Browse visualisation guide for publishers

Background

The Research Vocabularies Australia Portal can present a browse visualization of vocabulary data that has been described in SKOS format. Such vocabulary data can be complex; in particular, the SKOS model of concept schemes, collections, and concepts is very flexible, admitting cycles, polyhierarchies, nested collections, etc.

A key observation that has driven design and development to this point is that *almost all of the vocabulary data published in RVA is tree-structured*. We do have some vocabularies that have polyhierarchies, but they are still "tree-like", and it is *meaningful* to present such data in a tree-like way.

In fact, the visualization is of a *forest*, not a *tree*. The use of the term "tree" is historical: the initial implementation was known as the "browse tree". The following description may occasionally blur the distinction between tree and forest.

Things that have to be taken into account

Hierarchies

Certain SKOS relationships either directly state, or imply, a hierarchical relationship between resources.

- Two concepts may be explicitly related in a hierarchy using SKOS broader and/or narrower properties.
- Two concepts may be implicitly related in a hierarchy using SKOS broader and/or narrower properties that involve other concepts.
- A concept may be explicitly related to a concept scheme using SKOS hasTopConcept, topConceptOf, and inScheme properties.
- The broader/narrower hierarchy of concepts is orthogonal to membership of concept schemes. For example, suppose that concept A is broader than concept B, and A is a member of concept scheme C. That does *not* imply that B is also a member of concept scheme C.
- A concept may be explicitly related to a collection using SKOS member and memberList properties.
- A concept may be implicitly related to a collection using SKOS member properties that involve other collections. (I.e., collections may be nested.)
- Two collections may be explicitly related in a hierarchy using the SKOS member property.

Non-hierarchies

- All concept schemes are siblings of each other. (Well, maybe. The SKOS Reference provides that any type of resource can be "in" a concept scheme, and so, at least "in theory", a concept scheme can be "in" another concept scheme. The implementation does not cater for such possibilities.)
- Every collection that is not also a member of another collection is a sibling of every other such collection.
- Two concepts may be siblings by virtue of both being top concepts of the same concept scheme.
- Two concepts may be siblings by virtue of both being narrower than a third concept.
- Two concepts may be siblings by virtue of both being direct members of the same collection (i.e., without nesting of collections being involved).
- In some cases, there is an explicit order of siblings. (Ordered collections.)
- In some cases, there is an implicit order of siblings. (Lists of things that are presented to the visualisation user sorted in a convenient order.)
- In some cases, the order in which siblings are displayed in the Portal may be changed by the Portal user.

Gotchas

- By default, PoolParty doesn't add the `skos:inScheme` property to concepts. The consequence is that as one creates and populates a concept scheme using the user interface, *only* the top concepts are marked as belonging to the containing concept scheme (i.e., using the `skos:topConceptOf` and `skos:hasTopConcept` properties.). In order to make use of the functionality to display SKOS concept schemes as described here, it's necessary to configure each PoolParty project to add the `skos:inScheme` property to concepts. To do this, open the project in PoolParty, use the "Advanced" menu to select "In Scheme Settings" "Enable In Scheme". Then, again in the "Advanced" menu, turn on "In Scheme Settings" "Add In Scheme Relations to All Concepts", and also enable "Add In Scheme Relations On Concept Creation" and "Update In Scheme Relations On Concept Changes".

Deep dive

The rest of this document takes a "deep dive" into the implementation of the browse visualisation. For the most part, the implementation tries to do "what you expect". If you want to know why the implementation behaves the way it does, this description may provide the answers. For further clarification, please contact services@ardc.edu.au.

Terminology

We start with some terminology drawn from the Vocabulary Registry Schema.

- Each version of a vocabulary has a set of associated browse flags. These are settings made by a vocabulary publisher either using switches on the version dialog, or by specifying them in Vocabulary Registry data sent to the Vocabulary Registry API.
- The browse flags relevant here are of Boolean type, that is, their values are treated as either true or false. When we say that a flag is "set", we mean it has the value true.
- The flags relevant here are:
  - `includeConceptSchemes`: this corresponds to the value of the "Display SKOS concept schemes" setting; true if the setting is "Yes"; false if "No".
  - `includeCollections`: this corresponds to the value of the "Display SKOS collections" setting; true if the setting is "Yes"; false if "No".

The following terminology related to vocabulary data and its presentation comes up in various places in the rest of this document. These definitions are exclusive in this sense: a statement of the form "we say that X is Y if Z" is to be read as "we say that X is Y *if and only if* Z".

- By "a resource type of interest" we mean a SKOS class that is of interest to the processing algorithms. The class `skos:Concept` is always a resource type of interest. If the `includeConceptSchemes` browse flag is set, then `skos:ConceptScheme` is also a resource type of interest. If the `includeCollections` browse flag is set, then `skos:Collection` and `skos:OrderedCollection` are also resource types of interest.
- By "a resource of interest" we mean an RDF resource that is either identified by a URI reference or by a blank node, occurring in the vocabulary data, which the processing algorithms consider determine *could be* of a resource type of interest. (Subsequent processing may show that the resource can be ignored.)
- By "the top concept properties" we mean the `skos:hasTopConcept` and `skos:topConceptOf` properties.
- We say that *concept A is marked as broader than concept B* if the vocabulary data contains either a triple `B skos:broader A` or a triple `A skos:narrower B`. The vocabulary data does not have to contain *both* triples; if only one is present, the implementation automatically infers the other.
- We say that *concept A is marked as a top concept of a concept scheme CS* if the vocabulary data contains either a triple `CS skos:hasTopConcept A` or a triple `A skos:topConceptOf CS`. The vocabulary data does not have to contain *both* triples; if only one is present, the implementation automatically infers the other.
- We say that *concept A is marked as belonging to a concept scheme CS* if either A is marked as a top concept of CS, or the vocabulary data contains a triple `A skos:inScheme CS`. (SKOS S7: "`skos:topConceptOf` is a sub-property of `skos:inScheme`.")
- We say that *concept A is marked as directly belonging to a collection C* if there is a triple, or set of triples, which express A's direct membership of C. This can be done in these ways:
  - If C is an unordered collection, and there is a triple `C skos:member A`.
  - If C is an ordered collection, and there is a triple `C skos:memberList M`, where M is an `rdf:List`, and A is a member of M.
- We say that *concept A is marked as belonging to a collection C* if there is a triple, or set of triples, which express A's membership of C, either directly or recursively. This can be done in these ways:
  - If A is marked as directly belonging to C.
  - Recursively, via collection membership:
    - If C is an unordered collection, and there is a triple `C skos:member D`, where D is an unordered collection, and concept A is marked as belonging to D.
    - If C is an ordered collection, and there is a triple `C skos:memberList M`, where M is an `rdf:List`, and A is marked as belonging to a collection member of M.
- The terms *parent*, *child*, *ancestor*, and *descendant* apply to the visualization's presentation of the forest structure. The visualization is displayed as a forest with edges from parents to their children. (The various different contexts of nodes lead to various semantics of the parent/child relationship. For example, if both a parent and its child are concepts, the parent/child relationship represents the SKOS broader/narrower hierarchy. If the parent is a collection, the parent/child relationship represents collection membership.)

Exclusions

Both the RDF model and the SKOS model are flexible to a ridiculous degree. For example, an RDF list may have *any number* of "first" elements, e.g., zero, or three. We could try to support all of the possibilities, but we have instead decided to take a pragmatic approach that supports what we expect that people will actually use, including, in particular, the behaviour of PoolParty, and excludes as many pathological cases as possible.

These possibilities are excluded:

- RDF Lists used as the object of the `skos:memberList` property, in which there is not exactly one first element.
- RDF Lists used as the object of the `skos:memberList` property, that have members that are themselves RDF Lists.
- Cycles of the `skos:broader`/`skos:narrower` relations. We reject the entire vocabulary if we find such a cycle.
- If the `includeConceptSchemes` browse flag is set:
  - Concept scheme members that are concept schemes or collections.
  - For each concept scheme, there may not be a top concept of that concept scheme that has a broader concept that is also in the same concept scheme. We have this as an exclusion by permission of the SKOS Reference, section 4.6.3: "An application may reject such data but is not required to." NB: this is different from what we call the "classic" behaviour of the browse visualization, in which the top concept properties are ignored, and the hierarchy is displayed based solely on the broader/narrower properties.
- If the `includeCollections` browse flag is set:

    - Cycles of collection membership. (Collections may be nested to an arbitrary depth, and membership may induce a polyhierarchy, but the nesting must *not* induce a cycle.)
    - An ordered collection that has values specified using `skos:member` that are not also included in the collection's `skos:memberList`. An ordered collection must specify its members using the `skos:memberList` property. It may *also* specify those values using `skos:member`, since SKOS S36 says: "For any resource, every item in the list given as the value of the `skos:memberList property` is also a value of the `skos:member property`."
- Type inference based on the presence of triples with the `rdf:subClassOf` predicate, and the defined domain/range of user-defined predicates.
    - (But there *is* some type inference at the level of individual triples based on the rules stated in the SKOS Reference. For example, from the triple `my:C1 skos:broader my:C2`, the two additional triples `my:C1 a skos:Concept` and `my:C2 a skos:Concept` are inferred.)

How things should be

This section describes, using a question/answer format, what a user can expect of the behaviour of the visualization.

- Under what circumstances should a concept C appear at the top level of the visualization?
    - There are a couple of "big ideas" or guiding principles:
        - A concept C is shown at the top level if it would otherwise not be shown anywhere else in the visualization, i.e., as a descendant of some other node (whether that node is another concept, or a concept scheme, or a collection).
        - We want to show as much of the broader/narrower hierarchy information in the visualization as we can. If that might otherwise be "lost" or "hidden" (e.g., through the setting of the `includeCollections` flag and the fact that concepts in a broader/narrower hierarchy may also be members of collections), then a concept C *may* also be shown at the top level to enable this information to be plain to the user. (The exact circumstances under which it will be shown at the top level are explained below.)
    - We distinguish four cases based on the values of the `includeConceptSchemes` and `includeCollections` browse flags.
    - If `includeConceptSchemes` is false, and `includeCollections` is false.
        - This is the "classic", default case. No attention is paid to the `skos:inScheme` or top concept properties, either their presence or absence. The concept C appears at the top level if and only if there does not exist any other concept marked as broader. NB: this means that a concept marked as a top concept does not *necessarily* appear at the top level: it won't, if it has a broader concept, a situation explicitly permitted by the SKOS model. (See the SKOS Reference, section 4.6.3 and Example 8. Note that the exclusion mentioned in the Exclusions section above doesn't apply in this case, i.e., because `includeConceptSchemes` is false.)
    - If `includeConceptSchemes` is true, and `includeCollections` is false.
        - If the concept C is marked as belonging to *any* concept scheme, then the concept will *not* appear at the top level. (The concept schemes will appear at the top level, and the concept will appear (somewhere) as a descendant of every concept scheme to which it is marked as belonging.)
        - If the concept C is not marked as belonging to any concept scheme, then a situation similar to the "classic", default case above applies: the concept C will appear at the top level if and only if there does not exist any other concept marked as broader and which is also not marked as belonging to any concept scheme.
    - If `includeConceptSchemes` is false, and `includeCollections` is true.
        - This is like, but not the same as, the "classic", default case above. Hierarchy information from the broader/narrower properties is reflected in the visualization, and no attention is paid to the inScheme or top concept properties. The concept C appears at the top level if and only if *either* of the following applies:
            - The concept C is marked as a broader concept than another concept, and there does not exist any other concept marked as broader than concept C. NB: this means that a concept marked as a top concept does not *necessarily* appear at the top level: it won't, if it has a broader concept, a situation explicitly permitted by the SKOS model. (See the SKOS Reference, section 4.6.3 and Example 8. Note that the exclusion mentioned in the Exclusions section above doesn't apply in this case, i.e., because `includeConceptSchemes` is false.)
            - The concept C is not marked as a broader concept than another concept, nor is any other concept marked as a broader concept than C, and C is not marked as belonging to any collection. In other words, concept C is shown at the top level if it is completely "isolated" from all other concepts and from all collections.
        - (If the concept C is marked as belonging to a collection, then the concept will appear (somewhere) as a descendant of the collection. Concept C's membership of a collection prevents its appearance at the top level *in the case that* concept C is not *also* involved in a broader/narrower hierarchy, as explained in the previous point.)
        - (Does this mean that if there are collections, and all concepts are members of at least one collection, then a concept will nevertheless always be shown at least twice, e.g., by traversal from a concept at the top level, and as a descendant of a collection? No, not necessarily. In this case, if there is no use of the broader/narrower hierarchy, then all concepts will appear *only* as descendants of collections.)
    - If `includeConceptSchemes` is true, and `includeCollections` is true.
        - If the concept C is marked as belonging to *any* concept scheme, then the concept will *not* appear at the top level. (The concept schemes will appear at the top level, and the concept will appear (somewhere) as a descendant of every concept scheme to which it is marked as belonging.)
        - If the concept C is not marked as belonging to any concept scheme, and it *is not* marked as belonging to any collection, then a modification of the "classic" first case applies: the concept C appears at the top level if and only if there does not exist any other concept marked as broader and which is also not marked as belonging to any concept scheme. (In this case, concept C might or might not be marked as being broader than other concepts, which might or might not be marked as belonging to any concept schemes, but this plays no part in the determination of whether or not concept C is shown at the top level.)

        - If the concept C is not marked as belonging to any concept scheme, but it *is* marked as belonging to any collection, then a different modification of the "classic" first case applies: the concept C appears at the top level if and only if *both* of the following conditions apply:
            - There does not exist any other concept marked as broader and which is also not marked as belonging to any concept scheme.
            - The concept C *is* marked as a broader concept of any other concept which is not marked as belonging to any concept scheme. In other words, concept C will not appear at the top level if it can be "swallowed up" by a collection, without any loss of broader/narrower hierarchy information that would be displayed below a concept scheme node. (Oops, yes, we will lose broader/narrower hierarchy information in the case that C has one or more narrower concepts, each of which is marked as belonging to any concept scheme.)
        - (If the concept C is marked as belonging to a collection, then the concept will appear (somewhere) as a descendant of the collection. Concept C's membership of a collection prevents its appearance at the top level *in the case that* concept C is not *also* involved in the broader/narrower hierarchy, as explained in the previous point.)
- Under what circumstances should a concept scheme CS appear at the top level of the visualization?
    - The concept scheme CS appears at the top level of the visualization if and only if `includeConceptSchemes` is true.
- Under what circumstances should a collection CO appear at the top level of the visualization?
    - The collection CO appears at the top level of the visualization if and only if `includeCollections` is true, and CO is not itself marked as belonging to any collection.
- If `includeConceptSchemes` is true, under what circumstances should a concept C appear as a direct descendant of a concept scheme CS?
    - The "big ideas" for this are:
        - The concepts that appear as descendants of CS are precisely those concepts that are marked as belonging to CS.
        - Only broader/narrower relations between pairs of concepts that are both in CS are considered when determining the direct descendants. See the section "The graph of resources and relations" for the description of the edge-colouring of the graph of broader/narrower relations.
    - For every concept C:
        - If concept C is not marked as belonging to CS, then C will not appear as a direct descendant of CS.
        - If concept C is marked as a top concept of CS, then C will appear as a direct descendant of CS. (Reminder: in this case, there can not be a concept B marked as belonging to CS which is also marked as broader than C; see the section "Exclusions" above.)
        - If concept C is marked as belonging to CS, but not marked as a top concept of CS, then a situation similar to the "classic", default case above applies: the concept C will appear as a direct descendant of CS if and only if there does not exist any other concept marked as belonging to CS and which is marked as broader than C.
- If `includeCollections` is true, under what circumstances should a resource C appear as a direct descendant of a collection CO?
    - The resources that appear as direct descendants of CO are precisely those resources that are marked as directly belonging to CO.
- If `includeConceptSchemes` is true, under what circumstances should a concept C appear as a descendant of a concept scheme CS?
    - The concept C will appear as a descendant of CS if and only if it is marked as belonging to CS.
- If `includeCollections` is true, under what circumstances should a resource R appear as a descendant of a collection CO?
    - The resource R will appear as a descendant of CO if and only if it is marked as belonging to CO.

The graph of resources and relations

The SKOS relations (`skos:broader`, `skos:narrower`, `skos:inScheme`, `skos:member`, etc.) between resources in the vocabulary induce a directed graph, in which the resources are nodes, and the relations are edges. The processing of this graph has to produce a forest. To do this, the processing uses a depth-first search to produce a spanning forest. The construction of the spanning forest starts with roots that are the resources that have been determined should appear at the top level of the visualization, consistent with the description in the previous section. The configuration of, and therefore the result of, this depth-first search depends on the settings of the `includeConceptSchemes` and `includeCollections` browse flags.

If includeConceptSchemes is true, and includeCollections is false

We construct a graph in which every edge representing the broader/narrower relation is coloured. There is a colour for every concept scheme, and a special colour *none* to represent separation from all concept schemes.

For every concept scheme CS, we consider the graph to have a directed edge that is coloured with CS's colour from CS to every concept that has been chosen to appear as a direct descendant of CS.

For every pair of concepts C1 and C2 such that both C1 and C2 are marked as belonging to a concept scheme CS, if C1 is marked as broader than C2, then we consider the graph to have a directed edge from C1 to C2 that is coloured with CS's colour.

For every pair of concepts C1 and C2 such that both C1 and C2 are not marked as belonging to any concept scheme, if C1 is marked as broader than C2, then we consider the graph to have a directed edge from C1 to C2 that is coloured with the special colour *none*.

No other edges are added *between concepts*. To be specific: for every edge in the completed graph that is between two concepts C1 and C2 and which is of colour c, it is the case that either (a) c is the colour of a collection CS, and both C1 and C2 are marked as belonging to CS, or (b) c is the special colour *none*, and both C1 and C2 are not marked as belonging to any concept scheme.

The processing of the graph is then a set of depth first searches of the edge-coloured graph, one for each colour, starting from each concept scheme, and from the concepts that are not marked as belonging to any concept scheme and which have been chosen to appear at the top level of the visualization. The result is forest which is the union of the spanning forests produced by each search.

If includeConceptSchemes is false, and includeCollections is false

In this case, we ignore membership of concept schemes. We construct a graph in which edges are *not* coloured. For every pair of concepts C1 and C2, if C1 is marked as broader than C2, then we consider the graph to have a directed edge from C1 to C2.

The processing of the graph is then one depth first search of the graph; the result is a spanning forest.

If includeCollections is true

The above subsections cover the cases in which `includeCollections` is false. If it is true, then there is a graph node for each collection CO, and for each resource R marked as directly belonging to CO, there is a directed edge from CO to R. The depth-first search includes searches starting at the nodes for each collection that has been chosen to appear at the top level of the visualization. For the purposes of cycle detection, only collection nodes are considered, as the SKOS broader/narrower hierarchy of concepts plays no part in the visualization of collections.

Sort orders

There are currently two supported sort orders. The order "Label" was originally known as "Preferred label", since it was based on the values of the `skos:prefLabel` predicate. Now, a resource's label might come from the value of the `dcterms:title` or `rdfs:label` predicate.

Sort by "Label"

Sort by "Label" means: to order resources X and Y:

- Are X and Y members of an ordered collection? If so, then sort by their order in the memberList.
- If not, then are they different resource type? If so, order by resource type.
- If not, then is one a top concept and one not? If so, put top concept before non-top concept.
- If not, then do both have a label? If so, order by label.
- If not, then does one have a label and the other not? If so, put the one with a label before the other.
- If not, then order by IRI.

Sort by "Notation"

Sort by "Notation" means: to order resources X and Y:

- Are X and Y members of an ordered collection? If so, then sort by their order in the memberList.
- If not, then are they different resource type? If so, order by resource type.
- If not, then is one a top concept and one not? If so, put top concept before non-top concept.
- If not, then do both have a notation? If so, then order by their notation.
- If not, then does one have a notation and the other not? If so, put the one with a notation before the other.
- If not, or if they both have the same notation, then fall back to the result produced by Sort by "Label".