Fig. 1

Fig. 2

1. Evolving the TAP Stack

(vgl. Fig. 1)

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

(vgl. Fig. 2)

- TAP Implementation Notes
- Changes to ADQL
- Changes to UWS
- Changes to TAP

2. TAP Implementation Notes

ADQL, UWS, VOSI, TAP, and several more go into a TAP service.

During implementation and use, several issues came up that either

- need to be fixed, or
- would be handy.

TAP Implementation Notes (TIN) is an IVOA Note in progress describing quirks or new features and specifying solutions.

Current builds of the document[1] are in volute – and you're welcome to change the document there.

[1] `http://volute.googlecode.com/svn/trunk/projects/dal/tapnotes/tapnotes-fmt.html`

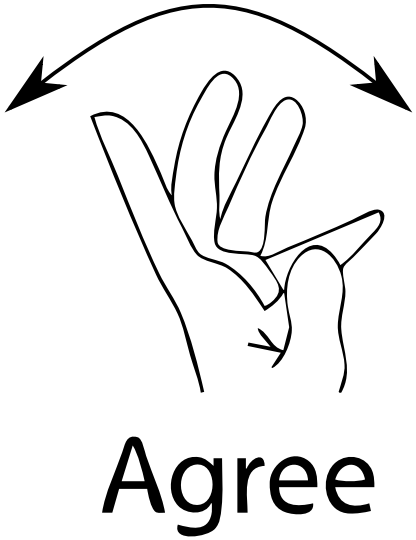


Fig. 3

3. Building Consensus

Ideally, what's published in each version of the note should reflect a consensus of (interested parts of) the DAL WG. The expectation that something like what's in TIN will eventually end up in a standard should not be unreasonable.

Now: The individual proposals.

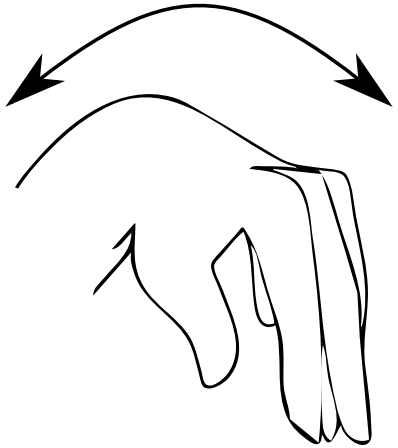
(vgl. Fig. 3)

(vgl. Fig. 4)

To help us get a feeling for what could fly and might get implemented (and what probably not), may I ask you to give the following gestures for each proposal?

For those reading this on the web: better read the document at http://volute.googlecode.com/svn/trunk/projects/dal/TAPNotes/TAPfmt.html; this also contains the reasoning behind the various proposals.

(Image Credit: New York City General Assembly, Arts & Culture WG)



Disagree

Fig. 4

4. The separator Nonterminal

The ADQL grammar as it stands mentions a *separator* nonterminal (whitespace or comment) but allows it only within a split-up character string nonterminal. Otherwise, comments are not mentioned (neither is whitespace, but that's probably implicit).

Solutions:

Either keep *separator* and basically say "Any token may be followed by a separator".

Or strike *separator* and say "Whitespace and comments can occur wherever they can occur in [std:SQL1992].".

5. Give ADQL a Type System

The ADQL spec only implicitely talks about data types (mainly via the grammar). That's been a pain for TAP, and it's a pain whenever actual data touches ADQL (mapping query results to VOTable, VOTable uploads, etc).

TAP has fixed that to some extent. TIN has a somewhat adapted table of VOTable vs. ADQL type mapping, thus defining what types ADQL should be thinking about.

Trouble: REGION is rather hard to map to at least pgSphere (that rather has Circles, Polygons, Boxes, etc).

6. First Geometry Argument

This is about

1. recommending against trying to do any magic with the reference frame argument to geometries (i.e., clients SHOULD be aware that servers will usually ignore them).
2. allowing leaving them out entirely in ADQL 2.1, i.e., make `CIRCLE(ra, dec, radius)` and friends legal – that's a minor grammar change.

7. Simple Crossmatch Function

This is about defining a simple function

`CROSSMATCH(ra1, dec1, ra2, dec2, radius) -> INTEGER`

The value is 1 if $\mathrm{dist}((\mathrm{ra1}, \mathrm{dec1}), (\mathrm{ra2}, \mathrm{dec2})) \leq \mathrm{radius}$, 0 otherwise.

8. No Decay of INTERSECTS

Right now, ADQL requires INTERSECTS with a POINT argument to decay to CONTAINS. That's the only ADQL feature you absolutely **must** do type inference for, and for that, this seems a rather minor convenience.

If we allow geometry-valued UDFs, things even become worse – so, let's drop this.

9. Geometry-valued UDFs

Grammatically, ADQL user defined functions (UDFs) can only return numeric or string values. Allowing user defined functions is a fairly small grammar change but would allow lots of interesting UDFs, e.g.,

- apply proper motion
- apply precession
- do predictable system transformation

TODO: We should come up with a library of such functions that server operators are encouraged to provide.

10. Case-Insensitive String Comparison

To allow string comparisons ignoring case, we should add

- a function `LOWER(s)`
- an operator `ILIKE`

Are there backend systems that don't already provide these natively?

11. Set Operators

`UNION`, `EXCEPT`, and `INTERSECT` are elementary operators of relational algebra – they shouldn't be missing in a relational query language.

Fixing this requires (at least) six new rules. Plus, of course, extra trouble if there are DBMSes that don't know those. Are there, really?

12. An ADQL Boolean Type

When ADQL grows a type system, a boolean type should be mentioned, and probably that `'True'` and `'False'` are the canonical boolean literals for comparsions.

We should not allow boolean column references or functions; catching typos like `WHERE ra` without type inference is more valuable than the saved keystrokes.

13. Casting to Unit

This would introduce an ADQL "function" (macro, rather) that lets you write

`IN_UNIT(raj2000, 'rad')`,

with the first argument a column reference, the second a string valid according to VOUnit. If all goes well, the ADQL translator would make a `raj2000/180.*PI` out of this if raj2000 were in deg.

[If we're fancy, we could allow an expression instead of a colref, but that would make writing ADQL translators significantly harder.]

14. UCD column references

This would allow writing things like `UCDCOL('pos.eq.ra;%')` or similar.

15. Updating UWS parameters

Right now, UWS allows several techniques to change job parameters for jobs that are not yet executing (about three of them).

Paul wants to add language that at least forbids creating parameters and say:

> From the client perspective, there is only one guaranteed way to set a parameter that all UWS services must implement: In the initial POST that creates the job.

16. Failed UWS Job Creation

In Section 2.2.3.1 of [std:UWS] a UWS is required to return a "code 303 'See other"' "unless the service rejects the request".

Can/should we specify what's to happen otherwise?

I've not put in an actual proposal yet, but I'm leaning towards allowing almost everything but saying it's nice if you produce a DALI-compatible error message with a VOTable content type.

17. Format of Quote

The quote resource (give the expected run time) is a number in seconds, in the schema (as returned by retrieving the job resource) the quote is a datetime giving the expected completion time (in a plain ISO string).

We propose to have the quote resource return a plain ISO string, too.

18. Names of Uploaded Tables

This is about upload syntax; right now, you could, by the spec, use SQL delimited identifiers (`',http://foo.bar;bla,`, say), except that the syntax of the UPLOAD parameter might blow up.

We propose to only allow names matching the *regular_identifier* ADQL symbol.

(Note that this doesn't fix the issue of upload URLs containing metacharacters of the UPLOAD parameter)

19. Multiple Upload Posts

What's supposed to happen when multiple UPLOAD parameters are given? When later POSTs contain uploads?

We propose to define uploads to be accumulating.

20. size in the TAP schema

TAP_SCHEMA.columns has a column named size, which, unfortunately, is an ADQL reserved word.

We propose to warn people and announce an upcoming renaming to arraysize in TAP 2.0.

An attractive alternative would be to unreserve SIZE in ADQL. SQL 1992 only uses it in the arcane DIAGNOSTICS SIZE argument to SET TRANSACTION. Since it's unlikely that ADQL will ever support different isolation levels and even more unlikely that it's going to have a diagnostic area, that's probably ok.

21. Error message formatting

The TAP spec currently says a QUERY_STATUS INFO element must be "before" the TABLE element. That's hard to do in error messages that have no TABLE.

We propose redactional changes to the language.

22. Sanitizing inline UPLOADs

Let's defer this to the DALI session.

23. An examples Endpoint

Talked about that in Urbana – basically, it's a proposal to have a browser-readable document under an examples endpoint next to async. There's simple additional markup to let machines parse the stuff that now happens to work as RDFa:

```
<div about="#katkatbib" typeof="Example" id="#katkatbib">
<h2 property="name">katkat bibliography</h2>
<p> To search for title (or other) words in
<a property="table" href="/tableinfo/katkat.katkat">katkat.katkat</a>'s
source field or in some other sort of bibliographic query, use the
<tt class="literal">gavo_hasword</tt> locally defined function.
[...] </p> <p>Try the following query:</p>

<pre property="query">select * from katkat.katkat
where gavo_hasword('variable', source) and minEpoch<1900 </pre>
</div>
```

24. A plan endpoint

This simply calls for a resource plan within each UWS job resource giving the actual query on the backend system and the query plan of the underlying DB in an essentially system-defined plain text format.

You'd be fine giving 404s here.

25. Scaleable tables Endpoint

The VizieR VOSI table is huge (and it's quite large for other services, too).

VODataService allows to return "empty" table elements (i.e., without column elements); this proposal defines that for those, clients can expect the full table metadata in a VODataService table element in tables/<table-name>.

Do we want to require descriptions in the top-level document? Should the second-level responses be VOSI tablesets rather than vs:Tables? What's to happen when people use delimited identifiers as table names ("obs/id")?

26. Name of the TAP Schema

If you're running more than one TAP service on top of one DBMS, the name of the TAP_SCHEMA is the main colliding resource. This proposal would allow changing it via a declaration in VOSI of the sort

```
<schema utype="ts:tap_schema">
  <name>myTapDescriptor</name>
  ...
</schema>
```

27. Phew!

:-6

So – implement what you like, improve what you don't, add what's missing.

It's all on Volute, in in projects/dal.