ArchivesSpace Contributors' Guide

May 15, 2019

If you are planning on contributing to the ArchivesSpace program, please read this document first.

**Table of Contents**

[General Overview 2](#_Toc8817935)

[Participating in the community 2](#_Toc8817936)

[Contributors 3](#_Toc8817937)

[Code approval 3](#_Toc8817938)

[Core Committers Group calls 5](#_Toc8817939)

[Code Contribution Guidelines 5](#_Toc8817940)

[High Level Guidelines 5](#_Toc8817941)

[Committing Guidelines and Flow 6](#_Toc8817942)

[When Submitting a Pull Request 7](#_Toc8817943)

[Bugs/Issues 8](#_Toc8817944)

[How to report a bug 8](#_Toc8817945)

[How to request a new feature 8](#_Toc8817946)

[Issue management 8](#_Toc8817947)

[Example Issue Workflow 8](#_Toc8817948)

[Acknowledgements 10](#_Toc8817949)

# General Overview

## Participating in the community

ArchivesSpace is an open­source program and is released under the Educational Community License, version 2.0 (http://opensource.org/licenses/ecl2.php).

There are many ways to contribute to ArchivesSpace, including updating and writing documentation, writing code, and participating in testing. If you'd like to contribute to ArchivesSpace, then look at our wiki pages: <https://archivesspace.atlassian.net/wiki/spaces/ADC> and our development catalog: <https://archivesspace.atlassian.net/issues/?filter=13630>.

The technical documentation can be found on the ArchivesSpace website at <http://archivesspace.org/application/technical-documentation/> and http://archivesspace.github.io/archivesspace/.

Get a copy of the latest master branch from <https://github.com/archivesspace/archivesspace>.

Join the "users group" and "core committers" mailing lists. The users group list, Archivesspace\_Users\_Group@lyralists.lyrasis.org, is where almost all user discussion takes place. The core committers group list, Archivesspace\_Core\_Group@lyralists.lyrasis.org, is for the core committers. All development questions should go there, though you might want to check the list archives first.

## Contributors

### Code approval

1. 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:

* Please be sure to share your plans with the ArchivesSpace community on the ArchivesSpace users list or via one of the monthly Core Committers' Meetings before embarking on any sizable development effort. This will ensure you achieve your goals in a way that is consistent with the ArchivesSpace architecture and plans of the rest of the community. It will minimize 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 ArchivesSpace architecture or the consensus of the community.
* Develop incrementally; try to 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 ArchivesSpace 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 or keeping your changes up-to-date with that core code base.
* Obtain the ArchivesSpace code using GitHub. This will make code management much easier. You can either [clone](https://help.github.com/articles/cloning-a-repository/) or [fork](https://help.github.com/articles/fork-a-repo/) the [ArchivesSpace Github repository](https://github.com/archivesspace/archivesspace).
* Read the [Code Contribution Guidelines](#_44sinio) in this document to ensure you are following ArchivesSpace conventions. This will ensure your code is more likely to be immediately accepted as part of out-of-the-box ArchivesSpace.
* Ensure that any third-party tools/libraries that you plan to utilize are released under compatible open source licenses.
* **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 both the Google Group and Users Group lists. This can help find collaborators or get early feedback. We also recommend you develop your project in GitHub, as it provides easier ways to review/collaborate with other developers.
1. Make your code available (preferably in GitHub) and create a ticket in our JIRA repository.

Once your code is ready, you must make your code available to the ArchivesSpace Core Committers Group for review. The easiest way for us to review your code is by putting your code into GitHub and then submitting a "Pull Request" to our GitHub repository.

You must also create a new ticket in our [ArchivesSpace JIRA Repository](https://archivesspace.atlassian.net/secure/Dashboard.jspa). This ensures that the ArchivesSpace Program Team and Core Committers are notified of your contribution, and JIRA provides a place for us to comment on the work or make suggestions for improvements.

1. Code Review Process

Once the code is made available, the Core Committers Group will take time to review the work and provide feedback/comments. Code must be reviewed by at least one other core committer from another organization before it can be merged.

Usually, one (or more) core committers who are interested in this work will contact you and 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 Core Committers notice).

The timeframe of a code review will vary, based on how much time the Core Committers have. Smaller changes may be reviewed within days, while larger changes/features may take many weeks to do a full review. All Core Committers are volunteers and only have a small amount of time to provide to the program 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!

1. Reworking Code (if necessary) & Next Steps

After the code review & feedback, interested Core Committers may help you to rework the code (if needed). They'll also provide you with next steps on getting the code into ArchivesSpace. 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 Core 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).

If you are unsure of next steps, please let us know by adding a comment to your issue in JIRA. 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 Core Committers Group and the ArchivesSpace Program Team if they know of any interested developers with time to spare.

1. Acceptance

Once your code is accepted, it will be released in the next version of ArchivesSpace software. It is time to celebrate, as your name will be added to the prestigious list of ArchivesSpace Contributors.

# Core Committers Group calls

ArchivesSpace core committers calls are monthly meetings of the core committers of ArchivesSpace, who discuss modifications and innovations in the code, and review new and open tickets. Calls are open to all interested community members. Information about these open calls can be found here: https://archivesspace.atlassian.net/wiki/spaces/ADC/pages/312836097/Call+in+information.

# Code Contribution Guidelines

## High Level Guidelines

1. Be consistent
2. Don't rewrite existing code just to follow these guidelines
3. Refactoring should be kept separate from functional updates, to make it easier to review.
4. Smaller updates are better and much easier to review. It's OK to submit a chain of small updates that build off each other.
5. Don't violate a guideline without good reason. If you do, expect to be asked to defend the reasoning.
6. A reason is good when you can convince another core committer and the violated guideline should be reviewed if it needs to be changed.

## Committing Guidelines and Flow

1. Make sure that you have completed and submitted an [Individual Contributor License Agreement](https://github.com/archivesspace/archivesspace/blob/master/contribution_files/ArchivesSpace_Individual_Contributor_License_Agreement_ICLA.pdf) and that your institution has completed and submitted a [Corporate Contributor License Agreement](https://github.com/archivesspace/archivesspace/blob/master/contribution_files/ArchivesSpace_Corporate_Contributor_License_Agreement.pdf).
2. Find or create a bug report or feature request ticket in JIRA or GitHub (to make sure you’re not duplicating work and document the intent of your contribution). It helps to explain both the existing behavior and the desired behavior that your change will implement. Assign the ticket to yourself prior to starting the work so that others in the community know that you have taken responsibility for implementing the solution.
3. Set up a GitHub account if you have not done so before.
4. Fork the [ArchivesSpace repository](http://github.com/archivesspace/archivesspace)on GitHub. The advantage of forking existing codebases is that you can do so without asking anyone and get started quickly. The disadvantage is that there's a risk of writing code that only lives in your fork, which -- if you wind up running this in production, or pointing other production code at your fork -- is a surefire way to make your work difficult to upgrade.
5. Create a feature branch. Do not work directly on the master branch, which is a protected branch. It is helpful to start the name of the branch with the JIRA ticket or Github issue number.
6. As you start working on your contribution, feel free to make commits as often as makes sense to you since no one will see these until you push your code up to GitHub; there will be an opportunity later to squash your commits and make sure your commit message is written succinctly and with sufficient detail. (Your commit message will be pulled in as the default description of your PR, so investing some time in writing a [high-quality commit message](http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html) will help you, and code reviewers, later.)
7. We also recommend writing tests early and running them often, making small commits as you go. This gives you frequent checkpoints to make sure you do not dig yourself into a deep hole or become bogged down towards the end of working on your changes.
8. Make changes in your fork. We advise contributors to follow these guidelines to expedite the contribution review process.
	1. Include the JIRA ticket or Github Issue number in your commit message.
	2. Follow established style guidelines
		1. Rails: <https://github.com/bbatsov/rails-style-guide>
		2. JRuby: <https://github.com/jruby/jruby/wiki/JRubyStyleGuide>
		3. RSpec: <http://betterspecs.org/>
	3. Include unit tests sufficient to cover the feature(s) you add or bug(s) you fix, and make sure the test suite passes when you run it locally.
	4. Make sure the code is
		1. stable and has no stability or security concerns
		2. properly using existing APIs, etc.
		3. not too specific to one institution's local policies or workflows. 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.
	5. Any third-party tools/libraries used by your code have compatible open source licenses.
	6. If you’re adding a feature or otherwise changing documented behavior, modify the documentation to reflect your changes.
9. Once your work is done, squash the commits in your branch — see [One Commit per Pull Request](http://ndlib.github.io/practices/one-commit-per-pull-request/) for some guidelines — and rebase it to the latest in the upstream master branch. We appreciate succinct but explanatory commit messages – see [A Note About Git Commit Messages](http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html).
10. Push your updated branch to your fork.
11. Create a Pull Request on GitHub. We have found that it is best when pull requests are small; targeted at a specific issue; and pushed up quickly for review.
12. Respond to feedback as the community reviews your contribution.

## When Submitting a Pull Request

When you have code ready to be reviewed, and it meets the Code Contribution Guidelines, open a pull request to ask for it to be merged into the codebase.

To help make the review go smoothly, here are some general guidelines:

* Your pull request should address a single issue.
	+ It's better to split large or complicated PRs into discrete steps if possible. This makes review more manageable and reduces the risk of conflicts with other changes.
* Give your pull request a brief title, and use the description to provide key information:
	+ Provide a list of the key changes
	+ If your PR addresses an existing GitHub ticket, link to it with "Fixes #123". This will make it easy to refer back to the original ticket and automatically close it when the PR is merged.
	+ If your PR addresses an existing JIRA ticket, include a link in the description to the corresponding JIRA ticket.
	+ If you've discussed the issue, or just want to alert someone to your PR, tag them by including their @username.
	+ Link to relevant resources, such as Wiki pages, mailing list threads, specifications, or other tickets. This makes it easier to understand the full context of your PR.
* Please be patient. Your PR may not be reviewed right away, since the people doing code review are often busy, and may be traveling, in a different time zone, or otherwise not available to review your code immediately.
	+ It is fine to ask someone to review, either by tagging them in a comment or asking on the core committers' mailing list, especially if it is blocking other work.
* Respond to code review comments, with discussion where it's appropriate or by pushing additional commits to the branch. They will be added to the same PR automatically.
* If another PR is merged and conflicts with your changes, you may need to rebase your PR.
	+ See [GitHub's rebasing documentation](https://help.github.com/articles/about-git-rebase/) and [One Commit Per Pull Request](http://ndlib.github.io/practices/one-commit-per-pull-request/) for more information on rebasing.

# Bugs/Issues

ArchivesSpace isn't perfect software. It contains bugs, lacks features, and has room for improvement like any other piece of software. Like most software programs, the ArchivesSpace program makes use of an issue tracking tool to manage known outstanding issues with the software. But perhaps unlike most software programs, we try to keep our issue tracker relatively free of debris. It's not that we don't want to hear about ArchivesSpace's problems — after all, we can't fix what we don't know is broken. It's just that we've found a mismanaged issue tracker to be more of a hindrance than a help.

The following are the policies that we ask folks to abide by when reporting problems or requested enhancements to ArchivesSpace.

## How to report a bug

The process is described in the ArchivesSpace wiki: [https://archivesspace.atlassian.net/wiki/display/ADC/How+to+Report+a+Bug](https://archivesspace.atlassian.net/wiki/display/ADC/How%2Bto%2BReport%2Ba%2BBug).

## How to request a new feature

The process is described in the ArchivesSpace wiki: [https://archivesspace.atlassian.net/wiki/display/ADC/How+to+Request+a+New+Feature](https://archivesspace.atlassian.net/wiki/display/ADC/How%2Bto%2BRequest%2Ba%2BNew%2BFeature).

## Issue management

Using JIRA sprints to manage issues is an important aspect of how the ArchivesSpace Program Team and core committers organize their efforts and communicate with each other and with the ArchivesSpace community at large. ArchivesSpace's sprints tend to be named to reflect release version numbers and variations thereof. The statuses of the JIRA issues are used to designate specific steps in the issues management workflow, so it's important to understand their meanings.

## Example Issue Workflow

A member of the ArchivesSpace community wants to request a new feature or report a bug so they open a JIRA ticket (status="Newly Added"). The Program Team reviews the "Newly Added" tickets for completeness and changes the status to "Awaiting Support" for support tickets, "Awaiting Prioritization" for code development tickets, "Awaiting More Information" for tickets that need more description and examples, or "In Review" for tickets that have been looked at by the Program Team but the next step has not been determined yet.

For support tickets, the Program Team will follow-up with the reporter and determine how to satisfy the need. Once the support is finished, the status is changed to "Closed" after the ASpace Version field in the ticket is updated with "No Version".

For code development tickets, the prioritization Subteam reviews the tickets for duplicates. A comment is added to the duplicate ticket with a reference to the ticket it duplicates, the status of duplicate tickets is changed to "Closed-Duplicate", and the ticket exits the workflow. For tickets that the prioritization Subteam determines we will not do, the status is changed to "Closed-Will Not Do" and the ticket exits the workflow. For tickets that the prioritization Subteam assign a priority, the status is changed to "Ready for Implementation". Next, the Program Team assigns the ticket to a particular sprint cycle, software version, or the backlog and the status is changed to "Not Yet Started". The ticket is then assigned to a developer who sets the status to "Started".

The ticket goes through another review by the developer who determines one of the following:

1. there is more information needed so the developer changes the status to "Waiting for Questions to be Answered" after documenting questions in the ticket using comments

2. it doesn't need code implemented so the developer changes the status to "Awaiting Support" (NOTE: this won't happen often but an example could be that the Program Team thinks the ticket needs an indexing code change but it is determined that a re-index is required instead.)

3. it is ready for implementation so the developer completes the implementation and changes the status to "Ready for Testing"

Once the code for the ticket has been tested, if accepted, the status is changed to "Ready for Inclusion in Release Candidate" and, if rejected because there is more work needed, the status is changed to "Rejected". When a developer is ready to start work on the ticket again, the developer changes the status back to "Started" and begins work on the ticket again. Once included in the release candidate, the status is changed to "Closed" once the ASpace Version field in the ticket is updated with the appropriate release version.

# Acknowledgements

This document is based upon the original [ArchivesSpace re-org and committer groups proposal](https://archivesspace.atlassian.net/wiki/display/ADC/ArchivesSpace%2Bre-org%2Band%2Bcommitter%2Bgroups%2Bproposal), the [Subversion Community Guide](https://subversion.apache.org/docs/community-guide/), and documentation from several open source projects – [Islandora](https://github.com/Islandora/islandora/wiki), [DSpace](https://wiki.duraspace.org/display/DSPACE/Contribution), and [Hydra](https://wiki.duraspace.org/display/hydra/Hydra%2BCommunity%2BFramework).