Code Contribution Guidelines

Contributing Code to DSpace Software

This page provides specific guidelines on how to contribute your code to DSpace, and details how the code review/acceptance process works. Developers - See also the Developer Guidelines and Tools for hints/tips on using popular IDEs to develop with DSpace.

Contribution Quick Checklist

This page has a lot of great information on it. But, if you just need the high level overview, here it is. NOTE: the checkboxes below are here for "decorative" purposes, and aren't really intended to be used. If you'd like, and you find it helpful, you may print this page and then check off each step as you proceed with your contribution. Happy coding, and please ask for help if you find you need it!

- **Create a ticket** in the DSpace Issue Tracker (describe your contribution, how to use it, and perhaps some use cases). This helps ensure your upcoming work is "known" to other developers, and we can let you know if someone else is working on something similar.
- **Make sure your code adheres to our Code Style Guide** (only required for DSpace 7.x or above)
- **Write unit/integration tests per our Code Testing Guide** (only required for DSpace 7.x or above)
- **Submit your code via GitHub**. Please submit a GitHub Pull Request (see GitHub's "About Pull Requests"), which references your newly created ticket by number.
  - If your code involves changes to the v7 REST API, please also include a REST Contract Pull Request which documents the endpoint(s) that require changes.
  - Ideally submit your code or ideas **early on** before it turns into a massive project. Larger code changes take a very long time to understand, review and test. Starting the discussion early (or breaking the changes up into several smaller changes) can make it much easier to get your code accepted.
- **Review your own code**. Does it follow our Contribution Checklist? Does it need Documentation? If you are using any third party tools/APIs, do they all have an acceptable Open Source License (see Licensing of Contributions)? The Committers will also be reviewing these aspects of your code, but if you can catch these gaps or issues up front it can speed up the process of correcting them.
- **Respond to feedback**. If the Committers ask questions or make suggestions for changes, please try to be responsive. The Committers are all volunteers and are trying to help as best we can, but the process moves more quickly if you can try to be responsive as well.
- **Help rework/update code as needed**. If suggestions for changes are made, if you can rework the code, it speeds up the process. If you submitted your code as a Pull Request, you can just quickly add changes/updates to the branch linked to from your Pull Request.
- **Ask questions**. If there is a long delay in the Committers responding, or if you aren't sure of the status of your contribution, please ask. We'd be glad to explain whether the delay is just because we are all busy, or if there's something else we are waiting on.
- **Pay attention to release deadlines.** As the next DSpace release approaches, the Committers will announce a "Contribution Deadline" for the upcoming release (usually these are found in Developer meeting notes). In order to keep releases on-time, the Committers must set a date after which they can no longer accept new feature contributions. Although you may add code contributions year round, they will only be considered for a specific release if they are contributed before that release's contribution deadline.

Overview of Code Approval Process – How to get your Code into DSpace!

0. Share Early, Share Often!

The overriding mantra is share early, share often. Here are a few things to consider **before** you begin working on your code:

- **For Larger Initiatives/Codebases:** *If you are building out a much larger project, we highly recommend notifying the community of the work early on via an email to dspace-devel@googlegroups.com (or via one of the weekly Developer Meetings or in #dev on Slack).* This has several benefits:
  - Ensures you achieve your goals in a way that is consistent with the DSpace architecture and plans of the rest of the community.
  - Minimizes the chances of a scenario where you have invested a large amount of time and effort into a body of code that does not fit in with the DSpace architecture or the consensus of the community.
  - This can help find collaborators or get early feedback.
- **Develop incrementally**; try and implement and contribute a basic form of your feature as soon as possible, rather than aiming to implement a complete and 'polished' solution. This will help ensure you're on the right track with regards to the rest of the DSpace community and platform. The sooner your code is part of the core code base, the less time you will have to spend 'chasing' the main code base, i.e. keeping your changes up-to-date with that core code base.
- Obtain the DSpace code using GitHub. This will make code contribution much easier, as we require a GitHub Pull Request for contributions.
- Read Code Contribution Guidelines (this page), Code Style Guide and Code Testing Guide to ensure you are following DSpace conventions. This will ensure your code is more likely to be immediately accepted as part of out-of-the-box DSpace.

- Ensure that any third-party tools/libraries that you plan to utilize are released under compatible open source licenses. See the Licensing of Contributions section below.

1. Make your code available (in GitHub) and create a ticket in our Issue Tracker

Start by creating a new ticket in our GitHub Issues. This ensures that the DSpace Developers are notified of your contribution, and acts as a place for us to comment on the work or make suggestions for improvements.

Once your code is ready, you must make your code available to other DSpace Developers for review. The easiest way for us to review your code is by putting your code into a GitHub Pull Request.

For DSpace, we follow GitHub's "Fork and Pull model" of collaborative development. This means you should fork our DSpace GitHub repository, adding your changes to a branch in your forked repository, and then create a pull request from your fork. In your Pull request, link back to your created issue by saying that the PR "Fixes #issue-id" (e.g. "Fixes #12345") as this will link your PR with the issue ticket.

Code Standards

Code contributions that meet certain standards are much more likely to be accepted immediately. For a list of our current standards, please read through the Code Contribution Standards section below.

Keep in Mind the "Feature Contribution Deadline" for the Next Release

When the next release of DSpace is getting close, the Committers will set a "Feature Contribution Deadline" date, after which no new feature submissions will be accepted for that release. The reason for this is that the Committers need time to review & stabilize the current code before the next release can be completed. Make sure to check the Next Release Status page for details on when the next "Feature Contribution Deadline" is. Please note that bug fixes are still accepted after the "Code Contribution Deadline", as they will help to stabilize the upcoming release.

2. Code Review Process

Once the code is made available in a Pull Request, we will work to find it reviewers/testers. By default, every Pull Request requires *at least* two separate reviewers (or testers), ideally from two different institutions. Some small PRs (usually 200 lines or less) might be flagged as "1 Approval", meaning only one person needs to review that work. During review/testing, other developers may contact you via GitHub to discuss any feedback they have, and whether or not there would need to be some general changes before we could accept it. Some patches/features are readily accepted (because they are stable and look good), others may require more work (if there are concerns or issues that others notice).

Code Review Timeframe

The timeframe of a code review will vary, based on how much time the core developers have. Smaller changes may be reviewed within days, while larger changes/features may take many weeks to do a full review. All Developers/Committers are volunteers and only have a small amount of time to provide to the project in a given week. We will make every effort to get back to you with feedback within a few weeks. But, if you haven't heard anything, feel free to ask!

What are we reviewing for?

When we review your code, we are mostly ensuring it generally follows our Code Contribution Standards. However, there are a few other things we generally check for:

- The code is well commented (e.g. has JavaDocs or TypeDocs)
- The code follows our Code Style Guide (*required for DSpace 7.x and above*)
- The code provides Unit and/or Integration Tests (see Code Testing Guide) *(required for DSpace 7.x and above)*
- The code is stable and has no stability or security concerns
- The code is properly using existing APIs, etc.
- The code is not too specific to one institution's local policies or workflows. (I.e. we will review the code to ensure it looks to be generally useful to most institutions, or configurable enough such that others can change it to match their own local policies/workflows)
- Any third-party tools/libraries used by your code have compatible open source licenses. See Licensing of Contributions

3. Reworking Code (if necessary) & Next Steps

After the code review & feedback, interested developers/Committers may help you to rework the code (if needed). They'll also provide you with next steps on getting the code into DSpace. If it can be accepted immediately, it will be. If not, we'll try to help figure out the best route forward.

How you can help speed up the process

As our developers/Committers are all volunteers, they don't always have the time to rework code changes for you. If you want your code change accepted in a timely manner, please offer to make the changes yourself (otherwise your patch suggestion may wait in a "holding queue" until someone has enough time to work on any necessary fixes).

Communicate, Communicate, Communicate

If you are unsure of next steps, please let us know by adding a comment to your issue/PR in GitHub. Communication is absolutely necessary to ensure that we can help you rework anything that needs reworking. If we don't hear from you, we'll assume you are hard at work. So, if you've run into issues, please let us know! If, locally, you don't have the time or expertise to do the rework that is necessary, also let us know. We can try to locate a community developer to help out, and/or ask both the Committers Team and the DSpace Community Advisory Team if they know of any interested developers with time to spare.

4. Acceptance!

Once your code is accepted, it will be released in the next version of DSpace software! Your name will appear in the Release Notes as a contributor to that release!

Code Contribution Standards

Code contributions that meet the following standards are much more likely to be accepted. If you don't understand any of these standards, please contact us – we'll be glad to explain or help.

Contribution Checklist

When you contribute to DSpace, please be sure that your submission adheres to the points in this checklist. The DSpace Committers need you to do this to keep quality of the DSpace code high and their work manageable.

1. Any changes *must* be compliant with the supported version of Java (e.g. for DSpace 7, Java 11 compliance is required)
2. Your code *must* adhere to our Java Code Style Guide. Most major IDEs can easily import our Checkstyle configurations to ensure alignment with this code style.
    a. Your code should be well commented with Javadoc (required for all classes, public methods and larger private/protected methods).
3. Your code *must* provide unit/integration tests for new features, bug fixes or improvements per our Code Testing Guide.
4. If your contribution adds REST API endpoints or changes existing endpoint(s) behavior, you *must* submit a corresponding REST Contract Pull Request which documents the require changes.
5. If your contribution adds new third-party tools or libraries, they must adhere to licensing requirements to be included. Refer to the Licensing of Contributions below
6. User interface changes must use i18n keys to allow for easy translation to other languages (see the DSpace 7 Translation - Internationalization (i18n) - Localization (l10n) guide). At a minimum, please provide English text so that others may translate it.
7. User interface changes should align with our User Interface Design Principles & Accessibility documentation. Primarily, this is a requirement for W3C WCAG 2.1 Conformance Level of "Double-A".
8. Your code must come with Documentation. Minimally, technical documentation must be part of the system docs – see Documentation Contributions below. Ideally, we'd also like User/Usage Documentation.
9. Ideally, new features should be configurable (i.e. generalized so as to not be specific to one institution's needs/use cases). Any new configurations should have sane defaults which can be overridden (as needed) in a site's local.cfg file.
10. Add appropriate/useful logging to your code (ERROR, WARN, INFO, DEBUG, etc).
    a. Provide informative log statements and/or the entire Throwable exception. For example, "log.error('My custom error message', e)" is more useful than "log.error(e.getMessage(), e)"
11. Where reasonable, retain feature/functionality backwards compatibility. This does NOT mean you have to create PRs for older versions of DSpace. But, it does mean that you do your best to keep in mind the behavior of features in older DSpace releases, and attempt to keep similar behaviors where reasonable. If there are questions/concerns about this, let us know. There are always exceptions.
12. No database schema changes unless absolutely necessary to support a new feature. See Database schema changes below.
13. If your code makes changes to the database schema *or* content, and you are patching more than one branch (for example, dspace-6_x and master), see Patching multiple branches below.

If there are questions/concerns about any of these guidelines, let us know on the 'dspace-devel' list. We are willing to make exceptions in some areas, if exceptions are necessary.

Attempt to Follow all Guidelines

⚠

Omission of one or more of these items is likely to result in the a request for further work. See the Overview of Code Approval Process above, for more information.

Licensing of Contributions

Any third-party libraries (e.g. JARs / Maven Dependencies) required to compile or run DSpace must be included. **The license of any required jar /dependency MUST be compatible with BSD.** It must not prevent any commercial use of DSpace, nor have any impact on the rest of the code by its inclusion. It is not acceptable to require additional downloads of JARs/dependencies to make DSpace compile or function.

Non-Java third-party web frameworks or tools (e.g. XSLT, CSS, Images) should follow these same licensing guidelines.

Examples of acceptable licenses:

- Apache License 2.0
- BSD
- Common Development and Distribution License (CDDL)
- Common Public License (CPL)
- GNU Library or "Lesser" General Public License (LGPL)
- MIT / X11 License
- Mozilla Public License
- Additional examples may be found in our LICENSES_THIRD_PARTY file in the source code. This file lists the licenses of all third-party libraries used by DSpace.

Examples of unacceptable licenses:

- GNU General Public License (GPL)
- GNU Affero General Public License (AGPL)
- European Union Public Licence (EUPL)
    - Similar to GPL, this license is "share alike" / "strong copyleft" and may require usage of the same license for redistribution or derivatives. For more information, see the EUPL FAQ (specifically the questions "What about compatibility issues?" and "Are there limitations to the use of the software?")
- Any license which requires "same license" (i.e. strong copyleft) as detailed at http://choosealicense.com/licenses/
- Any license which limits commercial use/redistribution of binary code

Why is GPL (and similar) unacceptable?

⚠

DuraSpace feels it is important for commercial entities and service providers to be able to customize the entire codebase and redistribute/repackage/sell it in a binary form. GPL licenses prevent this, as noted in the following FAQ questions:

- Can I release a modified version of a GPL-covered program in binary form only?
- If I distribute GPL'd software for a fee, am I required to also make it available to the public without a charge?

In addition, the Apache Software Foundation has a good explanation of why they are also forced to avoid GPL-based (copyleft) licenses because of its one-way compatibility with Apache License 2.0:

> "This licensing incompatibility applies *only* when some Apache project software becomes a derivative work of some GPLv3 software, because then the Apache software would have to be distributed under GPLv3.
>
> We avoid GPLv3 software because merely linking to it is considered by the GPLv3 authors to create a derivative work. We want to honor their license. Unless GPLv3 licensors relax this interpretation of their own license regarding linking, our licensing philosophies are fundamentally incompatible. This is an identical issue for both GPLv2 and GPLv3."

While DSpace is released under BSD licensing, the same issues exist between BSD licenses and GPL-based licenses.

JDBC drivers for databases are an exception since:

- They must correspond to the database version and not the DSpace version.
- They are not required for DSpace to compile and run; a variety of databases, including open source databases, may be used.

Database schema changes

Database schema changes will be done only on major revisions to the source; this is when the version number takes the form x.0 (e.g. 2.0). When making patches which cause schema changes, it is necessary to update all of the relevant SQL/migration files with your sequences, tables, views etc.

- For Database migrations/management, we use FlywayDB
  - The migration scripts are available in the `dspace-api` source code under the `org.dspace.storage.rdbms.sqlmigration` package: https://github.com/DSpace/DSpace/tree/master/dspace-api/src/main/resources/org/dspace/storage/rdbms/sqlmigration
  - Each new migration script should be named "`V[version]_[date]__[description].sql`", where `[version]` is the DSpace version supporting this change, `[date]` is the date of the change, and `[description]` includes the associated ticket number and brief description of the migration. For example: "`V5.0_2014.09.26__DS-1582_Metadata_For_All_Objects.sql`".
  - **NOTE:** Whenever possible, please **avoid** modifying migration scripts from a prior DSpace release. Flyway does not compare modified scripts against the current database structure, so it will not modify tables based on modifications to existing migration scripts. Therefore, modifications to existing scripts may result in some (or all) DSpace users having to *manually* run those database changes during their next upgrade.
    - Instead, where possible, consider creating a migration script (in SQL or Java) that will run just *before (or after)* the script you are looking to modify.
- If your database migration adds new sequences, then you should also be sure to update the update-sequences script at:
  - `[dspace-src]/dspace/etc/[db-type]/update-sequences.sql`
  - At this time, this updated-sequences script is maintained outside of the database migrations as it is useful to run manually after large restorations, etc.

Patching multiple branches

When you patch the same issue in multiple branches, database changes require special attention. This advice applies to both schema changes and content changes.

- FlywayDB migrations are **cumulative**. You should depend on changes that you made in the earliest affected branch to be available in later branches, and only do further migrations in later branches if additional changes are needed.
- When patching a branch earlier than 5_x, you will need to provide an SQL script or a tool to be run manually. 5_x and later patches should still depend on these manual updates, and you should document the need to run them before allowing automatic migrations to run.
- If the schema was already changed between branches, and those changes affect the same tables that you are updating, depend also on the existing upgrade process to make those changes for you. For example: if you make database changes to table T in branch X, and the upgrade from X to Y changes the schema for T, you don't need to rewrite your changes for branch Y because the upgrade already took care of that difference.

Documentation Contributions

All new features require documentation before they will be accepted. You may send us code before documentation is completed, but we will be unable to accept that code into DSpace until it is properly documented. Bug fixes may not require documentation, unless they somehow make a modification which changes how DSpace functions.

All documentation is built in a special section of the Wiki at DSpace Documentation. Therefore, the best way to send us Documentation is to actually create a new page(s) in this DSpace Wiki. You should link these Wiki page(s) to your issue in our DSpace Issue Tracker. We'll move them over into the official DSpace Documentation area once you code has been accepted.