A good-enough workflow for software citation

Stephan Druskat, Jurriaan Spaaks, Tom Morrell, Alexandros Loannidis, Caifan Du, Dan Katz (not physically in the group but reading and commenting frequently and thoroughly)

Scope: This document describes a better workflow for integrating software citation metadata (as **CFF** or **CodeMeta**) on **GitHub** with **software registries** (mainly **Zenodo** but also **others**). It should also mention and try to facilitate via general-purpose tooling/libraries possible solutions for other platforms, such as GitLab and Bitbucket.

As a result of this workshop, Zenodo will set up and curate a new GitHub Action.

This GitHub Action will help to make software citable. The action will add badges to README of repo, which would be a tangible benefit for the user (and is easier to sell than adding a CFF or CodeMeta file). By having a centralized action, we can better enforce best practices, such as ensuring that only one DOI is assigned to a release.

Workflow

Kim is a research software developer who has a repository on GitHub where they keep their software. Kim is getting ready to make a first release.

After their first release, with a CiteAs integration into GitHub, a checking program run via the CiteAs server shows Kim what items they are missing to make their release more citable. The CiteAs citation provenance checklist dynamically shows Kim that they can create either a CITATION.cff or a CodeMeta file to make their software more citable. The checklist will give Kim a link to a CodeMeta file creation form so Kim can start from there.

This file contains only the minimum absolutely necessary metadata for the proposed workflow to work: A list of authors, and the name of the software. As the software is not yet released, CiteAs cannot yet know the version number or DOI for the version, and therefore does not include this in the file.

Because this is their first release, and Kim wants a DOI to be provided by Zenodo, and for Zenodo to archive this and future releases, they activate the GitHub action provided by Zenodo on this repository.

Kim next tags their software with a Git tag: “v1.0.0”. They then push it to GitHub. GitHub reacts by creating a draft release on the releases page in Kim’s repository. Kim simply gives it a title that is the original Git tag: “v1.0.0”, adds a description, and pushes the **Publish release** button.

The Zenodo-GitHub action now kicks in because Kim has published a release. The GitHub action interacts with Zenodo to create a DOI for the released version v1.0.0, and also to create a concept DOI, which collects version 1.0.0 and all future releases that Kim will publish on Zenodo. (This happens the first time Zenodo see a version of a repository.) In the process of creating these DOIs, Zenodo also picks up the version string from the GitHub release (“v1.0.0”), and the current date (2021-11-14). It also uses the minimal citation metadata that Kim had supplied in their metadata file (CITATION.cff or codemeta.json) to complete the Zenodo metadata for this release.

Next, Zenodo updates the citation metadata file picked up from Kim’s GitHub repository (CITATION.cff or codemeta.json, depending on which Kim had created) with the metadata it has recorded for the release. The metadata file - let’s assume Kim is using the Citation File Format - now provides up-to-date citation metadata: list of authors, name of the software, version string, version DOI, concept DOI, release date.

For human users, the CITATION.cff file now also includes information about what the version DOI and concept DOI pertain to in a “message” field, the value of which will now contain something like: “If you use this software, please cite it using these metadata. The version DOI (<https://doi.org/10.5281/zenodo.1234>) refers to the latest released version of this software. The concept DOI (<https://doi.org/10.5281/zenodo.1233>) refers to the collection of all versions.”

For machine users, this information is encoded in the *identifiers* section of the file: The version DOI uses the “is-part-of” relationType to point to the concept DOI, and the concept DOI uses the “includes” relationType to point to the version DOI.

Next, the GitHub Action attaches the updated file to the GitHub release page for version 1.0.0 of Kim’s software. This way, when a user finds the page for the release, and wants to cite that release, they are presented with all the necessary, up-to-date citation information.

The GitHub Action also commits the updated citation metadata file to the root of Kim’s repository. This way, if a user of Kim’s software finds the landing page of the repository, they find up-to-date-enough citation information for Kim’s software. Because the metadata file at this stage makes it clear what the two DOI relate to, and also contains the version string (e.g., "v.1.0.0") for the latest released version, the user can determine whether this is the version they have used and want to cite. If it isn’t, they are also informed that the concept DOI resolves to the collection of all released versions of the software. This DOE will resolve to the landing page for the latest release of the software, which also lists *all* released versions of the software. From this list, the user can pick the version they want to cite. If Zenodo will at some point resolve concept DOIs to an actual landing page for the “concept” of the software, this will also list all released versions in some form.

Now Kim is happy: they are getting cited because the users of their software have access to the relevant correct information, and it’s easy for Kim to keep the metadata file up-to-date.

Kim simply has to change the list of authors if and when necessary, and add their ORCIDs, which are also picked up by Zenodo.

Kim's users are also happy: they can find and understand the necessary software citation metadata right in the GitHub repository, and also on Zenodo.

Zenodo users are also happy: the list of authors for the software version they found on Zenodo and want to cite is always up-to-date and author-curated, rather than a best guess.

The GitHub Action that is curated by Zenodo is generic, so other software registries can easily adapt it to their own identifiers, which may not be DOIs. So after a while, Alice and Wes could publish a version of the GitHub Action for ASCL.

As Kim can see the value in having an ASCL identifier for their software, they could activate the ASCL GitHub Action in addition to the Zenodo GitHub Action. After doing so, whenever Kim publishes a new release, the ASCL GitHub action will read the existing metadata and populate the ASCL entry metadata with it, and also update the citation metadata file with the *asclID* for the latest released version. Now Kim has citation metadata for entries of the same software in two different registries, but thanks to automation, these two identifiers will always be in sync, so Kim can finally relax, and count the citations to their software that come flooding in.

Minimal starting point

Starting from scratch - Have a static page that takes a url (probably GitHub) and returns starting CITATION.cff or codemeta.json file to be added to repo. Could be a call to CiteAs API?
Starting from scratch: Use the CFF initializer website at https://citation-file-format.github.io/cff-initializer-javascript/
Updating after preservation - Adding action to get metadata from Zenodo and updating CITATION.cff or codemeta.json
Updating after preservation - Let the CFF initializer (see above) resolve registry identifiers to pre-populate the form?

For reference:
Software citation best practices for Netherlands eScience Center (CFF-GitHub-Zenodo): https://bit.ly/citable-software