Acquisitions - Release Procedures

The intention of this page is to document the acquisitions-specific release procedures, best practices, lessons learned, and other related information.

Milestone Dates and Terminology

Description of feature freeze, code freeze, etc.

- **Release Cycle**
  - A release cycle consists of two consecutive sprints, typically coinciding with the community-wide "sprint demos"
- **Feature Freeze**
  - Any user stories started after this date will be applied to the next release, not the current one.
  - Usually takes place on the 3rd Friday of the release cycle.
- **Code Freeze**
  - Any changes made after this date will be applied to the next release, not the current one.
  - Release branches will be made as soon after code freeze as possible.
  - Usually takes place on the last Wednesday of the release cycle.
- **Sprint Demo**
  - The community-wide sprint demonstration.
  - Usually takes place on the first Tuesday following a release cycle.

Additional Details TBD.

Release Procedure

High level description of the steps required to perform a release.

Community Release Guides

Acquisitions-specific Steps - Refers to steps in the Community Release Guide linked above, but adds additional steps/sequencing for improved coordination and communication.

ui-modules: check that latest translations are in place (especially if you've changed existing values in en.json).

1. Before you do anything, **communicate with the team**. Let them know during daily standup that you're planning on starting a release.
2. Determine the release and next versions - See https://dev.folio.org/guidelines/contributing/#version-numbers - Discuss with the team if unsure about either.
3. Create a release branch (See Best Practices)
4. Update the `NEWS/CHANGELOG.md` file (See Best Practices)
   a. Make use of JIRA and add any stories that have been completed in this version but are missing from the NEWS/Changelog file.
   b. 

5. Bump interface versions that have changed (minor number if backwards compatible, major number if breaking changes)
   a. **N.B.** Make sure all required interface versions have been released. If not, raise the issue with the PO/Tech Lead.
6. Commit and push your changes
7. Create A PR (See Best Practices) and request approval. If there are mistakes this is the time to identify them, **BEFORE creating a tag**.
8. Once you have at least 2 approvals:
   a. Backend Only: Run the maven command to create the tag and prepare for the next release, then push the pom file changes AND tags
   b. UI Only: Create a tag.
9. If the PR checks pass, **DO NOT merge your PR yet**... Goto Jenkins, find your module, find your tag, and build it.
10. If the tag build succeeded, Goto your github repo, merge the release's PR, click on releases, and draft a new release. Copy/Paste in the section of your NEWS into the release notes and save/publish.

11. Announce your release in the #acquisitions-dev and #releases channels with a link to the release
12. Coordinate with the rest of the team before merging any PRs into master.
    a. If there were interface/dependency changes there's a good chance multiple PRs will need to be merged together to prevent breaking the testing environment.
    b. The team may decide to hold off on merging any PRs until after the sprint demo. You don't want to be the reason why someone's demo isn't working.
13. Create a PR updating platform-complete files as per https://dev.folio.org/guidelines/release-procedures/#platform-edit-steps. Don't forget install-extras.json for edge APIs!
14. Move your JIRA to "In Review" and assign to the Scrum Master.
15. The Scrum Master can safely close the story and update the JIRA releases/versions
16. If the module has API tests:
    a. Create a branch of folio-api-tests
    b. Update all schema references so that the commit hash from the release's submodule is used instead of HEAD/master.
    c. Update all mock data references so that the commit hash from the release's submodule is used instead of HEAD/master.
    d. Create and push a tag of your branch. Please follow the naming convention: `<module name>-<release version>`, e.g. **`mod-orders-3.0.0`**

Module Dependencies and Release Order

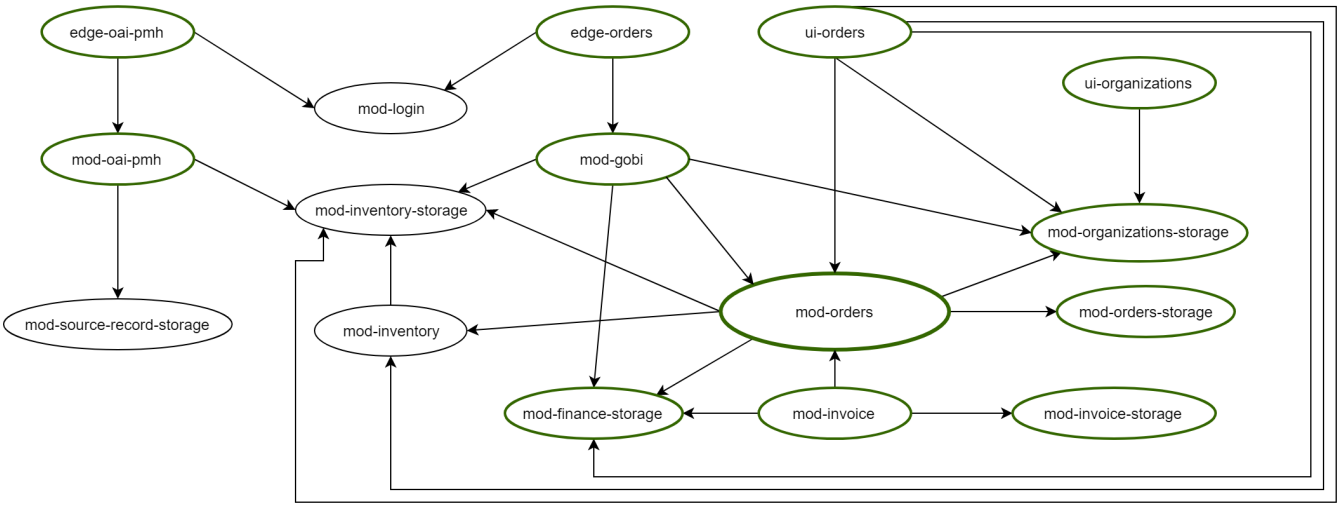
In order to satisfy dependencies at time of release, modules need to be released in a particular order.

Generally speaking the precedence looks like:

1. Storage modules
2. Business logic modules
3. UI modules

In some cases there are dependencies between business logic modules which require additional ordering.

TODO: add a dependency graph here showing the acquisitions modules and the interfaces they depend on.



Best Practices

General best practices - How to structure your NEWS file, what to name your release branches, tags, etc.

- **NEWS**
  - Latest release at the top of the file
  - Includes a link to the full changelog (github compare of the previous version tag to this tag)
    - `E.g. https://github.com/folio-org/mod-orders/compare/v2.0.1...v3.0.0`
  - Links to JIRA stories/bugs in descending order (highest to lowest)
  - You may want to separate stories and bugs into separate lists

Example

```
## 4.0.0 - Unreleased

## 3.0.0 - Released

The primary focus of this release was to implement backend logic necessary for ui-orders to manage
purchase orders and purchase order lines, integration between orders and inventory apps and
preparation for the receiving flows.

[Full Changelog](https://github.com/folio-org/mod-orders/compare/v2.0.1...v3.0.0)

### Stories
 * [MODORDERS-154](https://issues.folio.org/browse/MODORDERS-154) - PO Line's `location` property
is changed to `locations` i.e. from single object to array of objects
 * [MODORDERS-149](https://issues.folio.org/browse/MODORDERS-149) - Add PO Line's identifier to
item record in the inventory
 * [MODORDERS-148](https://issues.folio.org/browse/MODORDERS-148) - Populate PO `dateOrdered`
field field when Order is opened
 * [MODORDERS-146](https://issues.folio.org/browse/MODORDERS-146) - Return application/json (error.
json schema) for all errors
 * [MODORDERS-142](https://issues.folio.org/browse/MODORDERS-142) - Implemented `GET /orders/order-
lines` endpoint
 * [MODORDERS-134](https://issues.folio.org/browse/MODORDERS-134) - Assign PO Line's id to its sub-
objects
 * [MODORDERS-129](https://issues.folio.org/browse/MODORDERS-129) - PO Lines: share status of
corresponding PO
 * [MODORDERS-126](https://issues.folio.org/browse/MODORDERS-126) - Implemented `GET /orders
/composite-orders` endpoint
 * [MODORDERS-124](https://issues.folio.org/browse/MODORDERS-124) - Redefined existing order/lines
endpoints
 * [MODORDERS-121](https://issues.folio.org/browse/MODORDERS-121) - Create Instance Record in
inventory when Order's status is changed to `Open`
 * [MODORDERS-117](https://issues.folio.org/browse/MODORDERS-117) - Business Logic: handle `Create
Item` flag for e-Resources
 * [MODORDERS-105](https://issues.folio.org/browse/MODORDERS-105) - Implemented `GET /orders
/receiving-history` endpoint
 * [MODORDERS-100](https://issues.folio.org/browse/MODORDERS-100) - Create Piece Records in Orders
Storage for items quantity ordered on Order Placement
 * [MODORDERS-99](https://issues.folio.org/browse/MODORDERS-99) - Purchase Order Limit: Set system
default to 1
 * [MODORDERS-96](https://issues.folio.org/browse/MODORDERS-96) - Supporting PO number prefix and
suffix
 * [MODORDERS-93](https://issues.folio.org/browse/MODORDERS-93) - Assign system generated PO Line
number when creating a new PO line
 * [MODORDERS-87](https://issues.folio.org/browse/MODORDERS-87) - Implemented `GET /orders/po-
number` endpoint
 * [MODORDERS-72](https://issues.folio.org/browse/MODORDERS-72) - Define receiving endpoints: `
/orders/receive`, `/orders/check-in` and `/orders/receiving-history`
 * [MODORDERS-67](https://issues.folio.org/browse/MODORDERS-67) - Create Item Record in inventory
for physical/electronic items quantity
 * [MODORDERS-66](https://issues.folio.org/browse/MODORDERS-66) - Create Holding Record in
inventory for titles ordered that are not currently represented in inventory by a Holding

### Bug Fixes
 * [MODORDERS-153](https://issues.folio.org/browse/MODORDERS-153) - PO Line's id is absent from
sub-objects in response when creating new PO Line
 * [MODORDERS-145](https://issues.folio.org/browse/MODORDERS-145) - Unable to create new Purchase
Order with PO Line
```

- **Release branches** should be named as follows: `tmp-release-#.#.#`
  - This naming convention will ensure consistency across modules and makes it easily identifiable.
  - Example: `tmp-release-1.2.1`
- **Release PRs** should be named as follows: `<JIRA number> - Release #.#.#`
  - This naming convention will ensure that the PR is automatically linked to from the JIRA story
  - Example: `MODORDERS-131 - Release 2.0.0`
- TBD

Lessons Learned

A place to document pitfalls, gotchas, lessons learned in order to help others from repeating the same mistakes etc.

- Creating a tag before allowing others to review your changes will make it more difficult to make changes if any are required. (May only apply to backend modules)
    - Removing and re-tagging is known to cause problems with the jenkins. When building the release, jenkins sometimes won't be able to find the tag even though it exists.
    - Scrapping the release and moving to the next version creates additional overhead and makes the release notes more confusing.
- API tests: separate branch needs to be created and update any variables to point to release tags
- TBD

Open Issues and Questions

A place to list open issues for tracking purposes, or ask questions for gathering feedback/comments.

- TBD