Release Management

DEPLOYING SNAPSHOTS

Snapshots are automatically deployed every night to the Nexus snapshot repository at https://repository.apache.org/content/groups/snapshots-group/ . There is no need to manually deploy snapshots anymore.

MAINTAINING A FIXES BRANCH

dkulp: I'm adding this section to document what worked for ME when maintaining the 2.7.x-fixes branch for the 2.7.x releases. Each Release Manager may have their own style or tools or whatever. This is not a "set in stone" type thing.

Basically, almost all development and fixes and such are usually done by the various developers right on master. Thus, the main job of the fixes branch maintainer is to triage the commits on master and merge pure fixes to the fixes branches, resolve conflicts, run the tests, and periodically deploy snapshots. For the most part, when things go well, it doesn't take too much time or effort. An hour or two every couple days is about it.

To set up, you'll want to:

1. use git branch to make a branch.
2. On the branch, create a .gitmergeinfo file with a single line of "origin/master" to say the branch will be merging from there.

In trunk/bin, there is a DoMerges.java program that assists in the merging. If the branch is setup with .gitmergeinfo, if you run it from the root directory of the checkout, it will prompt for every commit on master to see if you want to "Merge" it, "Block" it, or "Ignore" it. It displays the commit log first so you can see what was involved. You can also check the cxf-commits archive to see the full details of the commit to help decide what action to take. If you select "Merge", it will merge the change and then prompt before committing. That will allow you to look at the merge and resolve any conflicts. (or even revert it if you didn't mean to hit Merge)

PERFORMING A RELEASE

The first step is to update the release_notes.txt in the distribution/src/main/release. This file's JIRA list of solved Bugs, Improvements, etc. can be obtained from the "Road Map" JIRA tab, selecting the desired version's Release Notes, and then the Configure Release Notes button (choose Text output).

Don't manually update the POM versions from X.Y.Z-SNAPSHOT to X.Y.Z, the Maven Release Plugin commands below will automatically take care of that. Also, prior to performing the release you'll need to have your Apache LDAP information configured in your Maven settings.xml file:

```
...
<server>

   <id>apache.releases.https</id>
   <username>apacheID</username>
   <password>yourLDAPPassword</password>
</server>
...
```

Then, to actually perform the release, run the below commands.

```
mvn release:prepare -Peverything
mvn release:perform -Peverything
```

If you are performing the release on a Mac, it is advisable to add -DpushChanges=false to the "release:prepare" step above. The version of git that Apple ships with some versions of OSX has problems pushing the changes in quick succession from the release plugin and can become corrupt. Having the release plugin NOT push the changes and then running "git push -tags origin master" works around that problem.

It is recommended to name the local maintenance branches the same as the remote ones ("2.7.x-fixes", "3.0.x-fixes", ...) to avoid issue with the branch names when running the release plugin.

The above commands tag the release, update the poms versions, etc., then build it (off the tag), gpg sign and deploy everything (including source jars and javadoc jars) to the Nexus repository location. When the build is done staging, you next need to login to the Nexus repository and "close" the staging area (click on Staging Repositories in the left-side menu, select the repo you just uploaded and then select the close button.) Closing is very important. After the staging area is closed, note the URL for the staging area as you will need that for the vote.

At this point, everything "pre-vote" is done. Call the vote.

RELEASING THE ARTIFACTS

- Maven artifacts - After the vote passes, you'll need to promote that staging repository to the main location. Login to Nexus repository location to do that as well, find the staging repository and click the Release button.

- Distributions - You will need to commit the distributions into the special svn distribution area: https://dist.apache.org/repos/dist/release/cxf
  after you commit they will be live on dist.apache.org fairly quickly, but it will still take time for the mirrors to get copies. It's likely easier to make the directory via an svn command, check out just that directory, and then add the files. The dist area is rather large (400MB or so) so checking out the entire thing may be slow.

```
svn mkdir https://dist.apache.org/repos/dist/release/cxf/2.6.3
svn checkout https://dist.apache.org/repos/dist/release/cxf/2.6.3
....  add files to 2.6.3 .....
svn commit
```

The download page of a currently released version will tell you the precise files you need to upload. Basically, the -src.tar.gz, -src.zip, tar.gz, .zip files, and the .md5, .sha1, and .asc signature files of each of those. For greatest accuracy, it's best to download the files from Nexus and use those. (For example, for CXF 2.6.2, you would check this folder).

- Update the download page - around 24 hours after committing the distributions, update the download page, release notes, etc. to point at the new versions. At that stage you'll want to delete (svn rm) the previous version of each branch you uploaded to https://dist.apache.org, so if you added https://dist.apache.org/repos/dist/release/cxf/2.6.3 and https://dist.apache.org/repos/dist/release/cxf/2.5.6, you'll want to remove https://dist.apache.org/repos/dist/release/cxf/2.6.2 and https://dist.apache.org/repos/dist/release/cxf/2.5.5. (Older versions will still be available at http://archive.apache.org/dist/cxf/ and are referenced from there on the CXF download page.)

- Javadocs - the javadocs in the distribution are a limited set of javadocs useful for MOST people. However, the CXF website contains a more complete set of javadocs. To generate the docs for the site, from the source distribution (or git tag), run:

```
mvn javadoc:aggregate
mvn package -DskipTests=true javadoc:aggregate-no-fork -pl distribution
-am -Peverything -Dmaven.javadoc.skippedModules=cxf-testutils
```

  The first call (which WILL fail while trying to process some of the system tests) makes sure all the necessary things are built, code generated, etc... The second call will generate the javadoc for the site. Then copy the contents of target/site/apidocs to a new versioned directory in an svn checkout of https://svn-master.apache.org/repos/infra/websites/production/cxf/content/javadoc and commit the new version. Make sure any new files are added.

- Unpack the schemas directory from the appropriate cxf-bundle-VERSION.jar and check if any of the schemas in https://svn-master.apache.org/repos/infra/websites/production/cxf/content/schemas need updating.

UPDATING JIRA

JIRA will need to be updated:

- Add new versions that JIRA items can be created against, and mark the version(s) you just released as "released":

1. Go to the CXF JIRA Home Page and select the "Versions" left menu item, then select "Manage Versions".

2. Select the tools icon (far right side) for the version(s) you've just released and select "Release".
3. Also on this screen type in new release versions for the branch(es) you've released and plan on releasing more versions from.

- Close the JIRA items that were marked resolved for each release:

1. From the previous screen, select "Exit Administration".
2. From the CXF JIRA home page, select Versions from the left-side menu, and for each version you just released:
    a. Select version, then issues left-side menu item, then select the "Resolved" list.
    b. From the tools icon, select Bulk Change -> select all items -> Transition Issues -> Close Issues -> Confirm.