Implementing Software Citation

Force11 Workshop on Data Citation Implementation
Montreal, October 10, 2018

Adopted from
https://docs.google.com/document/d/1ze2Bh0pZXCy7_bHcC7CumQRmBAv8qP6reao4yU4JToY/edit?ts=5b8e056c#

A. Flowchart draft:

1. Citing software (author)
    1.1. Has it been cited before?
        1.1.1. Check for a CITATION file or README; if this says how to cite the software itself, do that
        1.1.2. Find other locations for this citation, e.g. Bibtex file
    1.2. If not, follow these principles:
        1.2.1. List the authors: if the software developers declare who the authors are, list them; otherwise, just name the project (Project X for open source software, Company Y for commercial software) as the authors
        1.2.2. Add a Global Unique Identifier: Include a method for identification that is machine actionable, globally unique, interoperable – perhaps a URL to a release, a company product number
        1.2.3. Add metadata: If there's a landing page that includes metadata, point to that, not directly to the software (e.g. the GitHub repo URL)
        1.2.4. Add version/release info: Include specific version/release information

2. Making 'your software'[1] citable (developer):
    2.1. Publish it
        2.1.1. If it's on GitHub, follow steps in https://guides.github.com/activities/citable-code/
        2.1.2. Otherwise, submit it to Zenodo or Figshare, or a suitable domain repository, with appropriate metadata (including authors, title, citations of, & software that you uce)

[1] What is 'your' software? A few scenarios:

- Fork software and contribute to it: this is **your software**
- To existing software you have added a wrapper that adds features/functionalities: this is **your software**
- [Not your software] Pull request to other software, which you modify: this is **someone else's software**, that you have made a contribution to and is incorporated into original software: cite original creators

2.1.2.1. Metadata: should follow CodeMeta and the Citation File Format, e.g through universal software archive Software Heritage

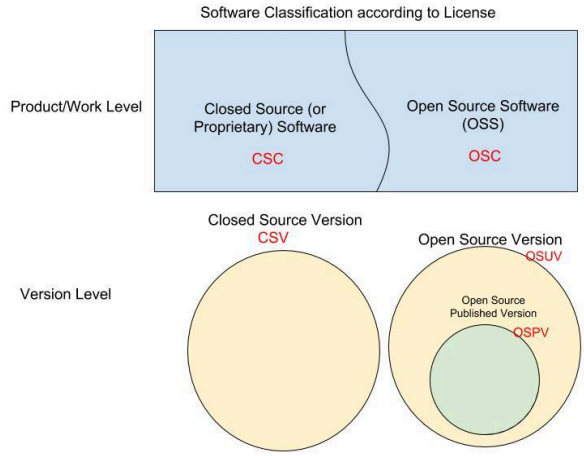
2.2. Get a DOI (by doing A, B, C…)

2.3. Create a CITATION file, update your README, tell people how to cite it

B. Software typology

There are many types of software, including source code, containers, executables, etc. We make the following distinctions[2] (see Figure for a summary):

- open source or closed source.
- a concept (all versions of the software) or a set of individual versions.
- published and archived (e.g. on Zenodo) or unpublished (e.g. shared on GitHub). [Note that Github is not archive or publisher, it makes no promise about longevity of itself or the material it holds; it also does not generate identifiers for the material it holds.] ***>>> Need more here, why is Github not a publication?***
- different versions of open source software can be published, and that those versions (and their corresponding DOIs) should then be cited



[2] These different types of software are often used together and have explicit and implicit dependencies. Konrad Hinsen describes four layers of scientific software that are composed in a stack when used, starting with 1) *project-specific software* like scientific workflows and scripts (SWS), that depend on 2) *discipline-specific software* libraries and applications, such as shared community models. These in turn depend on 3) general *scientific software infrastructure*, such as numerical libraries like LAPACK and NumPy, and all of this depends on 4) *general purpose, commodity software* such as operating systems and programming languages like python. While layers 2, 3, and 4 are generally written as re-usable software packages that should be independently citable, the project-specific software in layer 1 is generally not written explicitly for reuse, and when archived it is typically in mixed-type data packages. In addition, these research grade scripts and workflows often are implicit about their specific dependencies and their versions from layers 2-4. For example, the Chlorophyll R script from the Gentry data package above explicitly depends on 10 open source R packages (ggplot2, maps, sp, maptools, scales, mapproj, RColorBrewer, rworldmap, matrixStats, data.table), but does not specify the specific versions needed, and implicitly depends on the recursive dependencies of those imported packages as well as unspecified versions of the R language (OSC) as well as all of the underlying layers described by Hinsen.

The table summarizes, for each type, how they can be identified and cited:

Category	Definition	Example	How to identify	How to cite
Closed source version	A version of closed source (possibly commercial) software.	version 14.3 of SAS/STAT.	URL to a release, or a company product number	Gather metadata needed for citation and format appropriately.
Closed source concept	The software concept for a closed source (possibly commercial) software package	SAS/STAT	URL to the software, or a company product number	Gather metadata needed for citation and format appropriately.
Open source published version	published version of an open source software package	version 1.2 of the Application Skeleton package (https://github.com/applicationskeleton/Skeleton), found on GitHub at https://github.com/applicationskeleton/Skeleton/releases/tag/v1.2, published by Zenodo as https://doi.org/10.5281/zenodo.13750	The software is identified by the publisher, likely with a DOI.	Metadata we need for citation should be available as part of its publication Cite the software similarly to other published digital object, with the exceptions of identifying it as software, by adding "[Software source code]" or similar in the citation, and identifying the version
Open source	unpublished version of an	https://github.com/applicati	E.g. through Software	1.Look for the metadata in the archive of the repository,

unpublished version	open source software package	onskeleton/Skeleton as of commit 81c66c0db5c381dacc0841a4c16e0b3876b15b89.	Heritage, with search and browse capability	e.g. AUTHORS file that tells me the authors of the software. 2. If none, name the authors as "Application Skeleton Project." 3. If the software has been previously versioned, use the previous version's citation information as a starting point, adding to it any updates made to the AUTHORS, CITATION, README, ACKNOWLEDGEMENT etc. files as appropriate, along with the short commit hash.
Open source concept	The software concept for an open source software package	the Application Skeleton package (https://github.com/applicationskeleton/Skeleton)	Identified by the software repository, and is sometime represented by a software paper or the software users manual	Organizations who register DataCite DOIs can take advantage of the built-in support for codemeta, e.g.: https://doi.org/10.5438/qeg0-3gm3, points to https://github.com/datacite/maremma.

C. Metadata for software citation

The generally required metadata for software citation, from the Software Citation Principles are:

- Identifier
- Software name
- Authors - can fall back to "[Software name] Project" for open source projects, or "[Company name]" for commercial projects.
- Type: Software
- Version/Release date -if applicable
- Location/Repository

Metadata for software citation can be stored and made machine-actionable in a software repository in one of two formats:

- CodeMeta.json is a machine-actionable general-purpose exchange format for software metadata expressed in JSON-LD format (i.e. JavaScript Object Notation for Linking Data). The use of JSON-LD in the CodeMeta specification means that the software metadata includes some structure (in JSON format) and can be extended with semantics via context files, which provide mappings of the elements to vocabularies. The provided context in CodeMeta relies mostly on the schema.org vocabulary.
- CITATION.cff: The Citation File Format (CFF) is a machine-actionable, human-readable and -writable metadata format based on the Software Citation Principles. CFF relies on the YAML format for representing the citation metadata, which is a superset of JSON (i.e., JavaScript Object Notation)

Both formats can be used to provide citation-relevant metadata for a software but CFF is a "front-end" format, and CodeMeta is a "back-end" format. Although CFF is the suitable format for the initial provision of software citation metadata by the creators of a software, the metadata should be transferred to CodeMeta to leverage CodeMeta's linked-data features.For example, the Netherlands eScience Center's Research Software Directory (e.g., https://research-software.nl/software/xenon) uses CFF for the citation metadata input (converted to BibTeX etc.) and then also use it to generate CodeMeta JSON-LD using cffconvert, which can be embedded in a landing page for re-use by search engines.

The differences between the two are presented in the following table:

Feature	**CodeMeta**	**Citation File Format**
Metadata type	*general-purpose metadata*	*citation metadata*
Targeted at	*machine exchange*	*human and machine actors*
Linked data	*yes*	*no*
Self-documenting	*no (perhaps via file name)*	*yes*
Enforces Principles	*no (?)*	*yes*
Secondary references	*yes (general)*	*yes (scoped)*
Implementation	*JSON-LD*	*YAML*

And? So? What should the user do??

Once the citation metadata is known and stored, it can be presented in various formats, such as a text citation, or in a format such as bibtex, RIS, etc. This metadata can also be stored in a reference manager, which can then do this conversion.

For example, DataCite's Content Resolver can return a representation of DOI in different formats. An example of this is returning the metadata for version 1.2 of the Application Skeleton package (with doi https://doi.org/10.5281/zenodo.13750) in bibtex:

```
> curl https://data.datacite.org/application/x-bibtex/10.5281/zenodo.13750

@misc{https://doi.org/10.5281/zenodo.13750,
  doi = {10.5281/zenodo.13750},
  url = {https://zenodo.org/record/13750},
  author = {Katz, Daniel S. and Merzky, Andre and Turilli, Matteo and Wilde,
Michael and Zhang, Zhao},
  keywords = {computer science, application skeleton, co-design, distributed
computing, many-task computing, parallel computing},
  title = {Application Skeleton V1.2},
  publisher = {Zenodo},
  year = {2015}
}
```

NB: Type is "misc", not "software", since there is no software type in bibtex

Next steps:

1. What special cases are missing from the list above (q for Dan & rest of the group)?
2. What can be taken out?
    a. Do we need all these software types? As a test: how many of the examples under 3.a. fall in each category?
    b. Do we need more data on metadata?
3. What more do we need to do before sending this?
    a. Examples, from 5 domains (e.g. life, physical, earth, social sciences & computer science, e.g. NLP/data science, random Jupyter Notebook):
        i. Software examples (e.g. 5-10 per domain)
        ii. Metadata for these
        iii. Example citations for these
    b. Better description of:
        i. What metadata is required exactly
        ii. How to create it
        iii. Tools for creating metadata, exhaustive/editable list
    c. Easily legible/well-formatted version of the flowchart
    d. A website showing a-c (e.g. on Force11 website?)
4. Who do we want to reach?
    a. Open Source community
    b. Publishers
    c. Code repositories
    d. Other efforts:

        i. Scholix?
        ii. Make Data Count?
        iii. Enabling FAIR data in ESS
        iv. Others?
        v. RDA community?
5. What do we want from them:
    a. Sign on/off? How?
    b. Implementation: how do we know it worked?