Guidelines for persistently identifying software using DataCite

A JISC research Data Spring project

Version 1.0

Ian Gent, University of St Andrews; Catherine Jones, Science and Technology Facilities Council and Brian Matthews, Science and Technology Facilities Council.

September 2015

Contents

1. Introduction

Software underpins the academic research process, regardless of discipline. With the increased focus on the long term value of data and other research outputs, more attention needs to be paid to how software used in these processes is both identified and preserved for the long term as much data are meaningless without the related software.

Key reasons for persistently identifying software include the following.

- To ensure that others can identify the software used in a specific circumstance; this may be related to a publication describing the software.
- To enable citation of the software so that appropriate credit can be given to the creators (rather than using citation of a publication describing the software as a proxy for the software.)
- To be able to be sure that the correct experiment/software can be rerun to verify results recorded elsewhere
- To enable the preservation of software, unique identification enables the repository to know what is in the collection.
- To identify different versions as they would have different persistent identifiers

Many of these reasons are underpinned by the assumption that the software can be found. To be able to find and reuse software the following types of information may be used. It should be acknowledged that the discovery metadata doesn't need to be as detailed as metadata to enable reuse and that some of this information applies to many types of research output.

- **Purpose**: what was it designed to do
- **Programming language:** what is it written in? If the intention is to be able to modify the retrieved software, then what it is written in, may be a decision in whether it is suitable for the end-users needs and hence may form part of the search criteria.
- **Environment:** what tools and operating system will I need to be able to run or modify it? The end user may be looking for example for a Linux version rather than a windows version.
- **Who wrote it:** Do I trust them & their organisation?
- **Where does it live?** Do I trust the software repository, what version does the identifier point to?
- **What license is it issued under?** What does this enable me to do with the software?

The purpose of this document is to encourage consistency in the discoverability of software, by giving guidance to those who are intending to use DataCite DOIs as the persistent identifier scheme for software.

We note that DataCite DOIs are not the only persistent identifier scheme available; however there is an increasing use of DataCite DOIs for artefacts and there is acceptance of DOIs in the academic community due to the use of CrossRefs DOIs for scholarly articles.

1.1. Scope

This document is aimed at those who create or manage software which has the potential to be long lived but are not necessarily software engineers and those in institutions who support those developers. We assume that the software produced will be created within the academic domain to solve/address research issues and so this document doesn't address commercial software, whether or not it is used in research. This software may be written by a single person, a sole developer or PhD student, or it may be written by a team of developers, but it is not commercially focussed. Much of this software will be written by domain specialists rather than software engineers and so they may not have had much formal training in software development best practice. Some of this software will have a short development life span but yet may make an important contribution to the analysis or understanding of a specific topic. Other software will have a long life time and have many developers working on it. The unifying factor is that the software has value to the research community and so needs to be permanently identified so that it can be unambiguously referenced and accessed.

2. What is software?

To be able to discuss the issues around persistently identifying software, there needs to be a common understanding of what is meant by the term *software*. A reasonable starting definition below is from Wikipedia, and demonstrates the issues around where the application program stops and the environment needed to run it starts.

> *"Computer software includes* computer programs, libraries *and their associated documentation. The word software is also sometimes used in a more narrow sense, meaning* application software *only."*[1]

In this definition there is nothing explicit about what format the computer software will be in: is it source code which needs to be compiled/interpreted to be able to use it, is it some form of running code? Having the source code doesn't necessarily mean that one will be able to get it to work without a good understanding of the dependencies and operating system it was designed for.

Thus digital objects referred to as "software" may have quite different physical manifestations, and we should be clear what is being referenced using a particular identifier so that user has an idea of what artefacts are being referenced and how they may be used when accessed. We propose a model of software entities to assist in describing these artefacts.

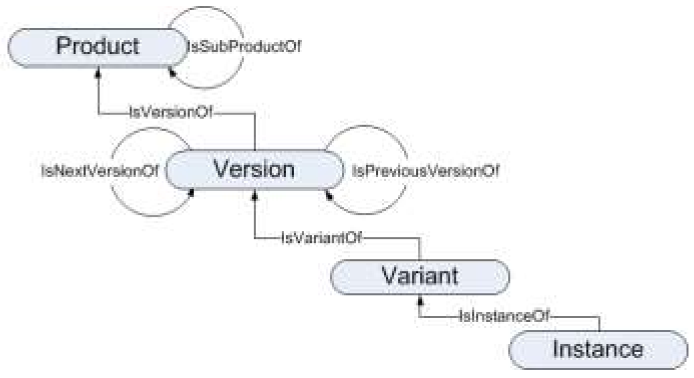
2.1. A Model of Software Entities

Software development generally takes place within a project lifecycle of design and release. As software evolves, changing its scope, functionality and the constraints imposed by the computing environment, new digital artefacts are made available – as "versions" with particular features. So while in some circumstances, a user may want to refer to the whole software

[1] https://en.wikipedia.org/wiki/Software Retrieved 12/6/2015

entity across its lifetime, in other cases, a particular version may be references, supporting a particular set of characteristics.   For example, an organisation may declare that it supports Microsoft Word in general, but for a particular document, a particular version of MS Word may be required.

To aid with distinguishing what software entities should be persistently identified, the following model describes different software entities and their relationships across the lifetime of a software product. It may be appropriate to assign persistent identifiers at any and each of these four levels depending on the use it is designed to support, though in general it would be acceptable to miss out some of these entities.



We define four entities reflecting the software lifecycle as in the diagram above

MANTID EXAMPLE
Mantid is an open source development for scientific data analysis used within the Neutron Scattering community It assigns DataCite DOIs for attribution and reuse

There is a **product** level DOI for the whole software package http://dx.doi.org/10.5286/SOFTWARE/MANTID. Each new version has new functionality and different developers and so there is also a **version** with its own specific DOI. e.g. http://dx.doi.org/10.5286/SOFTWARE/MANTID3.2.1

The relationships between these objects are identified in the DataCite Metadata.

Product: the top-level conceptual entity encompassing the whole development and release lifecycle of the software. It is likely to be how the system may be commonly or informally referred to. Using an identifier at this level may be appropriate to reference the general concept of a particular software artefact regardless of the specific version, or the continued use of this software over a long period,. It of use if different versions are going to be referenced as it can stand as a unifying record.

Version: The version of a software product is a single, coherent release of the product with a well-defined functionality and behaviour. It usually includes the way in which it interacts with the computing environment. This level may be needed to identify a specific team of contributors, or where a particular functionality of the version (which may be different in other versions) is being referred to. For example, this may be required for validation of results.

Variant: Versions may have a number of variations adapted to different operating environments (e.g. version of Windows, MacOS or Linux). A software Variant is usually a manifestation of the product that is adapted for deployment in a specific software operating *environment*. In this case, the functionality of the version is maintained as much as is practical. However, due to the different behaviours of different platforms, there may be variations in product behaviour, such as in error conditions and user interaction. This may be the appropriate level to use for validation of results or emulation.

Instance: An actual occurrence of a software product which is found on a particular environment or machine is known as an Instance. It may be also referred to as an installation or deployment. This would be an appropriate level for packaging, citation and so reproduction of results in a virtual environment.

Note that in this model, the actual physical object which is being referenced by an entity is most likely to be an *aggregation* of other objects – i.e. a collection of files which constitute the software entity itself. These files may be executables, source code, configuration files, documentation and other objects. The model of software above does not specify in general what will be in these aggregations; we discuss these issues in the next section.

3. Issues in persistently identifying software

Within a software version or variant, the actual digital artefacts in the aggregation are not specified, but a user will want to know what digital objects they will access when they dereference a persistent identifier. In this section, we discuss what is it exactly that is being identified and what you need to know about the build and runtime environment.

3.1. What are you identifying?

Different users might have different needs for identifying software. One user might wish to refer to the source code, while the executable might be critical for another. Another might need to refer to a particular version of an executable (for example with debug information). Yet another might need to refer to a particular installation of the software, either in general terms (MS Windows) or very specific terms (the version installed on a particular machine in a particular office on a particular date.).

Typically software is composed of several items packaged together. These might include binary files, source code files, installation scripts, libraries, usage documentation, and user manuals and tutorials. A more complete record may include requirements and design documentation, in a variety of software engineering notations (for example, UML), test cases and harnesses, prototypes, even in some cases, formal proofs. These items may have their own persistent identifiers.

As well as the composite nature of software, it is also a dynamic artefact as new versions are produced as errors are corrected, functionality changed, and the environment (hardware, operating system, software libraries) evolves. But earlier versions may still need to be recalled to reproduce particular behaviour.

3.2. Build and runtime environment

A key aspect of software is the environment it runs in. This can be the hardware environment, the core operating system, or other aspects of the environment such as the programming language used, and installed packages. Different versions of the same software may need entirely different environments, platforms or operating systems as well as changes in the versions of libraries or programming languages supporting them. Software run in different environments can be very different or very similar from the point of view of the person identifying or citing it, and so we have to provide the facility to identify it as appropriate. Just as in other areas, there are many levels of detail that might be necessary. It might be enough to refer to a Windows, Mac or Linux version. Or a very fine-grained reference to a version might be necessary. Similar variation in need of precision exists with hardware, for example 32- or 64-bit versions. As well as traditional hardware environments, we now commonly have distributed and virtualised environments, either locally or in clouds. Capturing complete environment information is a complex task, especially where dependencies are concerned, however basic information can be identified and added as a starting point.

An example of software set-up being important is the difference between Python 2 and Python 3 as they are not backwards compatible, so that programs written using Python 3 will not run in a Python 2 environment. Whereas documents created in Microsoft Word 2003 can be opened in Word 2010.

4. Recommendations for DataCite Metadata fields

DataCite DOIs have emerged as a solution for persistently identifying a number of research objects, initially for data, but now including grey literature, theses and software. The metadata schema that DataCite has developed, in discussion with the research community, must be flexible enough to accurately cite and describe this wide range of research objects, in a multitude of disciplines, countries and other contexts.

This section gives guidance on the use of specific DataCite metadata fields when applied to software. Every piece of software is different and they live in different organisational contexts, so it is not always possible to give a single piece of guidance which applies to all situations, so this section has adopted an approach of identifying relevant issues; giving examples of what is already in use; and highlighting where we feel there is a need for changes to DataCite fields themselves.

For each property we have identified as having specific uses for software there is a table which gives the DataCite Description of the property; some discussion, examples and recommendations as to how it might be applied to software; identification of stakeholders in the contents of the property; and then some questions for those applying the values to ask themselves to enable them to reach the best decision for their organisation.

For cross-institutional projects there should be a decision made as to who should issue the DOI for the software, bearing in mind the long term nature of such a decision. For software there are alternatives to academic institutions as code repositories also offer this functionality. Curation, credit and user behaviour should all be taken into consideration.

4.1. DataCite mandatory fields

These fields are those that are required in order to assign a DataCite DOI to any research object. They form the basis of the citation and are key to discoverability. They are also not necessarily as straightforward as could be anticipated due to the complexities of describing software.

DataCite Property	**Identifier**
DataCite Description	The Identifier is a unique string that identifies a resource
Purpose	This is the DOI itself. After the designated prefix, the issuer can set the rest of the unique identifier. Approaches taken include: • Automatically assigned string • Indication of issuer/automatically assigned string • SOFTWARE/Version details These reflect the differences in scale of issuing. It is unlikely that the identifier will be used as part of a search strategy when the item is not known.

Stakeholders	ISSUER: Is there a local policy for identification naming schemes?
Questions	How many DOIs for software are going to be issued by the organisation? Is the solution scalable? Are there any existing institutional policies for DOI naming than need to be followed? Is it important to you that the DOI has meaning?

DataCite Property	**Creator**
Description	The main researchers involved in producing the data, or the authors of the publication, in priority order.
Purpose	To identify the main people responsible for the software to enable recognition and credit to be given. The creator of software may not be straightforward to ascertain, as software has a long life-span and may be worked on by many people. The point during its development cycle that the first DOI is given may also affect those identified as creators. The creators need to be listed in order of importance, so it should be clear how this is decided upon. It could be down to the amount of code contributed, or decided by the level of input into the design and architecture. See also the entry on contributors Examples: ***Student project/single developer*** In this case, there is a sole developer and so the identity of the creator is straightforward. ***Project team – DOI on first production release*** In this case, the DOI is assigned at the point of the first production release and so the current team should be straightforward to identify. ***Project team- DOI after several years of production releases*** If the first DOI comes a significant time after the original release then it may be hard to identify all those who contributed to creating the software. The current release may have been worked on by a small team but the intellectual content will have built on the work of others. To a certain extent, who is credited may depend on the decisions taken on how future releases may be identified. If

	future releases are to be given their own DOI, then a general "top level" Product may be given a DOI with all creators and then future releases be they variants/versions/instances only name those who people have been involved in that object. This is the approach taken with MANTID ***Project team every major version released has a DOI*** In this scenario, every version is persistently identified, so that the creators for each DOI record can reflect those who contributed to that specific version and the versions can be connected through relationships.
Stakeholders	CREATOR: Will want appropriate recognition and credit. END-USER: May be used as part of relevance identification and may form part of a search strategy
Questions	Is it obvious who should be credited as a creator? How important is affiliation? If subsequent releases/versions will have a DOI, is there a strategy for ensuring correct attribution?

DataCite Property	**Title**
Description	A name or title by which a resource is known.
Purpose	Contains the most information in a mandatory field which can be used by an end-user to locate and then work out what the resource is. The title may be obvious but if there are granularity issues then the title chosen must be distinct from others to enable effective discrimination between objects.. Recommendation that abbreviations are also put in full.
Stakeholders	CREATOR: The title used is part of the "brand" of the software END-USER: Important piece of information for relevance decisions and the mandatory title is an important part of the search retrieval results.
Questions	If it a piece of software written by a single person for a specific project does it actually have a name? If this title is being used to find software is the official name different from the common name? What effective is versioning or branching of code going to have on the name? Are there any naming conventions that need to be adhered to? Will the name used be unique enough for it to be found and distinguished from other search results?

DataCite Property	Publisher
Description	The name of the entity that holds, archives, publishes prints, distributes, releases, issues, or produces the resource. This property will be used to formulate the citation, so consider the prominence of the role.
Purpose	This will be an organisation or corporate body who is responsible for releasing the software. It may not be the same as the issuing body as it may be a specific part of the organisation. The concept of a Publisher in this context for academic software is not well established especially if the code repository/dissemination is done from an institutional basis This is used in the citation so it needs to be a meaningful entity. This may be set by local policy. Approaches to date include • Corporate name of project • Issuing body • Centre/Body within the issuing body
Stakeholders	ISSUER: will have a view on who should be acknowledged as the Publisher CREATOR: Depending on how DOIs are assigned the CREATOR may not have a say in this. However they may have a view on who should be acknowledged as the Publisher.
Questions	Is the publisher set by local policy? With what body/organisation is the software associated?

DataCite Property	PublicationYear
Description	The year when the data was or will be made publicly available.
Purpose	Whilst identifying the year in which the DOI is issued is not a problem, care must be taken to ensure that if there are multiple versions released in one year, that other DataCite metadata or information, such as that on the landing page, enables users to identify the newest one.
Stakeholders	ISSUER: may have a local policy
Questions	Will there be more than one release per year? How will the latest release be identified? Will different releases be linked to each other? How will that be achieved?

4.1.1. DataCite optional fields

This section does not cover all DataCite optional fields, we have chosen to concentrate on those which we feel are particularly relevant to software generally.

DataCite Property	**Subject**
Description	Subject, keyword, classification code, or key phrase describing the resource.
Purpose	This can be used to add contextual information to the metadata record which can be used for retrieval. Adding relevant subject information will depend on the audience and the purpose of the code. For example if the software does a specific analysis on specific data, then using a very detailed subject/keyword would be appropriate. If the software can be used for multiple purposes, then a more general term maybe more useful than a list of specific terms. The additional information could come from an existing thesaurus or could be a user generated tag/keyword. This may depend on the user communities' general practice.
Stakeholders	END-USER: may be interested in software for a particular discipline and subject could assist in discovery of this.
Questions	What is the purpose of the software? What additional keywords, not already in the title, would help users to find this software? Would using a controlled term from an existing thesaurus help, or is a tag/keyword better?

DataCite Property	**Contributor**
Description	The institution or person responsible for collecting, managing, distributing, or otherwise contributing to the development of the resource.
Purpose	If this is used, then there is a controlled list of contributorTypes. Those of specific relevance to software are discussed below (this is not an exhaustive list): • ContactPerson • Distributor and HostingInstitution: If others are involved in the dissemination of the software or run code repositories etc, then they may be acknowledged. • Funder: For software developed under a grant then it is appropriate to recognise this. • ProjectLeader, ProjectManager and ProjectMember: For software which has been developed as part of a project, then it may be appropriate to acknowledge others who were not part of the software development team but contributed to the success of the project. • RightsHolder: If there is a specific license, then it may

	be appropriate to add the rights holder to the record
Stakeholders	CREATOR: This enables context and attribution for the software
Questions	Who else should be acknowledged in the DOI? If the intention is to create multiple DOIs, will the same people be acknowledged as contributors in all DOIs?

DataCite Property	**Date**
Description	Different dates relevant to the work.
Purpose	As well as the publication date, it is possible to add other specific dates to the metadata. There is a controlled list of dateTypes. Those of relevance to software are: • Available: This could be used to denote when open source software was opened up to Beta testers. • Issued: This can be used to denote the release of a version. If the code is being frequently revised/released then this can help in identifying the latest version. Updated: This could also be used for version dating, however as a significant change will lead to a different DOI being issued, we believe that issued is a better term
Stakeholders	CREATOR: A mechanism to ensure good versioning information. END-USER to be able identify the difference between versions of the software which are published in the same year.
Questions	How often will the software have a new release? Is it likely to be less than a year? What additional date information will add value to end users?

DataCite Property	**ResourceType**
Description	A description of the resource.
Purpose	If this is used then there is an additional term, called ResourceTypeGeneral, and the possibility of adding further information. Possible controlled values for resourceTypeGeneral which may apply to software in the widest terms: • InteractiveResource: defined as a resource requiring interaction with a user. If the DOI points not to source code, but to a runnable version of the software, then this may be a more correct term. However it depends on the use of the resource once found and the end-users understanding of the term. • Service: defined as a system which provides value to

	the end user. Running the software may well provide a service, especially if the DOI is pointing to a runnable resource. Software: defined as a computer program in source code or compiled form.
Stakeholders	CREATOR to denote the type of resource END-USER as part of search strategy to locate the correct resource.
Questions	Are these adequate? How accurately does it describe the object?

DataCite Property	**relationType**
Description	Description of the relationship of the resource being registered (A) and the related resource (B).
Purpose	This enables relationships between other records to be made. This is particularly important for software which is being actively developed as each major version will have a different identifier and thus distinguishing which is the most recent version may not be straightforward. Relationships of particular relevance to software are: *Controlled List Values:* • IsContinuedBy and Continues: Forked versions? • IsNewVersionOf and IsPreviousVersionOf: Denoting relationships between releases • IsPartOf and HasPart: If individual modules, parts of greater system • IsDocumentedBy and Documents: Information on documentation • IsVariantFormOf and IsOriginalFormOf: Could be used for different versions for different operating systems/compilers etc IsCompiledBy and Compiles is not intended to be used in the Computing sense of the terms. Dependencies may need a new relationship type as References/IsReferencedBy doesn't have quite the right connation. Using the Software Entities Model, we suggest the following relationships: - **Sub-product DOIs** are related to **product DOIs** using the DataCite metadata field relationType with value IsPartOf to substitute isSubProductOf. - **Version DOIs** are related to **product DOIs** using the DataCite field relationType with value IsPartOf to substitute isVersionOf.

	- **Variant DOIs** are related to **version DOIs** or **product DOIs** using the DataCite field relationType with value IsVariantFormOf or IsPartOf to substitute IsVariantOf. - **Instance DOIs** are related to **variant DOIs** or **version DOIs** or **product DOI**s using the DataCite field relationType with value IsPartOf to substitute IsInstanceOf. - **Versions DOIs** are related to each other, using the DataCite field relationType with values IsPreviousVersionOf and isNextVersionOf.
Stakeholders	CREATOR: To show the relationships between different resources to aid in understanding and discovery END-USER to understand the different related resources.
Questions	What relationships are of use to you & the end users?

DataCite Property	**Format**
Description	Technical format of the resource
Purpose	This is a free text field, the recommendation is to use the file extension of MIME type. Depending on the programming language used, there are MIME types for the standard languages, however this is probably not as useful as stating the language in plain text.
Stakeholders	CREATOR: To give information about the format of the resource END-USER to be able to understand what tools will be needed to view and use the resource.
Questions	What is the most meaningful value to put in this field? Is it likely that this field will be used in an automated fashion for anything? In which case using the formal MIME description may be appropriate.

DataCite Property	**Version**
Description	The version number of the resource.
Purpose	The recommended practice is to assign a new DOI for a major change of version. Good software engineering practice has always tracked changes and given them version numbers. It is recommended that this field is used for this as standard practice. The description field can be used to add a textual description of the change.
Stakeholders	CREATOR: To identify the resource precisely END-USER to be able to distinguish versions from each other
Questions	Does your software have a version number?

	What numbering scheme are you going to use? Has the way the software is versioned changed? How will that be identified? If the software forks how will the versioning identify this?

DataCite Property	Rights
Description	Any rights information for this resource.
Purpose	Ensure that the rights/license is explicit.
Stakeholders	CREATOR: To assert rights on the software END-USERS: To understand the rights on the software
Questions	Have you decided on the license to use for this software? Is it freely available? If not is it clear who holds the rights?

DataCite Property	Description
DataCite Description	All additional information that does not fit in any of the other categories. May be used for technical information.
Purpose	This enables extra information to be added to the metadata record. There is a controlled list of descriptionTypes and currently the main ones used for software are “Abstract” and “Other” What is actually entered in these fields falls into four main categories • More information about the purpose of the software • More information on releases • Information on where the active development is taking place as code deposited with an article • No additional information. This field must be part of the Zenodo requirements and if it is not filled in then a standard piece of text is added. Using this field should help potential reusers to discover whether it is appropriate for their own purposes. It can be used in the Advanced search on the DataCite search function. So categories of additional information might include: • Purpose as the title may not be descriptive enough • Programming language used and other technical information • Code repository where live development is taking place. Under the current descriptionTypes then it is recommended to use Abstract. Going forward we recommend the adoption of a new descriptionType of TechnicalInfo. We also suggest whether

	summary would be more appropriate than Abstract.
Stakeholders	END-USERS: This would be used for discovery of the software and to assist in end users assessing relevance of the software to them.
Questions	What information would someone need to know to make a decision about whether to use the software without accessing the digital object.