{{lastupdated_at}} by {{lastupdated_by}}

Code release

This page summarizes the steps for a code release. A code release can only be performed by an authorized code integrator. The code release consists in the following steps:

- Merge devel into the release branch
- Set release information
- Test the release branch
- Create a tarball
- Test tarball
- Publish code
- Merge release branch into master branch
- Tag the release
- Merge the master branch into the devel branch
- Bring the devel branch ahead of the master branch

Merge devel into the release branch

The first step in a code release is to branch off the actual code in devel and merge it into the release branch. This is done for an authorized code manager using the following sequence of commands:

```
git checkout release
git merge devel
```

Set release information

Before releasing the code, you have to modify a number of files to incorporate release information in the code distribution and to update the code version number. This may have been done before during the development. The command sequence for updating release information is:

```
git checkout release
nano configure.ac
  AC_INIT([gammalib], [0.10.0], [jurgen.knodlseder@irap.omp.eu], [gammalib])
nano gammalib.pc.in
  Version: 0.10.0
nano ChangeLog
  ... (update the change log) ...
nano NEWS
  ... (update the NEWS) ...
nano README
  ... (update the README) ...
nano doc/Doxyfile
  PROJECT_NUMBER         = 0.10.0
nano doc/source/conf.py
  # The short X.Y version.
  version = '0.10'
  # The full version, including alpha/beta/rc tags.
  release = '0.10.0'
git add *
git commit -m "Set release information for release X.Y.Z."
git push
```

The nano commands will open an editor (provided you have nano installed; otherwise use any other convenient editor). The lines following the nano command indicate which specific lines need to be changed. Changes in the files ChangeLog, NEWS and

README need some more editing. Please take some time to write all important things done, as GammaLib users will later rely on this information for their work! ChangeLog and NEWS should have been updated while adding new features, but you probably need to adapt the version number and release date.

Test the release branch

Use the release branch in the central repository to perform extended code testing. The code testing is done by the release pipeline in Jenkins. Make sure that this pipeline runs fully through before continuing. If errors occurred in the testing, correct them and run the pipeline again unit the pipeline ends with success. Don't forget to push any changes into the repository.

In addition to the pipeline, apply the testall.sh script in the gammalib folder to check all test executables and scripts.

Create a tarball

Use the script dev/release_gammalib.sh to build a tarball of the GammaLib release. This is usually done on the kepler.cesr.fr server.
Should be moved into Jenkins.

Test tarball

Test that the tarball compiles and all checks run. Type

```
tar xvf ../gammalib-0.10.0.tar.gz
cd gammalib-0.10.0
./configure
make
make check
```

Should be moved into Jenkins.

Publish code

TBW

Merge release branch into master branch

Now you can merge the release branch into the master branch using the commands:

```
git checkout master
git merge release
```

Tag the release

Now you can tag the release using

```
git tag -a GammaLib-0.10.0 -m "GammaLib release 0.10.0"
git push --tags
```

We use here an annotated tag with a human readable tag message. Please use always the same format. In the above example, the GammaLib version X.Y.Z was 0.10.0.

Merge the master branch into the devel branch

Now we merge the master branch back in the devel branch to make sure that the devel branch incorporates all modifications that were made during the code release procedure.

```
git checkout devel
git merge release
git push
```

Bring the devel branch ahead of the master branch

You now should being the devel branch ahead of the master branch. This means that the devel branch should have an additional commit with respect to the master branch. This is not mandatory, but it assures later that the master branch and the devel branch will never point to the same commit. In that way, git clone will always fetch the devel branch, which is the required default behavior.

To bring the devel branch ahead, execute following commands:

```
git checkout devel
nano AUTHORS
git add AUTHORS
git commit -m "Bring develop branch ahead of master branch after code release."
```

The nano AUTHORS step opens the file AUTHORS in an editor. Just add or remove a blank line to make a file modification.

Now you're done!