*Fig. 1*

# 1. Use Your Vocabularies!

Markus Demleitner
*msdemlei@ari.uni-heidelberg.de*

- Vocabularies 2 is (almost) REC; it's safe to code against it now
- What and when to code against it
- Examples in Python
- Examples in Javascript

(cf. Fig. 1)

# 2. What and When?

Use vocabularies when dealing with fields controlled by them (e.g., VOTable, Datalink, VOResource, SimpleDALRegExt, VODataService) to

- Show proper labels instead of the naked terms
- Work with term hierarchies
- Validate documents

On the other hand, if you just want to execute special code when a position's frame is ICRS (say), you probably won't need the reframe vocabulary.



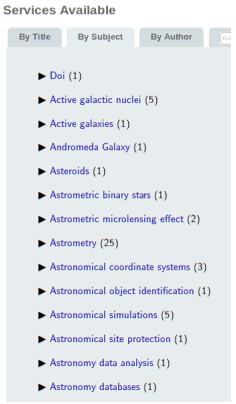


*Fig. 2*

# 3. DaCHS' Title Page

Use case: show nice labels rather than techno concept ids.

Rendering code:

```
# content is the subject metadata
try:
  uat = get_vocabulary("uat")["terms"]
  if content in uat:
    return T.a(href=
       "http://www.ivoa.net/rdf/uat#"+content)[
      uat[content]["label"]]
except (base.ReportableError, KeyError):
  # UAT retrival failure or out-of vocabulary term
  pass
return content
```

This code turns an arbitrary string into a proper link into the UAT if it's in the UAT, and it then uses the nice, human-readable label as link anchor.

Otherwise, it leaves the thing alone.

`get_vocabulary` is a function retrieving, caching, and memoizing an IVOA vocabulary. That's not big either; see the source in the DaCHS VCS[1] for how to write such a thing.

(cf. Fig. 2)

[1] `http://svn.ari.uni-heidelberg.de/svn/gavo/python/trunk/gavo/protocols/vocabularies.py`

# 4. PyVO Datalink

Use case: Return links for narrower concepts when calling, say, `bysemantics('#auxiliary')`.

Code from PyVO PR #241:

```
from pyvo.utils import vocabularies

if include_narrower:
  additional_terms = []
  voc = vocabularies.get_vocabulary("datalink/core")
  for term in semantics:
    if term in voc["terms"]:
      additional_terms.extend(voc["terms"][term]["narrower"])
  semantics = semantics+additional_terms

  semantics = set("#"+term for term in semantics)
  for record in self:
    if record.semantics in semantics:
      ... (as before) ...
```

In this code, semantics is a sequence of terms to match against; basically, for each term passed in we see if there are narrower ones in the vocabulary, and if so, add them.

PyVO, in the next release, also has vocabulary support built in as shown here: just pass a vocabulary name to `get_vocabulary`, and you'll get back a cached and memoized desise dictionary.

# 5. Adding Top-Level Keywords

Use case: fixing subjects for b2find (cf. blog post[2]):

Figure out the top-level terms in SKOS:

```
def get_roots_for(term, uat_terms):
  roots, seen = set(), set()

  def follow(t):
    wider = uat_terms[t]["wider"]
    if not wider:
      roots.add(t)
    else:
      seen.add(t)
      for wider in uat_terms[t]["wider"]:
        follow(wider)

  follow(term)
  return roots
```

This is general for the SKOS case: it will deal gracefully with cyclic graphs (and can easily be extended to only accept a defined subset of root terms).

2 https://blog.g-vo.org/semantics-cross-discipline-discovery-and-down-to-earth-code/



# 6. Tree-ifying a Datalink Response

Use case: Re-sort a datalink response so that the branches can be folded in and out (from datalink-xslt[3]).

```
function load_vocabulary() {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', "https://ivoa.net/rdf/datalink/core");
  xhr.setRequestHeader("accept", "application/x-desise+json");
...}

function _morph_table(load_event) { ...
  let dlcore = JSON.parse(load_event.currentTarget
    .responseText)["terms"];
  ...
  while (cur_term) {
    trace.push(cur_term);
    cur_term = this.vocab[cur_term].wider?.pop();
  }...
}

load_vocabulary().addEventListener("load", _morph_table);
```

This example shows how our vocabularies can be used from javascript; for one, they work nicely with XMLHttpRequests when you use their setRequestHeader method. Vocabulary loading is async in that case.

Make sure to change the vocabulary URI from http (in the official identifier) to https for javascript use, or most browsers will block your XMLHttpRequest. On the other hand, ivoa.net sets CORS headers, so the cross-origin request should work out.

The success handler then pulls the JSON out of the response event and simply parses it.

The while loop collects a trace from cur_term to a root term (which is simpler here than in the UAT example because datalink/core is a strict tree); note how conditional execution allows for nicely compact expressions here.

# 7. Concept Selector

Use case: Select an initial concept in SemBaReBro[4].

```
for (var term of Object.keys(uat)) {
  if (term.indexOf(inputSoFar)!=-1) {
    matches.push(term);
  }
}
...
for (var matchedTerm of matches) {
  var newEl = document.createElement("li");
  var uatLine = uat[matchedTerm]
  newEl.setAttribute("data", matchedTerm);
  newEl.setAttribute("title", uatLine["label"]);
  newEl.innerHTML = uatLine["label"];
  newEl.addEventListener("click", selectTerm);
  matchesBox.append(newEl);
}
```

In the first loop, we simply add terms that lexically contain the input we pull from an input field. These matches are then formatted into clickable items, which contain the term (the URI form)

3 https://github.com/msdemlei/datalink-xslt.git

4 https://dc.g-vo.org/sembarebro/q/ui/fixed

as data and show the human-readable label as text; that's simple because you can immediately pull all related metadata from the desise in the uat object.

SemBaReBro's code is available in the Heidelberg RD repo[5].

## 8. Go Forth

. . . and add semantics to your programmes. With IVOA vocabularies, it's simple and fun!
Thanks!

[5] http://svn.ari.uni-heidelberg.de/svn/gavo/hdinputs/sembarebro/