder --init
```

New in version 0.3.1.

The hudson jobs ending with `release-docs` and OMERO-docs-internal deploy the documentation artifacts to necromancer. The target directory (sphinx-docs) is controlled by the hudson:hudson user, so all file system operations are allowed. Each job has the target directory configured in the SSH publisher target directory property. After deployment has happened to a temporary directory, a series of symlink moves happens making sure that the symlink points to the updated content.

11.2.10 scc check-status

Check the status of the Github API:

```
$ scc check-status && echo "Passing"
Passing
```

-n <N>

Display N last status messages from Github API history:

```
$ scc check-status -n 4
2013-11-04 13:40:48 (minor) We're investigating an increase in error responses from␣
↪the API.
2013-11-04 14:33:55 (good) Everything operating normally.
2013-11-05 12:59:50 (minor) We're investigating reports of an increase in 502s from␣
↪the GitHub API.
2013-11-05 13:07:15 (good) Everything operating normally.
```

New in version 0.4.0.

11.3 OME administration: omego

omego is a Python library with a set of utility commands used for managing the installation and administration of OME applications like OMERO. More information can be found in the Python package page or in the source code page.

If you find a bug or if you want an additional feature to be implemented, please open an issue.

11.3.1 Installation

The omego tools are a set of Python based utility programs. The tools suite can be installed using `pip`:

```
$ pip install -U omego
```

This command will install and/or upgrade the **yaclifw** package dependency. If the version of Python installed is older than 2.7, this may also install the **argparse** package.

11.4 OME development platform: devspace

Devspace is a Continuous Integration tool managed by Jenkins CI providing an automation framework that runs repeated jobs. The default deployment initializes a Jenkins CI master with a predefined set of jobs. More information can be found in the source code page.

If you find a bug or if you want an additional feature to be implemented, please open an issue.

Running and maintaining Devspace requires:

- Docker engine https://docs.docker.com/.

Optionally a brief understanding of Ansible, Ansible inventory, and Ansible playbooks.

11.4.1 Installation

Manual

Install following prerequisites:

- Docker engine https://docs.docker.com/engine/installation/
- Docker compose

  ```
  $ pip install docker-compose
  ```

Checkout git repository and run

```
$ docker-compose -f docker-compose.yml up --build
```

OpenStack

This is an example of how to provision and deploy Devspace using ansible on openstack. Check out management tools and run:

```
$ source tenancy.rc
$ cd infrastructure/ansible
$ ansible-playbook os-devspace.yml -e vm_name=devspace-test -e vm_key_name=your_key
$ ansible-playbook -l devspace-test -u centos devspace.yml
```

To deploy devspace from custom branch, first set up vars:

```
omero_branch: develop
snoopy_dir_path: "/path/to/snoopy"

git_repo: "https://github.com/user_name/devspace.git"
version: "your_branch"
```

CHAPTER TWELVE

OME DEPLOYMENT TOOLS

This section describes deployment tools supported by the OME team. It is primarily designed for the core OME developers who want to bring new or upgrade existing prerequisites. The following steps explain the connections between basic repositories and the testing workflow.

Note: This section requires a brief understanding of Ansible and Docker engine https://docs.docker.com/.

12.1 Prerequisites locations

The list of OME prerequisites is stored in multiple git repositories, each of which is available from several locations.

12.1.1 OME Infrastructure

Infrastructure is provided to simplify deployment using Ansible.

The Infrastructure repository is available from:

- https://github.com/ome/infrastructure

Roles repositories are available in:

- https://github.com/ome?q=ansible-role

12.1.2 OMERO-install

OMERO installation scripts are provided to help new users with installing OMERO.server for the first time on a clean system, and can be used as the basis for more advanced configurations.

The OMERO-install repository is available from:

- https://github.com/ome/omero-install

The OMEROWEB-install repository is available from:

- https://github.com/ome/omeroweb-install

12.2 Testing workflow

The testing environment is split into two tiers: developer and production testing platform.

12.2.1 Devspace - Continuous Integration (Dev Testing)

Continuous integration tools managed by Jenkins CI providing an automation framework that runs repeated jobs. The default deployment initializes a Jenkins CI master with a predefined set of jobs.

The *Devspace* repository is available from:

- https://github.com/ome/devspace

Devspace Dockerfiles uses common devslave image. The Devslave repository is available from:

- https://github.com/ome/devslave-c7-docker

12.2.2 CI-master - Continuous Delivery (Production)

Production Continuous Delivery (CD) platform managed by Jenkins. More details about CI-master available on *Continuous integration*

12.3 How to add new/upgrade/remove old prerequisites

When the OME platform requires a new set of prerequisites all the above listed repositories may require updates. Depends on the nature of packages developers must consider:

- infrastructure repository:
  - adding new Ansible role in its own repo or update existing one https://github.com/ome?q=ansible-role
  - adding complete Ansible playbook to Infrastructure
- adding new scripts installing appropriate package and its dependencies to OMERO-install or OMEROWEB-install, that includes:
  - updating Linux and Mac installation scripts
  - updating documentation autogen
- deploying Devspace to test OMERO-install scripts, that includes:
  - adding new Docker container if requires to support additional processes
  - adjusting predefined Jenkins jobs

After successful testing new prerequisites can be proposed as a permanent adjustment to production CD.

Note: Any Python module that is distributed from Linux distro packages must be installed from RPM file. Python modules only available on PyPI should be added as PIP requirement.

12.4 Pre release testing

It is also very important to test all the dependencies before release to make sure sysadmin instructions are fully tested. The easiest way to test is to use Devspace.

12.5 EXAMPLE

This example shows how to test and upgrade OMERO dependencies.

1. Open a PR against omero-install (e.g. install Pillow from RPM https://github.com/ome/omero-install/pull/129).
2. Upgrade base docker image using (e.g. https://github.com/ome/devslave-c7-docker/blob/master/Dockerfile#L19) and open a PR against devslave-c7-docker.

   Run https://ci.openmicroscopy.org/job/DOCKER-merge. Latest merge image will be released to Docker Hub. For more details about configuring automated builds on Docker Hub, see https://docs.docker.com/docker-hub/builds/.
3. Update devspace to use newly released container (e.g. https://github.com/ome/devspace/pull/63/files#diff-296e14ae0dc392c7edd9369908467953).
4. Commit and push your changes to github (e.g. https://github.com/ome/devspace/pull/63), set your branch in devspace ansible config and deploy.

If all tests are passing, above repositories should be tagged and tag should be propagated accordingly.

Note: We are working very hard to improve and simplify that process.

CHAPTER

THIRTEEN

CONTINUOUS INTEGRATION

The OME project uses Jenkins as a continuous integration server. Bring up a web browser to access the OME Jenkins server.

The following sections summarize the main continuous integration jobs used for the development of OMERO, Bio-Formats and the OME documentation sets. Note this is not an exhaustive list of all jobs in the project. To know more about a particular job, click on the *Configure* button on the left-side panel of the job window. This panel should also include a *GitHub* button linking to the code repository the job is building from (alternatively, the console output for the build will indicate where the changes are being fetched from).

13.1 Continuous integration branches and jobs

13.1.1 Versioning

OME uses semantic versioning as defined in https://semver.org. Each version number is identified as MAJOR.MINOR.PATCH where MAJOR is the major version number, MINOR the minor version number and PATCH the patch version number. Additional pre-release labels are added as extensions of this version number, e.g. 4.4.0-rc1 or 5.0.0-beta1.

Major release
An increment of the MAJOR version or the MINOR version is typically considered as a major release in OME, e.g. 5.0.0 or 5.1.0.

Point release (patch release)
An increment of the PATCH version is called a point (or patch) release in OME, e.g. 4.4.9.

13.1.2 Development branches

Most of the OME code is split between four repositories: openmicroscopy.git, bioformats.git, scripts.git, ome-documentation.git. Each repository contains several development branches associated with development series:

- The “dev_5_y” branch(es) containing work on the current 5.y.x series.
- The “develop” branch containing work on the next major release series.

Note that only two branches are usually maintained simultaneously. With this workflow, it is possible to have a point release immediately, while still working on more major releases by ensuring that (nearly) all commits that are applied to dev_5_y are applied to develop in order to prevent regressions.

13.1.3 Labels

Labels are applied to PRs on GitHub under the "Issues" tab of each repository.

Each release series consists of PRs labeled according to the release version, which also matches the name of the branch they will be merged into e.g. 5.1.x series PRs will be labeled as "dev_5_1" and be merged into the dev_5_1 branch.

Multiple labels are used in the PR reviewing process:

- the "include" label allows you to include a PR opened by a non-member of the OME organization in the merge builds for review.
- the "exclude" label allows you to exclude a PR opened by any user from the merge builds.
- the "on hold" label allows you to signal that a PR should not be reviewed or merged, even though it is not excluded.

13.1.4 Job names

All core OME job names take the form `COMPONENT-VERSION-TYPE-DESCRIPTION`, where:

- `COMPONENT` refers to the core OME component, e.g. *OMERO* for OMERO or *BIOFORMATS* for Bio-Formats.
- `VERSION` is the MAJOR.MINOR version, e.g. *5.0* or *5.1*.
- `TYPE` represents the source of the job and can take the following values:
  - *latest*: build from the tip of the development branch, e.g. *origin/dev_5_0*;
  - *merge*: build from the tip of the development branch with all PRs merged using *scc merge* with the *org* default filter set;
  - *release*: build from and optionally create a tag at the tip of a development branch, e.g. *v5.0.1-rc4*.
- `DESCRIPTION` describes the job via a set of dash-separated keywords, e.g. *docs-autogen*.

13.2 OMERO jobs

13.2.1 Deployments

The table below lists all the hostnames, ports and URLs of the OMERO.web clients of the deployment jobs described above:

Series	OMERO.server deployment job	Hostname	Port	OMERO.web deployment job	Webclient
Merge	*OMERO-server*	merge-ci.openmicroscopy.org	4064	*OMERO-web*	https://merge-ci.openmicroscopy.org/web/

13.2.2 Jobs

Job task	Merge jobs
Merges the PRs and couple versions	*OMERO-gradle-plugins-push* *OMERO-build-push* *OMERO-push* *OMERO-insight-push* *OMERO-matlab-push*
Builds the OMERO artifacts	*OMERO-gradle-plugins-build* *OMERO-build-build* *OMERO-build* *OMERO-insight-build* *OMERO-matlab-build*
Deploy OMERO	*OMERO-server* *OMERO-web*
Runs the OMERO integration tests	*OMERO-test-integration*

OMERO-gradle-plugins-push
OMERO-build-push
OMERO-push
OMERO-insight-push
OMERO-matlab-push

These jobs merge all the PRs opened against the development branches and couple the component versions for the following repositories:

- https://github.com/ome/omero-gradle-plugins
- https://github.com/ome/omero-build
- https://github.com/ome/openmicroscopy
- https://github.com/ome/omero-insight
- https://github.com/ome/omero-matlab

OMERO-gradle-plugins-build
OMERO-build-build
OMERO-build
OMERO-insight-build
OMERO-matlab-build

These jobs build the OMERO server components, the OMERO bundles and the OMERO clients from the integration branches created by the push jobs.

OMERO-server

This job deploys the server (see *Deployments*) created by *OMERO-build*.

OMERO-web

This job deploys the Web application (see *Deployments*) created by *OMERO-build*.

OMERO-test-integration
This job deploys an OMERO.server and runs the OMERO.java, OMERO.py and OMERO.web integration tests.

13.3 Bio-Formats jobs

Job task	Merge jobs
Merge the PRs and couple versions	*BIOFORMATS-push*
Build the Bio-Formats artifacts	*BIOFORMATS-build* *BIOFORMATS-image*
Build the Bio-Formats documentation	*BIOFORMATS-linkcheck*
Run the Bio-Formats non-regression tests	*BIOFORMATS-test-repo*

BIOFORMATS-push
This job merges all the PRs opened against the development branch of https://github.com/ome/bio-formats-build and couples the component versions

BIOFORMATS-build
BIOFORMATS-image
This job builds all the Bio-Formats artifacts using Maven and Ant

BIOFORMATS-linkcheck
This job runs the linkchecker on the Bio-Formats documentation

BIOFORMATS-test-repo
This job consumes the Docker image built by *BIOFORMATS-image* and runs the non-regression automated tests against the curated QA repository

13.4 Documentation jobs

All documentation jobs are listed under the Docs view tab of Jenkins. A *GitHub* button in the left-side panel of the job window links to the code repository the job is building from (alternatively, the console output for the build will indicate where the changes are being fetched from).

More detail on how and where to edit OME documentation is available on the *Editing the OME documentation* page.

Job task	OMERO 5.x series
Builds the OMERO documentation for review	*OMERO-docs*

The Bio-Formats documentation jobs are described in the *Bio-Formats jobs* section.

The OME Model set is independent of the current OMERO/Bio-Formats version.

Job task	
Review PRs opened against the OME Website	*WEBSITE-push*
Review PRs opened against the Presentations website	*PRESENTATIONS-merge*

13.4.1 Configuration

For all jobs building documentation using Sphinx, the following environment variables are used:

- the Sphinx building options, `SPHINXOPTS`, is set to `-Dsphinx.opts="-W"`
- the release number of the documentation is set by `OMERO_RELEASE`, `BF_RELEASE` or by the relevant POM
- the source code links use `SOURCE_USER` and `SOURCE_BRANCH`
- for the Bio-Formats and OMERO sets of documentation, the name of the Jenkins job is set by `JENKINS_JOB`.

Note that the https://github.com/ome/sphinx_theme repository is no longer used, this hosted the theme to match the old plone website.

13.4.2 OMERO 5.x series

The branch for the 5.x series of the OMERO documentation is develop.

OMERO-docs
This job is used to review the PRs opened against the develop branch of the OMERO 5.x documentation

1. Merges PRs using *scc merge*
2. Pushes the branch to https://github.com/snoopycrimecop/ome-documentation/tree/merge_ci
3. Runs `make clean html` to build the HTML Sphinx documentation
4. Runs `make linkcheck`

13.4.3 Jekyll websites

The following set of jobs is used to review or publish the content of the *OME Jekyll websites*.

WEBSITE-push
This job is used to review the PRs opened against the master branch of https://github.com/ome/www.openmicroscopy.org

1. Merges PRs using *scc merge* and pushes the branch to https://github.com/snoopycrimecop/www.openmicroscopy.org/tree/merge_ci
2. The GitHub Pages service deploys the staging website content under https://snoopycrimecop.github.io/www.openmicroscopy.org/

PRESENTATIONS-merge
This job is used to review the PRs opened against the master branch of https://github.com/ome/presentations

1. Merges PRs using *scc merge* and pushes the branch to https://github.com/snoopycrimecop/presentations
2. The GitHub Pages service deploys the staging website content under https://snoopycrimecop.github.io/presentations/

13.5 Release jobs

The following table lists the main Jenkins jobs used during the release process. All release jobs should be listed under the https://ci.openmicroscopy.org/view/Release view.

Job task	OMERO
Trigger the OMERO release jobs	https://ci.openmicroscopy.org/job/OMERO-DEV-release-trigger
Tags the OMERO source code repository	https://ci.openmicroscopy.org/job/OMERO-DEV-release-push
Build the OMERO download artifacts	https://ci.openmicroscopy.org/job/OMERO-DEV-release
Generate the OMERO downloads page	https://ci.openmicroscopy.org/job/OMERO-DEV-release-downloads
Deploy the documentation for the decoupled repositories	https://ci.openmicroscopy.org/job/OMERO-DEV-release-artifacts

Job task	Bio-Formats
Build the Bio-Formats download artifacts	https://ci.openmicroscopy.org/job/BIOFORMATS-DEV-release

13.5.1 Bio-Formats

https://ci.openmicroscopy.org/job/BIOFORMATS-DEV-release

This job builds the Java downloads artifacts of Bio-Formats

1. Checks out the `RELEASE` tag of https://github.com/ome/bioformats
2. Builds Bio-Formats using `clean release tools utils docs docs-sphinx dist-bftools dist-matlab dist-octave test`
3. Downloads the documentation artifacts from OME artifactory
4. Copies the build artifacts to a LDAP-protected folder under downloads.openmicroscopy.org

13.5.2 OMERO

OMERO-DEV-release-trigger

This job triggers the OMERO release jobs. Prior to running it, its variables need to be properly configured:

- `RELEASE` is the OMERO release number.
- `ANNOUNCEMENT_URL` is the URL of the forum release announcement and should be set to the value of the URL of the private post until it becomes public.
- `MILESTONE` is the name of the Trac milestone which the download pages should be linked to.

1. Triggers *OMERO-DEV-release-push*
2. Triggers *OMERO-DEV-release*

See the build graph

OMERO-DEV-release-push

This job creates a tag on the *develop* branch

1. Runs *scc tag-release $RELEASE* and pushes the tag to the snoopycrimecop fork of openmicroscopy.git

OMERO-DEV-release

This matrix job builds the OMERO components with Ice 3.6

1. Checks out the `RELEASE` tag of the snoopycrimecop fork of openmicroscopy.git
2. Builds the OMERO.server and the clients using :omero_source: *OMERO.sh <docs/hudson/OMERO.sh>*
3. Executes the *release-hudson* target for the *ome.staging* Maven repository
4. Copies the build artifacts to a LDAP-protected folder under downloads.openmicroscopy.org
5. Triggers *OMERO-DEV-release-downloads*

OMERO-DEV-release-downloads

This job builds the OMERO downloads page

1. Checks out the *develop* branch of https://github.com/ome/ome-release.git
2. Runs *make clean omero*

OMERO-DEV-release-artifacts

This job deploys the Javadoc and the slice2html documentation

1. Loops through omero-{model,common,romio,renderer,server,blitz,gateway-java}
2. Checks the latest version available on https://artifacts.openmicroscopy.org
3. Deploys the documentation in the respective directory

Documentation release jobs are documented on *Documentation jobs*.

CHAPTER

FOURTEEN

EDITING THE OME DOCUMENTATION

This guide assumes you are already familiar with *Using Git* and GitHub and only covers where to find the sources, builds etc. you will need to edit and review any given content. Further information on the CI builds is available on the *Documentation jobs* page.

Some of the live web documentation also features 'Show on GitHub' and 'Edit on GitHub' links in the lefthand menu which will take you directly to the source file and allow you to edit within the GH interface and then open a PR (you should never use this functionality to edit autogenerated pages, see below for details of these).

14.1 Overview

This page covers technical documentation which is written in `.rst` files and generated into html using Sphinx. There are sets for each aspect of the project, plus this set of 'contributing developer' documentation aimed at giving an overview of the OME process and workflows across products that might be of interest to external people, and the OME internal docs for internal-only private workflows and processes.

Some of the content for these is either autogenerated or copied from external sources as described below. Formatting and style guidance can be found in the README for the ome-documentation repo, along with instructions for getting set up with Sphinx.

The jekyll websites hosted by OME, which include the Help workflow guides, are covered on *Jekyll-hosted websites*.

14.2 What goes where?

The decision trees below try to give an insight into the thought process behind choosing what information to host where.

14.3 Bio-Formats documentation

Hosted at https://docs.openmicroscopy.org/bio-formats/{{version}} (plus latest redirect - docs.openmicroscopy.org/latest/bio-formats/)

This documentation covers all aspects of Bio-Formats - using it with other tools, specific guidance for Bio-Formats developers, and supported formats. Related topics - OME file formats and the data model are covered in the OME Data Model and File Formats documentation.

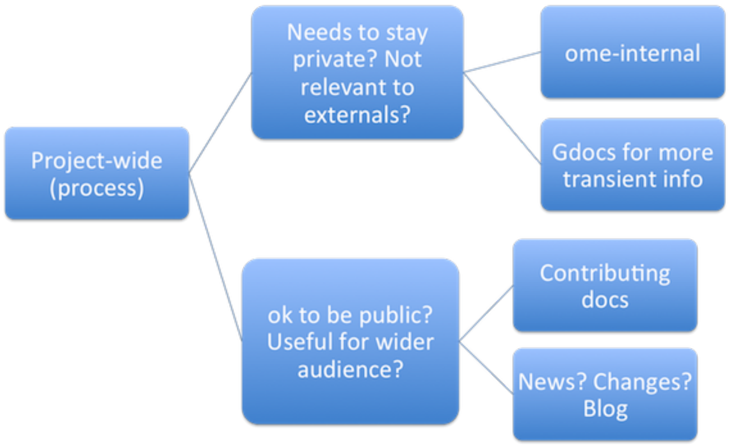

Project-wide (process)
Needs to stay private? Not relevant to externals?
ome-internal
Gdocs for more transient info
ok to be public? Useful for wider audience?
Contributing docs
News? Changes? Blog

Fig. 1: Project-wide information

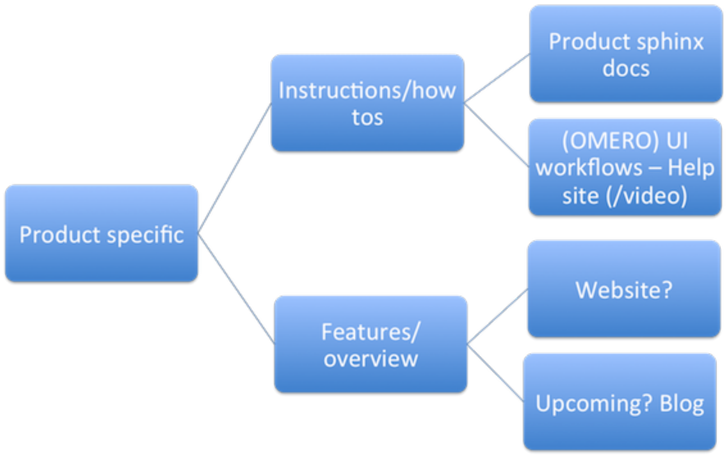


Fig. 2: Product-specific information

14.3.1 Builds

See *Documentation jobs*.

14.3.2 Source

The Sphinx documentation is decoupled from the code repository, at https://github.com/ome/bio-formats-documentation.

14.3.3 Building locally

The build uses Sphinx via `Maven`. `mvn` will generate the webpages provided you have both Sphinx and Maven installed. To avoid running the linkchecker by default use `mvn -DskipSphinxTests`.

14.3.4 Building/reviewing PRs via the CI

Once a PR is open, you can build it for review using the https://merge-ci.openmicroscopy.org/jenkins/job/BIOFORMATS-linkcheck job on the Jenkins CI. Staging documentation is no longer deployed at a URL but you can download it as a zip for review with the correct styling by going to the workspace folder in the job.

14.3.5 Autogenerated content

The following parts of the documentation are autogenerated during the build:

- Supported Formats table and format pages - generated from format-pages.txt
- Dataset structure table - generated from the readers
- Metadata support table - generated from the readers

Generally, unless you are adding new file support, the format pages are the only ones you are likely to be editing. There is documentation on Adding format/reader documentation pages in the Bio-Formats developer section.

14.3.6 Publishing

The live webpages are updated as part of the release process.

14.4 OME Contributing Developer documentation

Hosted at https://docs.openmicroscopy.org/contributing/ (always latest).

This covers the OME team processes and workflows that may be of interest to external contributors or other open source teams - information about what tools we use and how, rather than internal-only workflows (like standup prep) or anything which needs to be kept private (these belong in the internal docs instead).

14.4.1 Builds

See *Documentation jobs*.

14.4.2 Source

The source files are at https://github.com/ome/ome-contributing.

14.4.3 Building locally

The build uses Sphinx. You can build locally using `make clean html` provided you have Sphinx. There is further information on getting these set up in the README.

14.4.4 Building/reviewing PRs

Once a PR is open, a build on Read the docs will be triggered. Staging documentation will be available at a given URL linked to the PR. See https://docs.readthedocs.io/en/stable/pull-requests.html for more details.

14.4.5 Publishing

The live webpages are updated when a PR is merged.

14.5 OME Data Model and File Formats documentation

Hosted at https://docs.openmicroscopy.org/ome-model/{{version}}/ (plus latest redirect - https://docs.openmicroscopy.org/latest/ome-model/).

This covers the OME-TIFF format and the OME data model.

14.5.1 Builds

See *Documentation jobs*. Note that this documentation is built and hosted individually and as part of the OME Files documentation bundle.

14.5.2 Source

The documentation is in the `/docs/sphinx/` folder in the code repository at https://github.com/ome/ome-model.

14.5.3 Building locally

The build uses Sphinx via Maven. You can build locally using `make clean html` provided you have both installed.

14.5.4 Building/reviewing PRs via the CI

Once a PR is open, you can build it for review using the https://merge-ci.openmicroscopy.org/jenkins/job/BIOFORMATS-build job on the Jenkins CI. Staging documentation is no longer deployed at a URL but you can download it as a zip for review with the correct styling from the job page (see 'Last Successful Artifacts' at the top of the centre panel.

14.5.5 Publishing

The live webpages are updated as part of the release process.

14.6 OME Internal documentation (private)

For members of the OME team, this set of documentation is available at https://docs.openmicroscopy.org/internal/ behind an ldap log-in.

14.6.1 Builds

https://ci.openmicroscopy.org/job/OME-internal-merge-docs.

14.6.2 Source

https://github.com/openmicroscopy/ome-internal (private repository)

14.6.3 Building locally

The build uses Sphinx via `ant`. You can build locally using `make clean html` provided you have both installed.

14.6.4 Building/reviewing PRs via the CI

Once a PR is open, you can build it using https://ci.openmicroscopy.org/job/OME-internal-merge-docs and then view the rendered text on the live webpages.

14.6.5 Publishing

Content is automatically published to the private URL each day or when the merge build is run.

14.7 OMERO documentation

Hosted at https://docs.openmicroscopy.org/omero/{{version}}/ (plus latest redirect - https://docs.openmicroscopy.org/latest/omero/).

This documentation includes developer and sysadmin documentation for OMERO, version history, client overviews and CLI usage documentation. Workflow-based user documentation belongs in the Help instead while features and other overview material aimed at scientists and other non-IT people may belong on the website (see *Jekyll-hosted websites*).

14.7.1 Builds

See *Documentation jobs*.

14.7.2 Source

All the source files are in the `/omero/` folder at https://github.com/ome/ome-documentation.

14.7.3 Building locally

The build uses Sphinx via `ant`. You can build locally using `make clean html` provided you have both installed. There is further information on getting these set up and on build targets in the README.

14.7.4 Building/reviewing PRs via the CI

Once a PR is open, you can build it for review using the https://merge-ci.openmicroscopy.org/jenkins/job/OMERO-docs job on the Jenkins CI. Staging documentation are no longer deployed at a URL but you can download it as a zip for review with the correct styling from the top centre panel in the job, under 'Last Successful Artifacts'.

14.7.5 Autogenerated/inserted external content

The OMERO documentation is the most complicated set, being the only repo where material is sourced from other repositories. Source repositories are:

- https://github.com/ome/openmicroscopy/ (OMERO code repo)
- https://github.com/ome/omero-install (OMERO server with Web installation)
- https://github.com/ome/omeroweb-install (OMERO.web separately from OMERO.server installation)

Version history

Content for OMERO version history should first be submitted as a PR against https://github.com/ome/openmicroscopy/blob/develop/history.rst in the OMERO code repository. Best practice is to paste the content into the documentation page to test build it before opening the PR. Once the PR is merged, an autogenerated PR can be opened against the documentation repo to transfer the content.

CLI output

The output of the following CLI commands will be used as configuration files in the documentation:

- `omero config parse`
- `omero ldap setdn -h`
- `omero db script`
- `omero web config nginx`
- `omero web config nginx-location`

See autogen_docs to check the name of the output files. Changes to the output should be submitted as a PR against the OMERO code repository.

Installation walkthroughs

Installation walkthroughs for OMERO.server and OMERO.web are generated in separate repositories. When the installation instructions are modified e.g. a new dependency is added, a PR must be opened against one of the following repositories:

- https://github.com/ome/omero-install for server installation with OMERO.web
- https://github.com/ome/omeroweb-install for OMERO.web installation not with an OMERO.server

OMERO.server installation with OMERO.web:

- The walkthroughs are generated using a bash script
- Code snippets will be included in the documentation pages using *literalinclude* e.g. server-ubuntu-ice36.rst
- The changes made against https://github.com/ome/omero-install will only be included in the documentation once they are merged and the autogen job has been run. When making changes that need to be visible in the documentation during review, you will need to:
  - Generate the walkthrough(s)
  - Open a doc PR
  - Copy the generated walkthrough(s) under omero/sysadmins/unix/walkthrough
  - Adjust if required the start/end of the *literalinclude*

OMERO.web installation separately from OMERO.server:

- The walkthroughs are generated using ansible. The omeroweb-install README file contains instructions on how to generate the walkthroughs
- The generated walkthroughs are .rst files that are used as pages in the documentation. This workflow does not use *literalinclude*.
- The changes made against https://github.com/ome/omeroweb-install will only be included in the documentation once they are merged and the autogen job has been run. When making changes that need to be visible in the documentation during review, you will need to:

  - Generate the walkthrough(s)
  - Open a doc PR
  - Copy the generated walkthrough(s) under omero/sysadmins/unix/install-web/walkthrough

Model glossary

Content for Glossary of all OMERO Model Objects is generated using GraphPathReport.

To update the content:

- Run the command indicated in GraphPathReport to generate `EveryObject.rst`
- Replace EveryObject.rst with the generated one
- Open a PR with any changes

Training examples

The contents of the following examples files is not automatically updated:

- omero/developers/Java.rst
- omero/developers/Matlab.rst
- omero/developers/Python.rst

When the examples under https://github.com/ome/openmicroscopy/tree/develop/examples/Training are modified, you will need to **manually** make the changes in the above files and open a doc PR.

CHAPTER

FIFTEEN

JEKYLL-HOSTED WEBSITES

The main OME website is produced using Jekyll.

15.1 Installing Jekyll

Jekyll can be installed as a system application (requires administrator privileges) or for a single user.

1. Install a recent version of Ruby. Recent versions of OS X and Linux may already include a suitable version, however you will require administrator privileges to install Jekyll. Alternatively on OS X you can use Homebrew, and on Linux either rbenv or RVM:

   ```
   brew install ruby
   ```

2. Install Jekyll:

   ```
   gem install jekyll
   ```

15.2 Running Jekyll

In a terminal enter the local git checkout and run:

```
jekyll serve -w -b ''
```

This will build the web pages and start a local server on http://localhost:4000/. Jekyll should automatically rebuild the pages when changes are made to the source files, and will override any *baseurl* defined in *_config.yml*.

15.3 Editing the website

The source for the OME website is on GitHub at https://github.com/ome/www.openmicroscopy.org. The website uses the 'master' branch, PRs should be opened directly against it. Most of the files are written in html but the announcements and blog posts are in markdown and further content may migrate to markdown in future.

Once a PR is open and has passed Travis, it will be integrated in the staging integration branch for the OME Website site via the daily https://merge-ci.openmicroscopy.org/jenkins/job/WEBSITE-push job. The staging Jekyll website will be deployed by the GitHub Pages service at https://snoopycrimecop.github.io/www.openmicroscopy.org for review.

Once the PR is merged, the HEAD of master will be deployed by the GitHub pages service at https://ome.github.io/www.openmicroscopy.org.

Updating the live website requires two steps, first creating an archive of the static website and then deploying it on the web server:

- to release the Jekyll source code, a signed Git tag needs to be created from the master branch of the source code. Website tags must follow the Calendar Versioning scheme using the tag date
- after pushing the tag, an artifact of the static website will be built by GitHub Actions and deployed as an asset of the associated GitHub release
- on the server hosting the website, a cron job will update the website hourly if a new release has been created on GitHub
- if needed, the static website can also be updated manually by executing the **`sudo deploy -f`** command from the OME website server.

CHAPTER

SIXTEEN

DEVELOPMENT OF THE OME DATA MODEL

Warning: This page is being restructured following the decoupling of the data model from the Bio-Formats code repository. An updated version will be published shortly.

16.1 Introduction

This is a document describing a way to work and publish the OME model schema on the OME website, based on observations of the 2016-06 release being performed; that release version is used in the examples below. Throughout the process it is important to not just copy and paste, but to understand what is actually being done and why. The text below is not quite yet a step-by-step guide, more a set of explanations that should make the necessary steps clear. Many of the command-line scripts below assume that you start at the top level of your Bio-Formats repository, and they include some `/path/to` directories for you to adjust as appropriate.

16.2 Schema development

16.2.1 Clean the repository

In working with the Bio-Formats git repository, first clean the unnecessary files away so that they cause no confusing clutter that wastes your time. From the top-level `bioformats` folder, while **`ant clean`** and **`mvn clean`** are both fine approaches, the most thorough is **`git clean -dfx`**.

16.2.2 Major or minor release?

A minor release of the OME model schema may suffice for changes like adding new legal values to an existing enumeration. A release *must* be major if,

- some documents that validate under the current release will not validate under the new one (a major "data-level" change)
- some terms in the schema have changed meaning and may thus be acted on differently (a major "information-level" change)

A major release requires changing the schema's namespace. For a minor release it suffices to increment the value of the *version* attribute of the *xsd:schema* element, leaving the namespace unchanged.

See also:

PR #1999 (major schema change), PR #2553 (minor schema change)

16.2.3 Create the new schema directory

Note: This subsection is for major releases only. A minor release can reuse the schema directory for the current release, so skip over this part.

For the schema release process a high fraction of the necessary work occurs in Bio-Formats' *components/specification* directory. Inside there, *components/specification/released-schema* contains a subdirectory for each schema, even before its actual release.

To preserve the version history, the creation of the new schema directory is performed across a pair of commits. First, the latest patch gets its new name, for example:

```
cd components/specification/released-schema
mkdir 2015-01
git mv 2013-10-dev-5/catalog.xml 2015-01
git mv 2013-10-dev-5/*.xsd 2015-01
```

then, for the subsequent commit, remember to do:

```
git checkout HEAD^ 2013-10-dev-5/catalog.xml
git checkout HEAD^ 2013-10-dev-5/*.xsd
git add 2013-10-dev-5/*
```

to restore the released files from the latest patch. In this way, the files of the actually released schema retain their version history.

For an even later commit one may consider:

```
git rm -r 2013-10-dev-?
```

which removes the patch versions if no longer desired.

Note: It may make sense to adjust the above **`git mv`** commands to move fewer files to the release directory. For instance, `OMERO.xsd` is not used by the OME schema so need not be released alongside it in the `2015-01` directory if has not been changed since the previous release.

16.2.4 Catalog files

The released schema directories have catalog files that list their contents. For instance:

```
cd components/specification/released-schema
find . -name catalog.xml
```

Within each commit, each catalog file should be kept up to date with changes made in that same directory, such that the catalogs always list exactly the available schema definitions.

16.2.5 XML transforms

The changes made to the released schemas should be accompanied by changes to the XML transforms in *components/specification/transforms*. For major releases use **`git mv`** in renaming the upgrade and downgrade for the latest patch. Remember to restore the originals in a later commit, as above when restoring the schema definition files for the latest patch.

For minor releases it suffices to adjust the existing upgrade and downgrade transforms for the current release. Remember that users may be downgrading from an earlier minor version than this newest version.

The transforms' analog of the catalog files is *components/specification/transforms/ome-transforms.xml* which should describe the transforms in its directory for that commit.

16.2.6 Search and replace

Note: This subsection is for major releases only. A minor release reuses the current release and patch versions, so skip over this part.

There are various references to the latest patch version and even the latest release version to be updated; the whole Bio-Formats repository requires checking.

In replacing the "2013-10-dev-5" schema references within the actual schema definition files in the new `released-schema/2015-01` directory, also update the copyright date in their headers, and the date in `ome.xsd`'s first *xsd:documentation* tag. Likewise, with the XML transforms, update the copyright date in their headers, and in the attributes appearing near the start of *components/specification/transforms/ome-transforms.xml*.

Other files in which to fix the schema version include:

- *components/autogen/build.properties* and *ant/xsd-fu.xml* for code generation
- the Project Object Model, Maven's *pom.xml*
- the *components/specification/publish* because of the HTML within
- checks in the Bio-Formats code for the latest schema version, including various Java classes (*version.equals*, *SCHEMA_LOCATION*, etc.)

Avoid changing:

- sample files in *components/specification/samples*
- old schema releases

16.2.7 Testing

Once the above changes have been made and committed, it is time to test. This requires having various prerequisites installed for Bio-Formats development, including for the C++ implementation. Before each test, *clean the repository*:

```
git clean -dfx
ant test
git clean -dfx
mvn test
git clean -dfx
TMPDIR=/tmp/bf-build-`date +%s`
mkdir $TMPDIR
pushd $TMPDIR
```

(continues on next page)

(continued from previous page)

```
cmake `dirs +1`
make
ctest -V
popd
```

You may care to give **make** an additional `-j` option specifying the number of cores to use in parallelizing the build. Note that the **ctest** step can take a long time.

16.3 Sample files

16.3.1 OME-XML sample files

Once the schemas and transforms are moved and named to fit the release version, then the sample files can be upgraded. A new copy of the sample files is created in a new directory, updated to the new schema using **xsltproc** with the new transform, then pretty-printed with **xmllint** or similar. A sufficient command-line approach is:

```
cd components/specification/samples
for SRC in `find 2015-01 -type f -name '*.ome' -o -name '*.xml'`
do DEST=`echo $SRC | sed -e 's/^2015-01/2016-06/'`
   mkdir -p `dirname $DEST`
   <$SRC xsltproc ../transforms/2015-01-to-2016-06.xsl - | xmllint --format - >$DEST
done
```

The OME-TIFF files require special handling, as they do not have an automatic update tool. First, identify them and copy them to the new directory:

```
find 2015-01 -name '*.ome.tiff'
cp 2015-01/set-1-meta-companion/*.ome.tiff 2016-06/set-1-meta-companion/
```

Next, each OME-TIFF file must be edited to have the schema version changed to that of the new release. They are binary files so choice of editor is important; the other non-text data must be preserved. One of several suitable options is Emacs' Hexl mode.

16.3.2 OME-TIFF sample files

Sample files for each schema release version are available under https://downloads.openmicroscopy.org/images/OME-TIFF/. The sample files in the previous release's directory, and the multi-file samples in its `tubhiswt-*` directories, are upgraded to the new schema using **bfconvert** from the updated Bio-Formats repository: in that repository use **ant tools** to generate the necessary `bioformats_package.jar` Java archive file. The sample files from the subdirectories are provided also as compressed "zip" archive files. The files in the `bioformats-artificial` subdirectory are generated by other Bio-Formats classes. Putting these facts together, setting up the new "2016-06" samples folder is easily achieved:

```
mkdir 2016-06
mkdir 2016-06/binaryonly
mkdir 2016-06/companion
mkdir 2016-06/modulo
cd 2015-01
for i in *.ome.tif*
do /path/to/bioformats/tools/bfconvert $i ../2016-06/$i
```

(continues on next page)

(continued from previous page)

```
done
cd binaryonly
for i in *.ome.tif*
do /path/to/bioformats/tools/bfconvert $i ../../2016-06/binaryonly/$i
done
cd ../companion
for i in *.ome.tif*
do /path/to/bioformats/tools/bfconvert $i ../../2016-06/companion/$i
done
cd ../modulo
for i in *.ome.tif*
do /path/to/bioformats/tools/bfconvert $i ../../2016-06/modulo/$i
done
for i in tubhiswt-?D
do mkdir ../2016-06/$i
   FROM=`ls $i | head -n 1`
   TO=`echo $FROM | sed -e 's/_C0/_C%c/ ; s/_TP0/_TP%t/'`
   /path/to/bioformats/tools/bfconvert $i/$FROM ../2016-06/$i/$TO
done
cd ../2016-06
for i in tubhiswt-?D ; do zip $i.zip $i/* ; done
mkdir bioformats-artificial
cd bioformats-artificial
BF_PROG=loci.formats.tools.MakeTestOmeTiff /path/to/bioformats/tools/bf.sh
for i in *.ome.tif ; do zip $i.zip $i ; done
```

Review the new sample files to ensure that they look correct. At the end of the next step they are published online.

Binary Only and companion files: The OMETiffWriter does not support the writing of sample BinaryOnly or Companion files. If the only required update is to change the schema version then the files may be edited with a Hex Editor. Any additional editing may change the length of the file and invalidate the tiff header.

In instances where more detailed changes are required to BinaryOnly samples:

- Write a short program using OMETiffReader and Writer to read and write the existing sample
- Using debugging tools, inject the desired OME XML prior to saveComment in OMETiffWriter close function
- Ensure when modifying the XML that the UUID values are correct
- Verify that files pass using xmlvalid and tiffinfo commands

16.4 Schema publication

16.4.1 Schema release

Once a specification change has been made into an *ome-model* release, the *publish* script in the https://github.com/ome/schemas repository automatically generates new schemas pages published at https://www.openmicroscopy.org/Schemas/.

16.4.2 Generated documentation

Documentation for the released schema must be generated from the `ome.xsd` definition file. The XML editor oXygen is recommended for this task, and requires the schema definitions to have been published online as described above. To build the generated documentation for a given release:

```
/Applications/oxygen/schemaDocumentationMac.sh https://www.openmicroscopy.org/Schemas/
↪OME/$RELEASE/ome.xsd -cfg:components/specification/omeOxygenDocConfig.xml
```

Check that the documentation generated in the new `output` directory all looks correct.

The https://ci.openmicroscopy.org/job/SCHEMA-documentation job will generate the oXygen documentation for a given version of the schema. Once generated, this documentation can be transferred to a `$RELEASE` subfolder of `/var/www/html/www.openmicroscopy.org/specification/schema_doc` on *web-prod*.

CHAPTER

SEVENTEEN

CHANGING THE SCHEMA

17.1 Background

OMERO.server stores data in PostgreSQL, a relational database system. The data schema defines what data is stored and how, and new major versions of OMERO may change that schema. Database upgrade scripts transform data from an older version of OMERO so that it conforms to the new schema.

Sometimes, a pull request on GitHub against the develop branch of OMERO may change the code base in ways that cause changes in the resulting database schema. This is a problem because the schema must then be updated, and other developers need to know that code from that pull request may cause problems unless they update their database accordingly. To make sure that these database updates happen when necessary, if your pull request affects the schema then you **must** increment the database patch number and provide an updated schema as described below.

Changes to the OME-XML model typically require corresponding changes in the OMERO data schema as defined in its XML mappings files. These feed into OMERO's database schema so this process is then required.

17.2 Patch number conflicts

It is possible that another person may also be working on a pull request that changes the schema and increments the database patch number. This is unfortunate because if their pull request is merged it will be as if your pull request does not change the patch number. Others may then unwittingly attempt to use your code with an inappropriate database. If you are considering model changes, it is wise to discuss this with the core OME developers in advance. When working on a schema-changing pull request, first ask or check if yours will be the only one that includes a schema change.

17.3 Model object proxies

Changes to model objects that are passed from the server to clients may require corresponding changes to be made to the IceMapper class so that the client-side proxy objects are properly populated.

For example, commit 8815a409 adds fields to the *Roles* class in the server's System.ice whose instances can be passed to clients via the admin service API so a further commit 2426042a was needed to populate those fields in the proxy object.

17.4 Database patch numbers

omero-model.properties contains a configuration setting for `omero.db.patch`. An existing OMERO database records the patch number of its schema, as demonstrated from the `psql` shell:

```
omero=> select currentpatch from dbpatch;
 currentpatch
--------------
            4
(1 row)
```

indicating that a database is on patch version 4. Correspondingly, in https://github.com/ome/omero-model,

```
$ grep ^omero.db.patch= src/main/resources/omero-model.properties
omero.db.patch=4
```

By incrementing the patch number with each schema change, OMERO.server is prevented from attempting to use a database whose schema does not match its code.

17.5 Updating the schema and the SQL scripts

Warning: This section is **NOT** up-to-date. Steps like using `build-schema` will not work with 5.5.0

Users may wish to upgrade their database from an older version of OMERO to one that has your new schema. SQL upgrade scripts are provided to allow users to upgrade easily without having to understand the schema changes themselves, and part of the upgrade script will involve making the schema changes entailed with your pull request. The https://github.com/ome/openmicroscopy/blob/develop/sql/README.txt file describes where to find the appropriate script for you to adjust. SQL upgrade scripts must be supplied as part of the code changes to upgrade the database from:

- the last release database, e.g. `sql/psql/OMERO5.1DEV__5/OMERO5.0__0`,
- the previous patch's database, e.g. `sql/psql/OMERO5.1DEV__5/OMERO5.1DEV__4`.

In your git branch with the code that requires a schema change, edit omero-model.properties and increment the value of `omero.db.patch`. For instance, in the above example, edit the file so that

```
$ grep ^omero.db.patch= src/main/resources/omero-model.properties
omero.db.patch=5
```

Move the previous patch's SQL scripts into their new directory.

```
$ git mv sql/psql/OMERO5.1DEV__4 sql/psql/OMERO5.1DEV__5
```

Restore the upgrade to that previous patch.

```
$ mkdir sql/psql/OMERO5.1DEV__4
$ git mv sql/psql/OMERO5.1DEV__5/OMERO5.1DEV__3.sql sql/psql/OMERO5.1DEV__4/OMERO5.1DEV__
↪3.sql
```

Build OMERO.server with your code that changes the schema, then use the `build-schema` build target to update the SQL scripts in the new `sql/psql/OMERO5.1DEV__5` directory.

```
$ ./build.py build-schema
```

Now, when you use **omero db script** in setting up a database for your modified server, the generated SQL script creates the new schema that your code requires. Use this script to set up your database so that you can start OMERO.server and test your changes thoroughly.

A combination of `sql/psql/OMERO5.1DEV__4/OMERO5.1DEV__3.sql` and the changes within `sql/psql/OMERO5.1DEV__5` that **git diff** reports should help you to create a new `sql/psql/OMERO5.1DEV__5/OMERO5.1DEV__4.sql`.

When you commit your code and issue a pull request, include the changes to omero-model.properties and https://github.com/ome/openmicroscopy/tree/develop/sql/psql among the commits in the pull request.

CHAPTER EIGHTEEN

PYTHON DEVELOPMENT

18.1 Release process

Prior to release a new version, the maintainer:

- can create a GitHub project and/or milestone if required listing the PRs to be considered in the upcoming release (optional)
- must create an entry in the CHANGELOG: PRs included in the release should be listed in the CHANGELOG with a link to the PR.
- must have the ability to push the generated tag to `origin`.

Many OME repositories use bump2version to manage version numbers. These can be identified by the presence of a *.bumpversion.cfg* file at the top of the repository.

First fetch and checkout master or main branch:

```
$ git fetch origin
$ git checkout master
$ git rebase origin/master
```

You will need to be able to sign commits with *gpg*. Test this with:

```
$ echo "test" | gpg --clearsign
```

Compare the current version in *.bumpversion.cfg* with the last release version to see if the current difference represents a patch release. If any PRs are merged that would require the next release to be a `major` or `minor` version (see semver.org) then that PR can include a version bump created via:

```
$ bumpversion --no-tag minor|major
```

If this hasn't been performed prior to release and you wish to specify the next version number directly when creating the release, this can be achieved with:

```
$ bumpversion --new-version 5.9.0 release
```

If the version is already suitable, simply run:

```
$ bumpversion release
```

This will remove the `.dev0` suffix from the current version, commit, and tag the release.

To switch back to a development version run:

```
$ bumpversion --no-tag patch
```

NB: this assumes next release will be a `patch` (see below). To complete the release, push the master branch and the release tag to origin:

```
$ git push origin master v5.8.0
```

CHAPTER NINETEEN

PUBLISHING TO PYPI

Many of the OME Python repositories use GitHub actions to publish to Pypi when a new tag is created and pushed to GitHub. This is typically specified in a file such as *.github/workflows/publish_pypi.yml*.

Information specific to developing OMERO, the OME Data Model and file formats, and Bio-Formats can be found in their respective developer documentation sections:

- OMERO developer documentation
- Bio-Formats developer documentation
- OME Data Model and File Formats documentation

If you have any questions, please see our Support page for ways to get in touch.

INDEX