Enhanced Metadata Support

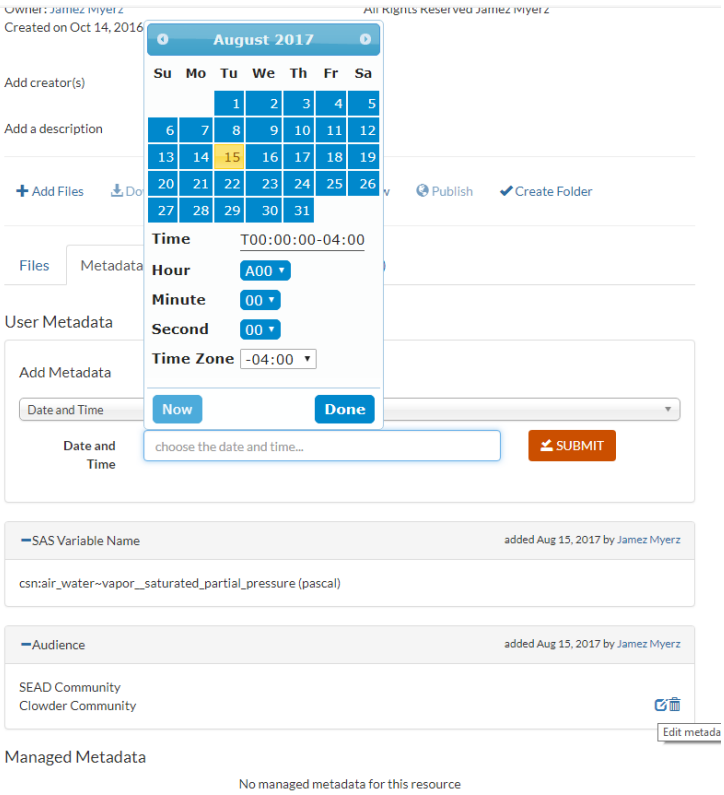
This page describes the updates made in the Editable Metadata Branch to provide enhanced support for managing metadata terms, and annotating resources, with a new design.

Prior Design

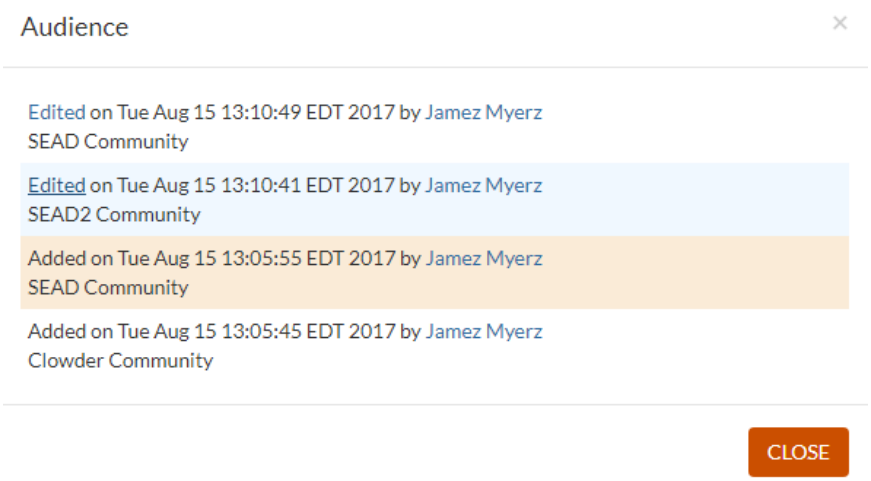
- Individual metadata entries are stored as Mongo documents and are associated with a file or dataset (comments say metadata can be attached to collections but this is not implemented in the GUI) and a JSON-LD context.
- Entries through the GUI usually result in one term/value per entry, but some types (e.g. geospatial) produce more than one term/value/a json value that is displayed with structure. Entries through the API can include multiple term/values which can only be deleted as a unit. Deletion simply removes a metadata document and does not track provenance.
- JSON-LD contexts are stored as another document collection and can be referenced by multiple metadata documents, but these contexts are not aligned with the space metadata choices (e.g. existing metadata docs and contexts are not updated with changes to a space's choices for metadata terms. That only affects the terms available as input options in the GUI). This allows multiple labels to be in use for one semantic term and for multiple terms to use the same label (in different entries).
- Metadata management allows space admins to add/edit/delete entries starting from an instance default list controlled by the overall sys/instance admins. Entries have a label, a formal URI predicate (which together form the context entry. If a URI is not given, a unique URI is generated), and a type which controls the input widget used for entry (but not display). Some types connect to a 'third party' earth cube geosemantics server to retrieve controlled vocabulary lists or to invoke a geo name to coordinates mapping service, etc. Simple internal types include string and date.
- Metadata definitions are separate docs associated with a space

Updated Design

- Metadata from extractors is not affected - this metadata is entered, stored, and displayed as before, with the exceptions that 1) it is pulled into a separate section (name configurable - Managed Metadata) by default) from the semantic metadata entered by users through the GUI or API, and 2) it starts collapsed rather than expanded. As in the original design, extractor metadata can have a context document, but the metadata itself is treated as json (the json-ld verification performed only checks that part of the message is json-ld but stores any plain json as well). The context provided is not used for display or interpreting the metadata.
- For semantic/json-ld metadata entered via the GUI or through a semantic extension of the metadata api, the primary artifact is a json doc of the current metadata values referencing a space json-ld context. Each 'metadata summary' doc refers to the dataset/folder/file it describes by ResourceReference (type + id). CRUD operations from the GUI or API update this document via the metadata service. A history section of this summary document captures updates (documenting not only new adds, but edits, deletes as well) - it could be managed as a separate document at some point.
- Context/definitions are maintained, one per space, as part of a doc containing all current terms (each term has label/URI and definition, type, and a flag indicating whether it is available in the add metadata menu. Additional info about any cardinality, recommended/required info, etc. could be added)
- The metadata api is extended to allow a 'content' entry which works as before (e.g. as used by extractors through pyclowder) or a content_ld entry which must provide a valid json-ld context for all submitted terms. Terms that match existing entries in the space's definitions are recorded using the labels from the definitions, regardless of what label is used in the api. Terms not in the space definitions are added to the space but flagged to not show up as new options in the 'add metadata' part of the GUI. This allows any terms to be entered while still allowing space admins to control what can be added via the GUI.



Metadata display showing widget for input (date) and display (a single line display of the scientific variable term and unit from the SAS service), and the edit /delete icons for term currently hovered over. Adding/updating/deleting entries will dynamically update the page display without a reload.



The provenance of Audience entries. Hovering over an edit or delete will show the add or prior edit that it acted upon.

- Metadata is displayed by term, with the label showing, and the semantic term available by popup. Individual entries are shown within the section for that label/term with edit and delete icons. The overall section has one entry showing the creation or last modified date as a link. Clicking the link shows a popup with the detailed provenance of each entry (showing all adds/edits/deletes by person and date, with highlights showing the provenance of which edits/deletes correspond to which adds)
- The 'add metadata' section has been updated to allow edits of existing values. The widgets for different types (string, list, dynamic list, scientific variable, date/time, location wkt, and (for SEAD) person) are invoked for add/edit and, as appropriate, display of the entered values (strings just show as strings, but locations, for example show in single line form rather than as a block of json.) Stored formats for some have been changed to better support round trips through editing. Displays are forgiving - if, for example, the api, which is not type-aware, is used to enter a value that does not match the expected form, the raw value will be shown (the people widget is an example - if the value is not recognized as an ID, it is just shown as a string. If it is recognized, it is shown as a name with email popup and link to the person's profile document).
- Each metadata entry is given an id - an attempt to edit or delete an entry that has already been edited/deleted (e.g. from a stale webpage) will fail.

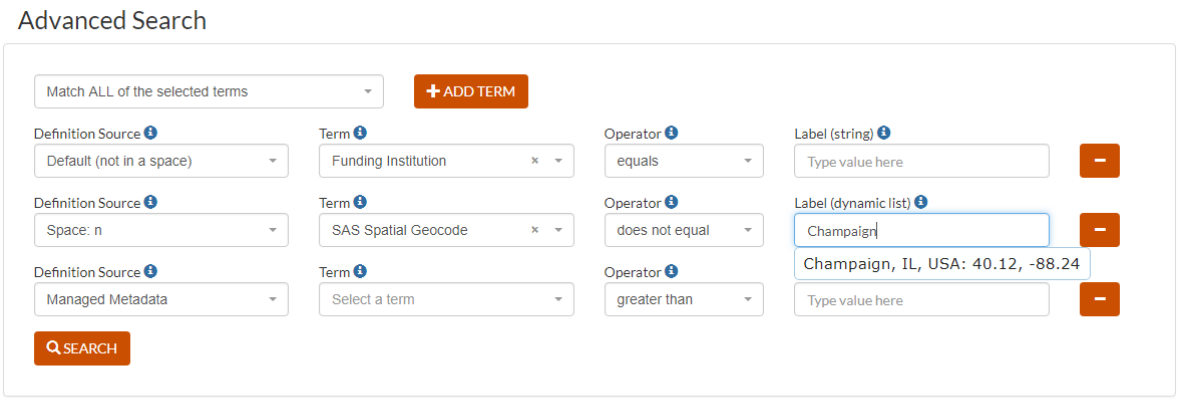
Metadata Terms & Definitions

The following metadata terms are defined within this Space. To add a new term scroll to the bottom of the page.

Label	Description	Formal URI	Type	Allow New Entries	Actions
Abstract	abstruse	http://purl.org/dc/terms/abstract	String	☐	Edit / Delete
Alternative2;Title	what else would you call it?	http://purl.org/dc/terms/alternative	String	☑	Edit / Delete
Audience		http://purl.org/dc/terms/audience	String	☑	Edit / Delete
CSDMS Standard Name		http://csdms.colorado.edu/wiki/CSN_Searchable_List	List ("http://ecgs.ncsa.illinois.edu/gsis/CSN")	☑	Edit / Delete

Updated metadata definitions table showing sorting and info about columns, condensed type info for lists derived from services, and the 'allow entries' column. In this example, 'Abstract' is no longer an option in the 'add metadata' list for this space, but existing abstract entries have not been deleted.

- The space metadata definition display has been updated to compress information about any connection to an external service and to add a column for a flag that indicates whether a term should be shown in the 'add metadata' options in the GUI. Deselecting that flag stops it from being added through the GUI, but keeps any existing annotations using it. Deleting a term brings up a warning box and, if delete is confirmed, will remove any annotations in the space using it. Editing/adding new terms will now block any attempts to create conflits where two terms have the same label or same formal URI. FOr SEAD9with the StagingArea plugin), the labels and terms used to publish non-metadata fields (e.g. fields in the dataset and file models) are also blacklisted. Edits that change a label update summary docs where that label is used (using a db query). The table is also sortable now, allowing, for example, terms from Dublin Core to be grouped together.



Updated search interface showing the addition of a source (a space, no space (using the defaults for the clowder instance) or 'Managed Metadata' - the json terms generated by extractors.) Also shown is how widgets are used for terms that are, for example from dynamic lists. Queries from json-ld metadata use the formal URIs rather than labels and will find semantically matching entries across spaces. "Managed Metadata" is a configurable term, which could be changed, for example, to "Extracted Information". Typing in the term box for this entry will pull up matching terms indexed in Elastic Search as before.

- The search interface and mechanism have been updated to better separate extractor/managed metadata (works as before) and semantic labels /terms. To do that, users indicate which space should be the source of the metadata definitions used in a query term and only labels from that space are available as options. The query itself is done using the formal URI, so searches will work across spaces, finidng entries that match for the semantic URI, regardless of what label is used in that space. The place to enter values to search for has been updated to use the same widgets as for metadata entry, i.e. a calendar will display for dates, list types will only display the available list options instead of allowing free text entry, etc.

- The sharing plugin is supported by defining one space as the 'context' space for each entry - the metadata definitions form that space are used to display the resource's metadata.
- Any operations that move a dataset or file between spaces (or a change in the context space when the sharing plugin is used and a dataset is in multiple spaces but is removed from its context space), is treated analogously to when the API is used: terms that match definitions in the new space are updated to use the appropriate labels,any definitions that do not exist in the new space are added but flag to not show in the new space's add metadata' fields.
- For SEAD, publication requests and files do not change space when the dataset they refer to moves - they remain in the staging area for that space and keep the original space as their context for publication. (They do still show as publication requests for the Dataset on its page.)
- Because the semantic support is through an extension of the api, extractors could write json-ld entries that would appear as user metadata instead of part of the managed/extracted metadata. This would allow such entries to be published (extracted metadata is nominally redundant and is not included in SEAD's publications) and would allow user editing. This might be appropriate, for example, for a face detection extractor - the faces are not simply inferred from the data format/embedded metadata like image properties are, and the conclusions as to who is in the photo could be incorrect and need to be editable by users. Whether to do this is up to the extractor and/or decisions about what to include in pyclowder.

- Metadata counting is updated - extractor entries will continue to count as one per submission, regardless of the number of terms, while semantic user metadata will count the number of individual entries and account properly for edits (no change to count) and deletes (-1). The metadata count display is the sum of these two metadata types, but could be separated into two counters. (Since duplicate deletes are caught, it should no longer be the case that hitting the delete button more than once can artificially decrement the count.)
- Other bug fixes including one that forced all location entries to be labelled "GeoJSON" regardless of the label(s) defined in the space for locations and one that dropped all but the last submission for a given metadata term for files during the publication process. Other minor fix/updates are captured in commit comments.

Status

As of Aug 15, the branch is up-to-date with the dev branch and includes all of the functionality described above. It passes 80 tests (3 tests for add /get metadata apis on datasets and files, which I believe are deprecated, have been removed since the underlying methods, which use the old metadata model, have been removed.). A working test server is available at http://141.211.23.234/ for anyone who'd like to sign up. At present, metadata summary documents are generated on-demand when a page is viewed. This could be moved to a database updater. A new collection - metadatasummary - is created for the new metadata model. At present, existing user metadata entries are not removed, which means that a given clowder instance could be ~rolled back to its current state by switching back to the dev branch software and removing this collection. (Adds/edits /deletes made using the new model will be deleted. New metadata definitions will remain, and will have the new flag, which will be ignored by the dev-branch code.) To accept this new code, a decision needs to be made about whether to process existing user metadata up front, in a database update, or dynamically, and to remove the old metadata docs and context entries that are replaced (nominally they can remain but they won't be used).

Comparison:

With goals and design ideas: The above design as implemented is consistent with the ideas outlined below (from the original discussion) though support for cardinality and selecting definitions from external vocabularies have not been implemented. (They should be straight forward to add).

Design Goals:

The redesign retains the benefits of the current design (e.g. ability to add custom terms to a space, provenance regarding the origin of specific annotations, ability to add widgets for input/display, send notices of changes to the message bus) and supports:

- organize metadata entries by vocabulary (future) and/or alphabetically and/or by purpose (future)
- clearly distinguish extracted (and therefore ~redundant) metadata from provided values
- allow some extracted terms to be propagated through publishing (right now only provided metadata is published, but space admins or repositories may want to keep high value/hard to reproduce extracted metadata as well)
- emphasize the term/definition/value for metadata and de-emphasize/hide by default the provenance of who entered it when
- enable metadata values to be edited
- deletions/edits of metadata should be captured in provenance
- manage JSON-LD context(s) at the space level (i.e. assuring that all instances of a label refer to the same predicate, all predicates have one label, even as space admins make changes, and handling metadata entered via the API if/when it has conflicting context info)
- support the exchange of contexts/metadata 'profiles' between spaces/ the use of metadata profiles from one space in a read-only manner from another (i.e. one space controls the label/URL/type/definition info)
- support (entry/display) for structured metadata values (e.g. json/json-ld)
- support consistency of metadata for all publications in a space (for a given space config)
- enable blacklisting of predicates/labels used internally in the software (e.g. for title, description, owner, dates, license, etc.)

and should simplify future enhancements such as:

- support metadata that references external URLs/identifiers and/or internal identifiers (e.g. with Collections/Datasets/Folders/Files)
- add some or all terms from existing third-party vocabularies
- allow metadata to be marked as recommended and/or indicate which metadata will be required during publishing, specified directly by space admins and/or based on available/preferred/default registered repositories)
- support cardinality constraints (number, order?)

Design Ideas

- Metadata mgmt GUI would edit context/definition doc, doc would include any new fields such as term definition, cardinality, etc. as needed. JSON-LD context would be generated from this doc when needed
- Metadata entry would use this doc from the global instance or the space the entry is in to populate the add metadata box. (with sharing on, a decision would be needed as to whether to allow annotation based on one spaces info or to allow annotation in terms of other spaces). Type/definition, cardinality and other info would be used to enhance the GUI.
- Display of existing terms would be changed ~per the GUi redesign and mockups done by Michael Iannaccone (see SEAD-1042 and https://opensource.ncsa.illinois.edu/jira/secure/attachment/21290/21290_metadata-GUI.jpg) - presenting metadata values by term alphabetically or by namespace/category, with a widget to expand the provenance of the entry (which would include any trail of edits).
- Edits to a metadata definition would tie to entries via the formal predicate, e.g. for an entry for 'method'/http://example.org/method, a change of the label for that term to "Experimental Method" would show in the display, whereas the attempt to create a new term "method": http:otherorg/method" would either be blocked due to the label conflict until the original label was updated or would trigger a confirm that the existing metadata should be updated.
- Changes to the space metadata definitions would not be tracked (could be tracked via update docs as with metadata values)
- Metadata added through the API would be matched to existing terms (required to be json-ld, the predicate would be matched to the space context and that label used (i.e. the api would not generate an alternate label for existing space definitions). New terms seen in

incoming API requests would be added to the context doc as 'view-only' terms - this allows the API and extractors to add any terms they use. Admins could add additional info (a definition, type, etc. to such generated definitions and/or allow them to appear as add choices in the GUI.
    - Movement of items between spaces would be handled as though the API had been used to enter the metadata as part of the transfer assuming no space sharing. With sharing on, the decision of how to handle annotation would have to handle being removed from one of the spaces)
    - Metadata edits would invoke the same type-specific entry widget as adding a new value
    - Cardinality of 1 would remove an option from the add menu if a value exists
    - A new type of metadata would be a link/relationship and there would be a widget(s) that would recognize URLs and internal IDs/ID URLs and make them live links (URL might just become live, but a urn:<id> value would get replaced by a dataset name or path and be a link to the page for that item. Other GUIs could be added (e.g. once bulk ops exist, one could select 2 items and then create a link between them, etc.) - these could populate the same metadata field, i.e. relationships would not be a separate type/managed separately. (It would be possible to do the lookup from Clowder URN to item via javascript, which would enable them to be displayed by name in other apps as well if desired - this is what is done for ORCID IDs where we share a javascript across Clowder and repositories.)
    - Required/recommended metadata would be identified by color/icon in the add metadata list (perhaps required fields could be shown with empty values and could be edited)
    - Metadata mgmt GUI could be extended to allow viewing/selection from other vocabs (e.g. Dublin Core, from an external website (in their format or a json conversion we or geosemantic server maintain), or another space profile (e.g. reading that space's context/definition doc and listing the entries for selection - perhaps sortable by label/URI/type/or category, etc.). Once selected, there would be no live link to the original vocab/profile.

Pros

This design builds on the strengths of the existing one in terms of maintaining (and improving provenance), supporting typed metadata, and enabling space-level customization. Many of the GUI-level changes are design agnostic and could probably be used/adapted for other storage designs. However, changing the low-level design to focus on a single space-level context/config artifact and pre-integrating the metadata for a given item, provides a clear mechanism to handle edits, context differences between sources (different extractors, different spaces, changes in space config over time) that will avoid confusion going forward - cases where different entries for the same predicate get shown with different labels, where a label is show twice because it is used for two different predicates, where it would be unclear whether type or cardinality rules only apply to the correct label/predicate combo, matching labels, or matching predicates regardless of label. It would also clean-up these types of issues for publication, and would make it easier to share the context/config of one space with another. Aside from changing how information is stored, which would be best to do early, most of the other functionality could be pursued incrementally.

Cons

In some sense, the benefits of this redesign only accrue to projects that actually use the flexibility and customizability of the system. For instances with a single space or no customization of metadata per space, where all of the metadata comes from the GUI rather than the API, where metadata is added by an authoritative user and isn't edited, etc., the current design is OK - the issues pointed out above could happen, but probably won't in practice. (Over time, and with more independent groups using an instance, these issues can and do become real though.). Some groups may thus see this design as overkill/premature. Aside from the added work to implement it, it's not clear that there are big cons. Adding a new metadata value means editing a document rather than just storing a new one, but retrieving the existing values would be one document recall rather than a scan across separate docs. Extracted metadata could be managed (with the space determining the label and/or enforcing type/cardinality rules) which would make the interface look more consistent but raises the issue of what happens if extracted metadata doesn't meet constraints? (Similar for other uses of the API.) Most of these are relatively neutral, but they make it clear that this design would have impacts outside the core of user-entered metadata. To be fair, once spaces have access control and customizable metadata, the issues of how extractors and api users handle those customizations, and how features such as being able to share datasets across spaces interact with these features (if two spaces have different metadata terms, which terms show up in the add metadata list? is it affected by whether I have access to one space or the other? what happens for extractors that work on behalf of a user through their key (a capability in development I think), etc.). I think this design can help answer those questions over time, but it does favor/flavor the answers towards a space-centric view of management/control.

Other design ideas:

GUI-centric

Leave the current metadata and context storage ~as is and focus in the GUI changes:

- info about the annotations of a given object would occur as they do now, but there would be more logic to organize the display alphabetically, etc.
- metadata management page could be updated to support new types, cardinality, etc. Enforcing cardinality would require tracking existing entries (and deciding what constitutes a match)
- editing would require some update to the model (is it delete and add? what happens when a user wants to edit one value and the original entry contains several (the API allows entry of several terms in one call which is stored as one doc))

Pros

potentially less work up front

Cons

Would still leave questions of consistent labels as metadata is edited open unless some design changes are made (e.g. when a label is updated, do exiting entries get scanned and changed? are they changed dynamically for display?) - Similarly, a means of tracking deletes is required if edit is delete/add and/or if we want to track provenance for deletes. And, if deletes are tracked, assembling the set of metadata to display involves playing the sequence of operations forward. One can keep going through all of the other requires/desirable features above and ask similar questions and ask whether this approach, by the time it addresses those requirements, remains simpler/easier/faster to implement than the proposed design. I.e., it's not clear that this design really avoids any issues in the proposed one - if we want the functionality listed some decisions and changes are needed and the question really becomes whether it is easier to implement given this underlying storage/service design.