

Release Procedure

This document is intended to be kept up to date by the *DSpace Release Team*. It details the steps necessary to perform snapshot and official releases of DSpace and supporting Modules.

Table of Contents

- 1 [Release Numbering Convention](#)
- 2 [Prerequisites](#)
 - 2.1 [Verify Release Privileges](#)
 - 2.2 [Update Maven settings.xml](#)
 - 2.2.1 [Using SSH and GPG agents](#)
 - 2.3 [Java Version](#)
 - 2.4 [Unset MAVEN_OPTS, if you use it](#)
- 3 [Making a Snapshot Release \(e.g. 'dspace-x.y-SNAPSHOT'\)](#)
 - 3.1 [One Step Process](#)
- 4 [Releasing a Single Module/Project](#)
- 5 [Making an Official Release \(e.g. 'dspace-x.y' or 'dspace-x.y-rc1'\)](#)
 - 5.1 [The Day Before: Double Check Contents of all README \(and similar\) files in GitHub](#)
 - 5.1.1 [Request any CVEs from GitHub \(if security fixes are included\)](#)
 - 5.1.2 [Regenerate the LICENSES_THIRD_PARTY file](#)
 - 5.2 [Final Commits & Preparation](#)
 - 5.2.1 [Release DSpace Language Packs \(I18N\) Modules](#)
 - 5.2.2 [Double Check Language Packs' Version Ranges](#)
 - 5.2.3 [Ensure Documentation is Updated Appropriately](#)
 - 5.2.4 [Ensure you tag multiple releases in chronological order!](#)
 - 5.3 [Warning: optional profiles need to be specified at compile time](#)
 - 5.4 [Checkout Main or Branch to Release](#)
 - 5.5 [Do a Dry Run](#)
 - 5.6 [Tag and Increment Version](#)
 - 5.6.1 [What do successful tagging results look like?](#)
 - 5.6.2 [What to do if you get tagging Errors?](#)
 - 5.7 [Deploy Artifacts to Staging in Sonatype](#)
 - 5.8 [Verify and Release Staged Artifacts in Sonatype](#)
 - 5.9 [Release the Frontend \(UI\) via a GitHub Release Tag](#)
 - 5.10 [Create the PDF version of Wiki Documentation](#)
 - 5.11 [Create a new GitHub release](#)
 - 5.12 [Update demo.dspace.org](#)
 - 5.13 [Build Docker Images for Tagged Releases](#)
- 6 [After the Release is Finished](#)
 - 6.1 [Create Maintenance Branches \(after major release\)](#)
- 7 [Possible Errors you may Encounter](#)
 - 7.1 ["Could not find model file" error \(with language packs\)](#)
- 8 [Advice for future Release Coordinators](#)

Useful Sonatype Links

For lack of a better place at this time, here's some useful pages on Sonatype which detail the Sonatype Maven Release Process:

- [Sonatype Maven Repository Usage Guide](#)
- [Uploading 3rd Party Dependencies to Maven Central](#)
- [How to Generate PGP Signatures with Maven \(required for all Sonatype releases\)](#)

Release Numbering Convention

As of 2012 (starting with [DSpace 3.0](#)), DSpace has moved to a new release numbering scheme/format. Release numbers will now only consist of two numbers.

Release Numbering Scheme: [major].[minor] (e.g. 3.0, 3.1, 3.2, 4.0)

- **Major Releases:** incrementing the first number ('major') will represent a new MAJOR release of DSpace. A major release may include any or all of the following: new features, system improvements, architectural changes, bug fixes. All major releases end in ".0", so "3.0", "4.0", and "5.0" would all represent major releases.
- **Minor (Bug-Fix) Releases:** incrementing the second number ('minor') will represent a new MINOR release of DSpace. A minor release will **only include** bug fixes to an existing major release. For example, "3.1" and "3.2" would represent two minor releases which only include bug fixes to the "3.0" major release..

For more information see [DSpace Release Numbering Scheme](#)

Small Exception for Language Packs Releases

The one exception is that the Language Packs (dspace-api-lang and dspace-xmlui-lang) use the numbering convention [major].[minor].[sequence-number] (e.g. 3.0.0, 3.0.1, 3.1.0, etc.). This allows us to release new versions of the language packs more frequently than normal DSpace releases, as needed.

Prerequisites

Verify Release Privileges

To perform a release, you must have **all** of the following:

1. Write access to the DSpace GitHub repository hosted at <https://github.com/DSpace/DSpace>. (All Committers should already have this, obviously)
2. Write access to the org.dspace groupId in the snapshot and staging repositories hosted at oss.sonatype.org. If you don't already have this, you will need to:
 - a. Sign up for a [Sonatype JIRA account](#). This account will also serve as your login to the [Sonatype OSS system](#). (If you already have a Sonatype account, you can skip this step)
 - b. Ask a Committer with release privileges (e.g. a previous release manager) to request that your Sonatype account be given release privileges to the "org.dspace" GroupID. This request should be submitted via the Sonatype JIRA system in the [Open Source Project Repository Hosting](#) project.
 - c. Once Sonatype gives you the proper authorization, you should be able to login to the [Sonatype OSS system](#) using the same login /password you setup in Sonatype JIRA. You should also have access to publish new releases to the "org.dspace" GroupID.
 - d. NOTE: The full details of signing up and getting access to Sonatype are also posted online here: [Sonatype Maven Repository Usage Guide](#)
3. You **must** generate and publish your own personal Code Signing Key (required by Sonatype). Here are two sites that give hints on how to do that:
 - See [Sonatype instructions on creating a Code Signing Key](#). Or, Fedora's instructions on [Creating a Code Signing Key](#)
 - [How to Generate PGP Signatures with Maven \(required for all Sonatype releases\)](#)
 - Make sure to publish your Key file to `hkp://pgp.mit.edu`, as this is the Key Server Sonatype uses for verification:
 - (e.g.) `gpg --keyserver hkp://pgp.mit.edu --send-keys [yourKeyID]`
 - `[yourKeyID]` can be found by running the following command and copying the alpha-numeric string after the "/" on the "pub" line
 - `gpg --list-keys`
 - You can see if your key is already on that Key Server by visiting <http://pgp.mit.edu> and searching on your name

Update Maven settings.xml

DSpace's root pom.xml already has the correct staging and snapshot repositories listed in the OSS parent's '<distributionManagement>' section. In order to deploy, you will need to add your Sonatype OSS username and password to your local `~/.m2/settings.xml` file. For example:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
    http://maven.apache.org/xsd/settings-1.0.0.xsd">

  <servers>
    <!--Login info for Sonatype SnapShot repository-->
    <server>
      <id>osrh</id>
      <username>YourSonatypeUsername</username>
      <password>YourSonatypePassword</password>
    </server>
  </servers>
</settings>
```

If you don't yet have a `~/.m2/settings.xml` file, you should create one, and copy the full contents above (obviously make sure to put in your username and password).

Using SSH and GPG agents

During a release you have to type the passwords of your GPG and SSH keys very often. Every DSpace module produces several files, all have to be signed and transferred to Sonatype. GPG and SSH agents help you to avoid typing passwords again and again. To use an ssh-agent just start it, export the required environment variables and add your ssh-key. To use a gpg agent start it, export the required environment variables and add the following to your `~/.m2/settings.xml`:

```

<settings>
...
  <profiles>
    <profile>
      <id>gpg-profile</id>
      <properties>
        <gpg.useagent>true</gpg.useagent>
        <!-- If you have gpg2 instead, you can tell Maven to use it instead of 'gpg' by uncommenting the
following -->
        <!--<gpg.executable>gpg2</gpg.executable>-->
      </properties>
    </profile>
  </profiles>
...
</settings>

```

On each maven run it may ask you once about your password which is a big improvement.

Java Version

For DSpace 7.x, you must use Java 11.

Use Maven 3 or above

For more information see the Prerequisites section of the [Sonatype Maven Repository Usage Guide](#)

For DSpace 7.x, you must use Maven 3.5.4 or above. It's necessary to regenerate the LICENSES_THIRD_PARTY file (see notes below)

Unset MAVEN_OPTS, if you use it

If you normally use a MAVEN_OPTS environment variable on the machine you're using to cut the release, be sure to unset it, with this command:

```
unset MAVEN_OPTS
```

It's highly unlikely the configuration in your MAVEN_OPTS will be useful for the release. It's highly likely to cause problems. Better to be safe.

Making a Snapshot Release (e.g. 'dspace-x.y-SNAPSHOT')

For a new module (or a major modification), sometimes it can be useful to release a SNAPSHOT version to Maven. That way you can test this SNAPSHOT version in a local DSpace build (or in a Docker build) **before** you do the official release.

One Step Process

From a clean, up-to-date copy of master/branch, run the following command:

- `mvn clean javadoc:jar source:jar deploy`

You will have to enter in your GPG passphrase (which you established when you created your Code Signing Key).

The snapshot will be immediately available in the public Sonatype snapshot repository: <https://oss.sonatype.org/content/repositories/snapshots>

(NOTE: it's not possible to list the contents of the Snapshot repository, but our DSpace Parent POM references it as a source... so anything released to the Snapshot repository can be immediately tested/used by the DSpace codebase.)

Releasing a Single Module/Project

- If you'd like to release a Snapshot Release of that module, follow the instructions at: [Making a Snapshot Release \(e.g. 'dspace-x.y-SNAPSHOT'\)](#)
- If you'd like to tag & release a new version of that module, use the module instructions at: [Release DSpace Language Packs \(I18N\) Modules](#) (NOTE: These instructions obviously have some specific notes around how the Language Packs modules are versioned. You obviously don't need to follow those versioning notes. Individual modules may have their own version schemes)

Making an Official Release (e.g. 'dspace-x.y' or 'dspace-x.y-rc1')

For More Information

These same steps are also covered in the [Sonatype Maven Repository Usage Guide](#)

The Day Before: Double Check Contents of all README (and similar) files in GitHub

This is time-consuming, don't leave this task to release day

On skimming these instructions, this might look like a small thing, but it is not. It is a big information management task. Ask for help in wrangling/verifying the license information, and, if at all possible, DO NOT leave this job for release day.

Make sure that the contents of all README, LICENSE, LICENSES_THIRD_PARTY, NOTICE files are up-to-date in GitHub. These files reside in [\[dspace-src\]](#). If anything is out-of-date, make sure to update it and commit the proper changes before continuing.

Request any CVEs from GitHub (if security fixes are included)

If this release includes fixes to any [draft security advisories](#), make sure to request CVEs from GitHub as early as you can. As of 2022, GitHub's policies currently say they will respond within 3 working days. Since the announcement of the release requires the CVEs, you'll want to make sure that you have the CVEs assigned as early as reasonably possible.

(Keep in mind requesting CVEs does NOT make the security advisories public. They will not become public until you publish them. We recommend not publishing the security advisories until the release is already completed and the announcement is about to go out.)

Regenerate the LICENSES_THIRD_PARTY file

The latest version (v2.0.0) of license-maven-plugin requires Maven v3.5.4 or above to run these commands.

The "LICENSES_THIRD_PARTY" file is now autogenerated via a Maven command (using the [codehaus license-maven-plugin](#)). Simply run the following from your local source directory to re-generate this file:

```
# Install latest version of all dependencies in local cache
# (Only necessary if you haven't run this recently)
mvn -U clean install

# Regenerate LICENSES_THIRD_PARTY file
mvn verify -Dthird.party.licenses=true
```

On completion, a new, updated version of the LICENSES_THIRD_PARTY file will be written to your source directory. Please double check this file or "git diff" it to see if the changes look reasonable. Here are some things to especially be on the lookout for:

- If any dependencies are listed with an "UNKNOWN" license, then that means that dependency failed to specify its OS License in their own Maven POM file. We will need to manually lookup the license for that project, and manually add it to our [src/main/license/LICENSES_THIRD_PARTY.properties](#) file which corrects all "UNKNOWN" licenses. Finally, rerun the command above to regenerate the new LICENSES_THIRD_PARTY based on this update.
- If any dependencies are listed under an [INCOMPATIBLE License](#) (GPL, AGPL, etc), then we need to take a closer look at that dependency. It is possible that the dependency is dual-licensed and therefore may be listed multiple times in the generated LICENSES_THIRD_PARTY file. If so, that's fine. If not, we may need to **remove** that dependency prior to the release.
- If any Open Source Licenses are listed under multiple names (e.g. "BSD" vs. "BSD License" vs. "BSD licence"), then we may need to update our POM configurations for the [codehaus license-maven-plugin](#) to tell it to merge licenses of those names into one. Those configurations are in the Parent POM under the <licenseMerges> tag of this plugin: <https://github.com/DSpace/DSpace/blob/master/pom.xml#L406>

If the file was updated, commit it.

Final Commits & Preparation

Release DSpace Language Packs (I18N) Modules

Before performing an official release, you should see if the DSpace Language Packs (i18n modules) need an updated release. *The easiest way to check if they need to be released is by checking to see if any commits have occurred since the previous release (see below for links).* Please note that you can release these I18N Modules on the *same day* as the main DSpace release. The DSpace [parent pom.xml](#) is now configured to also check [Sonatype's Release Repository](#) for any Maven artifacts (so you do NOT need to wait for the I18N modules to appear in Maven Central)

At the moment the i18n modules are maintained in two separate GitHub projects. There are currently two i18n modules you will need to release:

- [dspace-api-lang](#) - Check if any [new commits have occurred on 'dspace-api-lang'](#) since the last release. *(Also required for 7.x and above!)*
- [dspace-xmlui-lang](#) - Check if any [new commits have occurred on 'dspace-xmlui-lang'](#) since the last release. *(For v6.x or below only)*

Version Numbering Convention for Language Packs

Note that the version numbering convention for Language Packs is always the same as the current DSpace release, with an additional `[sequence-number]`. For example, the i18n modules for 3.0 were numbered as follows: 3.0.0, 3.0.1, etc.

Older Branches Do Not Update Language Packs

Language pack updates are not back-ported. If you are making a security release for an older branch of DSpace, there will be no language pack commits to release. Continue with [Final Commits & Preparation](#), below.

For each module, perform the full release steps that follow. To save space, the steps are only listed for one of the modules (but don't forget to run it for **both** language packs):

1. Checkout the Language Pack Module:
 - a. `git clone git@github.com:DSpace/dspace-api-lang.git dspace-api-lang`
 - b. `cd dspace-api-lang`
 - c. `git checkout main`

NOTE: always release language packs from the main branch -- we do not use a maintenance branch for language packs.
2. [Do a Dry Run](#): `mvn release:prepare -DdryRun=true`
3. [Tag and Increment Version](#): `mvn release:prepare -Dresume=false`
 - Make sure to assign a version number of the format: [major].[minor].[sequence-number] (e.g. 5.0.0, 5.0.1, etc for 5.0 releases of language packs)
 - NOTE: The release process should suggest the correct version number by default
4. [Deploy Artifacts to Staging in Sonatype](#): `mvn release:perform`
5. [Verify and Release Staged Artifacts in Sonatype](#) (see instructions at link)

Once both Language Packs have been released, you can **immediately** perform the DSpace release. You do not need to wait for them to appear in [Maven Central](#), as our DSpace parent pom.xml will find them in [Sonatype's Release Repository](#) immediately.

Double Check Language Packs' Version Ranges

NOTE: if you're skimming these instructions, you may be tempted to think you've already handled this step, because you have already released new language packs, as detailed above. If you think so, you probably have NOT yet completed the steps below. The steps below tell DSpace what version of language packs to use. The language packs you've released following the steps above won't ever get used if you don't do the steps below. This is an easy thing to miss. Don't. Just check, to be sure.

Once the Language Packs are released, you will probably need to modify the DSpace root pom.xml (<https://github.com/DSpace/DSpace/blob/master/pom.xml>) to reference the new version of the Language Packs. This should be similar to the following:

If possible, you'd only want to commit this *after* the i18n modules are available in the Maven Repository. But, if you are in a rush, you can commit this change earlier (though be warned that this will break the build process for anyone who hasn't manually installed the i18n modules to his/her local ~/ .m2/ directory).

In the main pom.xml, provide the proper version range for each language pack. In the below example, we are saying to use **any** language pack version which is **at least** version 7.0.0, but is **less than** version 8.0.0:

```
<!-- DSpace Localization Packages -->
<dependency>
  <groupId>org.dspace</groupId>
  <artifactId>dspace-api-lang</artifactId>
  <version>[7.0.0,8.0.0)</version>
</dependency>
```

Ensure Documentation is Updated Appropriately

Hopefully, you've already been talking with others about getting Documentation updated! 😊

You should also double check that the following "main pages" are updated in the Documentation:

- **Release Notes**- Should contain a very basic overview of the Release. Make sure the Release number is updated here!
 - NOTE: For minor releases (bug-fix-only releases), you may want to leave all information about the previous major release, and just enhance the content to state that this was a bug-fix release, and list any new contributors, etc.
- **Installation** - Obviously make sure the Installation Documentation is updated for this Release
- **Upgrading a DSpace Installation** - Same for the Upgrade Documentation, make sure it's up to date
- **History**- Make sure the online History for this latest Release is included. This section just links to all the tickets/PRs closed under the release "milestone" in GitHub.

Obviously, this is just a brief reminder of important areas of Documentation which always require updates. There's surely other areas, like **Configuration** section, which will require some updates for your release.

Ensure you tag multiple releases in chronological order!

If you are performing multiple releases at once (e.g. backporting security or bug fixes), it is **IMPORTANT** to tag your releases chronologically. For example, the backported fixes to 3.x should be tagged **BEFORE** 4.x which should be tagged **BEFORE** 5.x. The reason for this is that the timestamp of the tag determines the **ORDERING** of the releases in GitHub. So, in order for the 5.x release to appear **after** the backported releases, it needs to be released **LAST**. **The last tagged release will become the "Latest Release" in GitHub.**

Warning: optional profiles need to be specified at compile time

Just including a little warning about this up front. The following optional modules need to be specified with every mvn command below. We will make an effort to keep this list up to date, but you should verify it before you cut a new release. Things change. Forgetting an optional module means you'll have to cut another release.

```
mvn {target} -Drelease
```

```
# NOTE: for DSpace 7.x, you MUST use the "-Drelease" flag in all commands. It will automatically release all modules.
```

Checkout Main or Branch to Release

Checkout a fresh copy of the to-be-released version either from a branch or main. For example:

```
git clone git@github.com:DSpace/DSpace.git dspace-release
cd dspace-release
git checkout main
```

Note: do not just re-use an old working copy of the DSpace Main branch, for obvious reasons, you don't want your own work in progress sneaking into the release. It's also important to use the SSH repository path as noted above (NOT the https URL), otherwise you will be prompted for your GitHub credentials during the release process. More than once. Save yourself some time, be sure to use the SSH path.

Note: if you are doing a maintenance release, you will need to check out the maintenance branch, and not the main branch. In this case, the example above would instead read:

```
git clone git@github.com:DSpace/DSpace.git dspace-release
cd dspace-release
git checkout dspace-7_x

(or whatever the current maintenance branch might be named)
```

Do a Dry Run

This step is not required, but performs a useful sanity check without committing any changes. From your clean, up-to-date copy of master/branch, run the following command (from [dspace-src]):

```
# For DSpace 7.x or above (the "-Drelease" flag is required and it selects all modules to release)
mvn release:prepare -DdryRun=true -Drelease
```

You will have to enter in your GPG passphrase (which you established when you created your Code Signing Key).

If GPG is not working, you can "prime" the GPG agent by signing something like this:



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

If you notice an issue or an error occurs, you can re-run the Dry Run using the following command:



- `mvn release:prepare -DdryRun=true -Dresume=false -Drelease`

You can also clean up any of the release files that the Dry Run created, and just re-run it.

- `mvn release:clean -Drelease`
- `mvn release:prepare -DdryRun=true -Drelease`

Tag and Increment Version

This step will set the version declared in the project's pom.xml files, commit the changes to master/branch, tag the release, and finally, check in the master /branch change that increments the next development version (e.g. x.y-SNAPSHOT) in the pom.xml files. Run the following (from [dspace-src]):

```
# For DSpace 7.x or above (the "-Drelease" flag is required and it selects all modules to release)
mvn release:prepare -Dresume=false -Drelease
```

(Optionally, you may also include the parameters -Dusername=YourGitHubUsername -Dpassword=YourGitHubPassword at the end of the above command, though I've not found these to be necessary)

The above command will ask you three basic questions. Here are sample answers for DSpace 3.0:

"What is the release version for: XXX" : **3.0**

- *NOTE: This is the release # to put in the final tagged Maven POMs.*
- Examples:
 - For a "3.0" final release, it should look like: 3.0
 - For a "3.0 Release Candidate #1" release, it should look like: 3.0-rc1

"What is SCM release tag or label for: XXXX" : **dspace-3.0**

- *NOTE: This is the [tag name in GitHub](#) .*
- Examples:
 - For a "3.0" final release, it should look like: dspace-3.0
 - For a "3.0 Release Candidate #1" release, it should look like: dspace-3.0-rc1

"What is the new development version for: XXXX" : **3.1-SNAPSHOT**

- *NOTE: This is the **next** release number which all POMs should be incremented to on "master" branch.*
- Examples:
 - For a "3.0" final release (3.0), the next version should be: 3.1-SNAPSHOT
 - For a "3.0 Release Candidate #1" release (3.0-rc1), the next version should be: 3.0-rc2-SNAPSHOT

You will also have to enter in your GPG passphrase (which you established when you created your Code Signing Key).

Many JavaDoc WARNING messages will scroll by



As the release process scrolls by, you likely will see a LOT of "WARNING" messages. Don't worry, these should be just Javadocs warnings, and can be safely ignored. Just be patient, and see if it all succeeds in the end. We know it's nerve-racking, but it will all be OK.

What do successful tagging results look like?

Assuming everything worked right, you should see ALL the following changes in GitHub:

1. A newly tagged version of DSpace under: <https://github.com/DSpace/DSpace/tags>
 - a. For example, if you are releasing "3.2" you should see a newly listed tag "[dSPACE-3.2](#)" in the list of tags above.
2. The primary "pom.xml" file in that newly tagged version should have a <version> tag that correspond to the newly released version.
 - a. For example, if you are releasing "3.2", then the "[dSPACE-3.2](#)" tag's main "pom.xml" should have a [<version>3.2</version>](#) tag
3. The primary "pom.xml" file in the original branch ("master" or a ".x" branch) should now be updated to the next SNAPSHOT version
 - a. For example, if you are releasing "3.2", then the [original "dSPACE-3.x" branch's pom.xml file should now have a <version>3.3-SNAPSHOT</version>](#) tag
4. The results from Maven look similar to this. (Don't worry about the "SKIPPED" messages, those are normal, as the actual release process just runs from the "DSpace Parent Project")

```

[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] DSpace Parent Project ..... SUCCESS [2.320s]
[INFO] DSpace Services Framework :: API and Implementation SKIPPED
[INFO] DSpace Kernel :: API and Implementation ..... SKIPPED
[INFO] DSpace Addon Modules ..... SKIPPED
[INFO] DSpace Kernel :: Additions and Local Customizations SKIPPED
[INFO] DSpace XML-UI (Manakin) ..... SKIPPED
[INFO] DSpace XML-UI (Manakin) :: Local Customizations ... SKIPPED
[INFO] DSpace LNI ..... SKIPPED
[INFO] DSpace LNI :: Local Customizations ..... SKIPPED
[INFO] DSpace JSP-UI ..... SKIPPED
[INFO] DSpace JSP-UI :: Local Customizations ..... SKIPPED
[INFO] DSpace SWORD ..... SKIPPED
[INFO] DSpace SWORD :: Local Customizations ..... SKIPPED
[INFO] DSpace SWORD v2 ..... SKIPPED
[INFO] DSpace SWORD v2 :: Local Customizations ..... SKIPPED
[INFO] DSpace SOLR :: Local Customizations ..... SKIPPED
[INFO] DSpace OAI 2.0 ..... SKIPPED
[INFO] DSpace OAI 2.0 :: Local Customizations ..... SKIPPED
[INFO] DSpace Assembly and Configuration ..... SKIPPED
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----

```

What to do if you get tagging Errors?

If you receive a project dependency error

 The `mvn release:prepare` command may fail to compile part way through the process, complaining that an internal project dependency is not met. If this occurs, don't worry. You just may need to ensure those dependencies get installed to your local cache. To fix this, you should be able to run the following:

1. `mvn release:clean` (clean out any files created by the previous "mvn release:prepare")
2. `mvn release:prepare -Drelease`

Backing out of changes

 If backing out of this step is needed for any reason, the following will restore the github repository and your working copy to the state it was previously in:

1. `mvn release:rollback`
 - **Warning:** running a "mvn release:rollback" will perform an immediate, automatic push to GitHub master, rolling back any previously committed POM version changes.
 - If you want to avoid the immediate, automatic push to GitHub master, you may be able to use "`mvn release:clean`" to just clean up the locally made release preparations.
 - If the rollback fails for any reason, you can also run the following to simply revert all POMs back to a specific version:
 - `mvn versions:set -DnewVersion=[version-to-revert-to] -DgenerateBackupPoms=false -Drelease`
 - This will revert POMs to the specified version in your local copy. You'll then need to commit the changes and push them up to GitHub
2. Now delete the tag in GitHub (local & remote)
 - `git tag -d dspace-x.y`
 - `git push origin :refs/tags/dspace-x.y`

Deploy Artifacts to Staging in Sonatype

This step will sign, checksum, and push all release artifacts (including javadocs and sources) to the Sonatype staging repository (<http://oss.sonatype.org/>). Run the following (from [dspace-src]):

```

# For DSpace 7.x or above (the "-Drelease" flag is required and it selects all modules to release)
mvn release:perform -Drelease

```

You should be prompted by Maven to specify your GPG passphrase (which you established when you created your Code Signing Key). If you run into any issues, it's possible to specify your GPG key and passphrase as arguments to the above command (e.g. `-Darguments="-Dgpg.keyname=YourKeyId -Dgpg.passphrase=YourKeyPassword"`)

If upload to sonatype stalls

In case the upload to sonatype seems to be stalled, be patient. Maven will wait for a timeout and automatically retry the upload.

If you need to re-deploy

If any errors or problems occur during the deploy, you can re-run the same `mvn release:perform` safely after fixing those issues (re-running it will just overwrite existing staged contents).

If you run into issues, or need to perform the `mvn release:perform` from a different machine, it is possible to run it simply against the created tag in GitHub.

1. Create a "dummy" `[src]/release.properties` file in your local DSpace source directory. It should simply state the basic SCM info for the tag you wish to release, e.g.

```
# Location of DSpace's SCM. Keep this as-is.
scm.url=scm\:git\:git@github.com\:DSpace/DSpace.git
# Change this value to point at the tag in GitHub
# For example, this example tells "release:perform" to perform a release to Sonatype based on the
'dspace-6.0-rc1' tag.
scm.tag=dspace-6.0-rc1
```

2. Run the specified `mvn release:perform` command (see above). Maven will then proceed to checkout the specified tag into your `[src]/target/checkout` folder, package up the release and send it off to Sonatype.

Verify and Release Staged Artifacts in Sonatype

For screenshots and more details on this step, visit the Sonatype Repository Usage Guide's section on [Releasing your artifacts](#)

1. Login to <http://oss.sonatype.org/>
2. Click "Staging Repositories" in the left column, then select the checkbox next to the staged repository on the right. The contents of it will open up at the bottom of the page.
 - a. The staged repository should begin in the "Closed" state, which means some automated verifications on the POM structure etc. have already been run. If it is not yet closed, select it and click the "Close" button.
3. Ensure that the artifacts in staging are exactly as they should be once deployed to Maven Central. Here's a few things to watch out for...
 - a. Download one (or more) of the POMs, and make sure the `<version>` tag is correct (e.g. 6.0 and not a SNAPSHOT version or similar)
 - b. Compare it against a past release in Maven Central (<https://search.maven.org/search?q=org.dspace>), making sure it has the same JARs or WARs, etc
 - c. Check if the file sizes looks reasonable (0 Bytes is probably not reasonable ;-)). You can also compare those to previous releases.
 - d. You can also verify the checksums of one or more of the JARs/WARs in Sonatype versus those that were installed into your `.m2` directory. They should be the same.

If You Need to Revert Back before Releasing

If anything is incorrect, select the staged repository and select "Drop". After the problem is resolved, you can re-deploy the artifacts to staging and verify them again. To re-deploy an already-tagged release:

```
mvn release:perform -Drelease -Dtag=dspace-x.y -DconnectionUrl=scm:git:git@github.com:DSpace/DSpace.git -Darguments="-Dgpg.keyname=YourKeyId -Dgpg.passphrase=YourKeyPassword"
```

4. If everything looks good, select the repository and select "Release". The artifacts should be synced to Maven central (<https://search.maven.org/search?q=org.dspace>) within 2 hours.

Once Released, There is No "Undo" Option

Once you select "Release", there is no way to "undo" the release. If any major issues are found, you'll have to increment the version number and perform a new bug-fix release.

- a. Keep in mind however, that the release should become almost immediately available in the public Sonatype repository: <https://oss.sonatype.org/content/repositories/releases/org/dspace/>

Release the Frontend (UI) via a GitHub Release Tag

Only required for DSpace 7.x and above. In 6.x and below, the UIs are in the same repository as the backend

1. Before running the "yarn version" command, you will need to tell your local "yarn" to NOT create a git tag. We'll tag this release ourselves:

```
# This only needs to be done once. You can check your settings via "yarn config list"
yarn config set version-git-tag false
```

2. First, increment the release in our package.json. Node.js / NPM / Yarn requires that release tags all be valid semantic versioning (<https://semver.org/>).
 - a. So, our "dspace-angular" release numbering looks slightly different than the backend release numbering. It's MAJOR.MINOR.PATCH
 - i. Major releases: 8.0.0 (would be compatible with v8.0 of the backend)
 - ii. Minor releases: 7.4.0 (would be compatible with v7.4 of the backend)

- iii. Patch releases: 7.4.1 (would be compatible with v7.4 of the backend, but with very minor patches/fixes to the frontend codebase)
- b. Increment the version by running (NOTE: This will immediately apply a git commit to update the "version" in package.json). In the below example, the current version is "7.4.0-next", and we've updated it to be "7.4.0" in preparation for the "dspace-7.4" tagged release.

```
git checkout dspace-7_x (or main)
yarn version
...
info Current version: 7.6.1-next
question New version: 7.6.1
...
(EXAMPLE for 8.0)
info Current version: 8.0.0-next
question New version: 8.0.0
```

- c. Commit the change to package.json

```
git commit -a -m "Update version tag for release"
git push upstream dspace-7_x (or main)
```

3. Create a new [Release](#) & Tag in GitHub. See <https://help.github.com/en/github/administering-a-repository/managing-releases-in-a-repository> for full instructions
- Note: Alternatively, you can choose to tag the release from command-line (via git tag), but GitHub allows you to create a new tag when creating a new release.
 - Just create a new tag (e.g. "dspace-7.4") off the current "main" branch.
4. Make sure the GitHub Release description links to the [Release Notes](#). It should also link to the Backend's GitHub Release (and visa versa). Look at past 7.x releases for examples.
5. After the release, update our package.json with the next planned version to represent that the "main" branch is now for developing the **next** release. NOTE: for Node.js / Angular, the "-next" suffix is the same as the "-SNAPSHOT" suffix used for Maven on the backend.
- a. If the next release is planned to be a major release, set the version to "[major].0.0-next" (e.g. "8.0.0-next")
 - b. If the next release is planned to be a minor release, set the version to "7.[minor].0-next" (e.g. "7.5.0-next")
 - c. You'll need to run the "version" command a second time to update package.json for our next release. In the below example, the current version is "7.4.0" and we've updated the version to be "7.5.0-next" for the next release.

```
yarn version
...
info Current version: 7.6.1
question New version: 7.6.2-next
...
(EXAMPLE for 8.0)
info Current version: 8.0.0
question New version: 8.1.0-next
```

- d. Commit the change to package.json

```
git commit -a -m "Update version tag for development of next release"
git push upstream dspace-7_x (or main)
```

Create the PDF version of Wiki Documentation

Export the latest Wiki-based Documentation as PDF.

How to Generate PDF Documentation

 See this DSpace documentation management guide: [How To Export Downloadable Docs from Wiki](#)

Create a new GitHub release

- From the GitHub UI, visit: <https://github.com/DSpace/DSpace/releases>
- Find the newly tagged release & click on "Add Release Notes"
- Add in some basic release notes (refer to prior versions for some standard text). Please be sure to provide the following information:
 - A link to the Wiki Release Notes (in the DSDOC area)
 - A link to the general documentation for this release (again in the DSDOC area)
- When you are satisfied, publish the new release!

Update demo.dspace.org

- For 6.x, see [demo6.dspace.org Notes](https://demo6.dspace.org/Notes)
- For 7.x, see [Updating DSpace 7 Demo Sites](#)

Build Docker Images for Tagged Releases

This step is now AUTOMATED via our 'docker' GitHub Action. However, you should double check DockerHub to ensure that newly tagged Docker images were auto-created, especially for:

- dspace/dspace : https://hub.docker.com/r/dspace/dspace/tags?page=1&ordering=last_updated
- dspace/dspace-angular: https://hub.docker.com/r/dspace/dspace-angular/tags?page=1&ordering=last_updated

If the tag doesn't appear in DockerHub (it may take a few hours!), then you can check our GitHub Actions for possible failures, and/or build & tag the images manually as described below.

Images can be built & pushed from command-line to [DockerHub](#). Again, use the same tag name (e.g. "dspace-7.0") as above.

Tim & Kim currently have Push access. Request it from one of them if you don't have it yet.

Build docker image

```
cd <DSpace-SRC>
--checkout tag--
docker build -t dspace/dspace:<tag> .
docker push dspace/dspace:<tag>

cd <ANGULAR-SRC>
--checkout tag--
docker build -t dspace/dspace-angular:<tag> .
docker push dspace/dspace-angular:<tag>
```

After the Release is Finished

Don't Announce Until Maven Packages Have Propagated

 You must wait for all the packages to be available at <https://search.maven.org/search?q=org.dspace> before you announce the release. Until the DSpace packages are available in the Maven repository, no one else will be able to build DSpace using Maven.

- Make sure all contributors to the release have been added to the Release Notes!
 - **Find the contributors:**
 - For bugfix releases

```
# This example is to find the list of contributors to 6.3
# It lists contributors to 6.x branch since the 6.2 tag
git shortlog -ns dspace-6_x ^dspace-6.2
```

- For major releases (or from main branch), you can use GitHub contributors pages with a date range.
 - Find date of last major release (e.g. 6.0 was released on Oct, 24, 2016)
 - Use contributors graph to find contributors between 6.0 and 7.0, e.g. <https://github.com/DSpace/DSpace/graphs/contributors?from=2016-10-24&to=2021-07-29&type=c>
- Coordinate Announcements with LYRASIS Staff:
 - You might post draft announcements to a service such as <https://gist.github.com/> and send out a call to committers for review. When finalized, DSpace releases should be announced on the dspace-community, dspace-devel and dspace-tech lists/groups.
 - Announcement on dspace.org website, twitter
 - (As necessary) Ensure that the [Download](#) page on dspace.org is updated.
 - Plus, ask dspace.org admins to upload latest documentation in PDF/HTML format
 - Announce on all DSpace mailing lists
 - Link announcement on [Home](#) of DSpace Wiki, change any version numbers listed on that page.
- Update Wiki pages, particularly these pages which refer to the Current and Next Releases:
 - [Current Release](#)
 - [Next Release Status](#)
 - [Releases](#)
 - NOTE: See [DSpace Wiki Style Guide](#) for notes on how to actually edit the above Redirect pages
- Also, update the Documentation Wiki area! Specifically:
 - [All Documentation](#) page -> Has current release info
 - Add a warning to the documentation of the newest unsupported release (e.g. the warning for [DSpace 1.7](#)) and link to our [Support Policy](#).
 - Spaces - Space directory - (i) next to the space - Space Admin - Themes - Configure Theme - Header
 - Homepage for the current Documentation (e.g. [DSpace 7.x Documentation](#)) -> Has links to download latest version of DSpace
 - (If possible) [Update the database schema diagram](#)
- Updates to GitHub:
 - Move any uncompleted PRs to the next DSpace version tag / project board
 - Close any release-specific project board (after moving any uncompleted PRs or issues elsewhere)
 - Close the release milestone (adding date of the release)

- See also "Create Maintenance Branches" section below

Create Maintenance Branches (after major release)

After a new major release (e.g. 8.0), we need to create a maintenance branch for any bug-fix releases. This allows the "main" branch to hold commits for the next major release, while bug fixes get applied to the maintenance branch.

- For example, after 8.0, you'd create a "dspace-8_x" maintenance branch, while the "main" branch would move to pre-9.0 development.

Creating a maintenance branch must be done for the backend and frontend separately.

- For the Backend (DSpace/DSpace), you can use Maven to quickly & easily create this maintenance branch
 - The easiest way to create a new branch is by using the [release:branch](#) command. This command uses the same params as "release:prepare" (see above examples), but will create a new **branch** instead of a new tag. For example:

```
# Start from current main (latest code)
git checkout main
git pull upstream main
# The examples below assume that "main" currently has a POM version of "[majorversion].1-SNAPSHOT"

# DRY RUN: Create a branch named "dspace-7_x" from main
# This will copy the existing POM version tags to the "dspace-7_x" branch.
# It will ask you what the next version is, and update to that version in the "main" branch.
mvn release:branch -DdryRun=true -Drelease -DbranchName=dspace-7_x

# If everything looks good, run it for real. This will immediately create the branch in GitHub,
# and then ask you again what new version to update the POMs on "main" to
mvn release:branch -Dresume=false -Drelease -DbranchName=dspace-7_x

# When selecting the next version for main, make sure it's a SNAPSHOT of the next major version (e.
# g. 8.0-SNAPSHOT)
```

- Double check that the POM versions are now correct in both the "main" and maintenance branches. If something is wrong, you should be able to use release:update-versions

```
git checkout [branch-to-correct]
mvn release:update-versions -Drelease
```

- For the Frontend (DSpace/dspace-angular), you will need to create the maintenance branch in a more manual fashion.
 - First, create the new branch using GitHub's UI. Go to <https://github.com/DSpace/dspace-angular/branches> and click "New Branch".
 - Give it the same name as the backend maintenance branch (e.g. "dspace-8_x")
 - Select "main" as the source branch
 - Now, check the "version" at the top of the package.json file in **both branches**. Ensure the maintenance branch version looks like "[major-version].1.0-next". Ensure the "main" branch looks like "[next-major-version].0.0-next". If either one is incorrect, then fix it using "yarn version". For example:

```
git checkout main
yarn version
...
info Current version: 8.1.0-next
question New version: 9.0.0-next
```

- Then, commit your changes.

```
git commit -a -m "Update version tag for development of next major release"
git push upstream main
```

- You likely should only need to update the version of one branch. In the end, they should look like this:
 - "main" branch should have version="[next-major-version].0.0-next" (e.g. 9.0.0-next)
 - maintenance branch should have version="[current-major-version].1.0-next" (e.g. 8.1.0-next)

Possible Errors you may Encounter

"Could not find model file" error (with language packs)

If you encounter one of these errors when building/packaging DSpace:

FATAL ERROR: "Reason: Could not find the model file '../dspace-xmlui-lang'. for project unknown"

OR

FATAL ERROR: "Reason: Could not find the model file '../dspace-api-lang'. for project unknown"

This is a known bug in Maven. The problem is that you likely have a 'dspace-xmlui-lang' or 'dspace-api-lang' folder at the same level as your [dspace-source] parent folder. Essentially, Maven located them and tried to add them into the build process (which it shouldn't have). The fix is to completely delete the "dspace-xmlui-lang" and "dspace-api-lang" folders, and try to rebuild DSpace.

Advice for future Release Coordinators

With a straight face assure the next Release Coordinator that "Maven is easy" and there is nothing to be afraid of, then put your feet up and open a beer.