# VIVO Release Process

This document is intended to be used and kept up to date by the *VIVO Release Manager*. It details the steps necessary to perform an official release of VIVO.



# Before Release Day

## Release Numbering Convention

As agreed by the Steering Group, VIVO follows the Semantic Versioning guidelines - http://semver.org/

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

## Verify release privileges

To make sure release day goes smoothly, you should ensure that:

1. You have an account with commit access for the vivo-project on github. As a committer, you should already have this level of access.
2. You have an account with edit privileges on the lyrasis.org Confluence wiki.
3. You have an oss.sonatype.org account and have requested to be given permission to publish to the org.vivoweb groupId by adding a comment to the VIVO Sonatype Hosting Ticket
4. You have project configuration privileges on JIRA (you'll see an error here if you don't) : must be added to `Project Settings  Administrators` role

## Update Maven settings.xml

Vitro and VIVO 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">
  <profiles>
    <id>ossrh</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
       <gpg.keyname>YourKeyID</gpg.keyname>
       <gpg.passphrase>YourKeyPassphrase</gpg.passphrase>
       <gpg.defaultKeyring>false</gpg.defaultKeyring>
       <gpg.useagent>true</gpg.useagent>
       <gpg.lockmode>never</gpg.lockmode>
       <gpg.homedir>YourGPGDir</gpg.homedir>
       <gpg.publicKeyring>YourGPGDir/pubring.gpg</gpg.publicKeyring>
       <gpg.secretKeyring>YourGPGDir/secring.gpg</gpg.secretKeyring>
    </properties>
  </profiles>
  <servers>
    <!--Login info for Sonatype SnapShot repository-->
    <server>
      <id>ossrh</id>
      <username>YourSonatypeUsername</username>
      <password>YourSonatypePassword</password>
    </server>
        <server>
      <id>sonatype-nexus-snapshots</id>
      <username>YourSonatypeUsername</username>
      <password>YourSonatypePassword</password>
        </server>
    <server>
      <id>github</id>
      <username>your-github-id</username>
      <password>your-github-pwd</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, and GPG details).

## Ensure you have a trusted code signing key

- create one if you haven't before
- ensure that it's listed within the contributor keys
- 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:
    - 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

## Ensure you have a SSH key setup locally and in GitHub

- Create SSH Key
    - `mkdir -p ~/.ssh`
    - `ssh-keyscan -t rsa github.com >> ~/.ssh/known_hosts`
    - `ssh-keygen -t rsa -C "user.email"`
- Setup on GitHub
    - open the public key with this command `$ cat ~/.ssh/id_rsa.pub` and copy it.
    - Add the *id_rsa.pub* key to SSH keys list on your GitHub profile.

# Creation of release candidates

- The test plan should also be ready prior to code freeze.
- Checkout VIVO-release-publishing project (git checkout https://github.com/vivo-project/VIVO-release-publisher)
- Configure variables for creation of release candidate (defineVariables.sh)

**Variable release number**

```
#!/bin/bash
# RC define common variables Script

export RC_TARGET_DIR=RC
export ORG=vivo-project
export PERSONAL_ACCESS_TOKEN=XXXXXXXX
export BRANCH=main
export RC_VERSION=1.14.0
export RC_VERSION_MINOR=1.14
export RC_NEXT_SNAPSHOT=1.14.1
RC_NUM=5

export Vitro_REPO=Vitro
export Vitro_TAG=vitro
export Vitro_RC_NUM=${RC_NUM}
export VIVO_REPO=VIVO
export VIVO_TAG=vivo
export VIVO_RC_NUM=${RC_NUM}
```

- run prepareReleaseCandidate.sh (./prepareReleaseCandidate.sh)
- announce that Release candidate is ready for testing (Release Testing)
    - Release testing page should contain
        - instructions how to build and run VIVO Release candidate or how to get credentials for an active and public VIVO instance (for instance https://vivo.tib.eu/vivorc)
        - links to google forms where community can provide test results and feedbacks
- analyze results of release candidate testing inside VIVO committers group and classify reported issue to release blockers and non-release blockers
- if there are release blockers, after resolving those issues, the new release candidate should be published, if not, release is ready to be published (see the next section)

# Release Day

- Checkout VIVO-release-publishing project (git checkout https://github.com/vivo-project/VIVO-release-publisher)
- Configure variables for creation of release candidate (defineVariables.sh)

  **Variable release number**

  ```
  #!/bin/bash
  # RC define common variables Script

  export RC_TARGET_DIR=RC
  export ORG=vivo-project
  export PERSONAL_ACCESS_TOKEN=XXXXXXXX
  export BRANCH=main
  export RC_VERSION=1.14.0
  export RC_VERSION_MINOR=1.14
  export RC_NEXT_SNAPSHOT=1.14.1
  RC_NUM=5

  export Vitro_REPO=Vitro
  export Vitro_TAG=vitro
  export Vitro_RC_NUM=${RC_NUM}
  export VIVO_REPO=VIVO
  export VIVO_TAG=vivo
  export VIVO_RC_NUM=${RC_NUM}
  ```

- run publishReleaseVitro.sh (./publishReleaseVitro.sh)
- Go to https://github.com/vivo-project/Vitro/releases/ and check if everything is ok

- Go to https://oss.sonatype.org/index.html (Nexus Repository Manager)

- Click on Log In (Top Right Corner) and use your ossrh id from your settings.xml file
- Click Staging Repositories in left navigation under Build Promotion which will open a new tab
- Search for "vivoweb" in upper right search box (project will not have $REPO in title)
- Select repository and verify that Vitro is present in the Content tab
    - Look for the correct types as well - war, pom, jar, md5, asc, etc.
        - Note there is sometimes a delay on larger files showing in the Repo.
- Click Close, then Refresh, then Release
- After a few moments click into the search under Artifact Search in the left navigation and type "vivoweb"
- A new Search tab will appear with all of the org.vivoweb Release artifacts
- Verify that the new release versions are now listed
    - Note there is sometimes a delay on larger files showing in the Repo.
- This will publish the artifacts to the Sonatype releases repository and start the process of syncing them with Maven Central, which may take several hours. When finished, they'll be available at https://repo1.maven.org/maven2/org/vivoweb.

- Wait until the Vitro release is available in the maven repository (http://repo1.maven.org/maven2/org/vivoweb). This make take several hours or one day.
- run publishReleaseVIVO.sh (./publishReleaseVIVO.sh)
- Go to https://github.com/vivo-project/VIVO/releases/ and check if everything is ok
- Go to https://oss.sonatype.org/index.html (Nexus Repository Manager)
- Click on Log In (Top Right Corner) and use your ossrh id from your settings.xml file
- Click Staging Repositories in left navigation under Build Promotion which will open a new tab
- Search for "vivoweb" in upper right search box (project will not have $REPO in title)
- Select repository and verify that VIVO is present in the Content tab
    - Look for the correct types as well - war, pom, jar, md5, asc, etc.
        - Note there is sometimes a delay on larger files showing in the Repo.
- Click Close, then Refresh, then Release
- After a few moments click into the search under Artifact Search in the left navigation and type "vivoweb"
- A new Search tab will appear with all of the org.vivoweb Release artifacts
- Verify that the new release versions are now listed
    - Note there is sometimes a delay on larger files showing in the Repo.
- This will publish the artifacts to the Sonatype releases repository and start the process of syncing them with Maven Central, which may take several hours. When finished, they'll be available at https://repo1.maven.org/maven2/org/vivoweb.
- Make release announcement and update technical documentation at wiki
- Distribute the message that new release has been published

# Push Release Branch to develop and Maintenance

- Checkout VIVO-release-publishing project (git checkout https://github.com/vivo-project/VIVO-release-publisher)
- Configure variables for creation of release candidate (defineVariables.sh)

**Variable release number**

```
#!/bin/bash
# RC define common variables Script

export RC_TARGET_DIR=RC
export ORG=vivo-project
export PERSONAL_ACCESS_TOKEN=XXXXXXXX
export BRANCH=main
export RC_VERSION=1.14.0
export RC_VERSION_MINOR=1.14
export RC_NEXT_SNAPSHOT=1.14.1
RC_NUM=5

export Vitro_REPO=Vitro
export Vitro_TAG=vitro
export Vitro_RC_NUM=${RC_NUM}
export VIVO_REPO=VIVO
export VIVO_TAG=vivo
export VIVO_RC_NUM=${RC_NUM}
```

- run pushToMaintenance.sh (./pushToMaintenance.sh)

# Announce release

Let Dragan Ivanovic know that the release is complete and can be announced.

# Helpful Tips - Debugging Issues

## Key Issues?

```
#Verify that your GPG key is in your ring
gpg --list-secret-keys


#If the key isn't listed import the private key your previously created
gpg --import name-of-private-key.asc


#List the keys again but in keyid LONG format
gpg --list-secret-keys --keyid-format LONG


#Take the keyid and setup git to use it as your global default.
git config --global user.signingkey YYYYXXXXYYYYXXXX
```