GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

Transitioning Between Major Standard Versions

Markus Demleitner

College Park IVOA Interop, June 2025

Talk Outline

- What's a “major version” anyway?
- Desiderata for transitioning between major version
- How we did so far: SIAP, UCD, VOTable BINARY2…
- Can we do it properly?

Breaking Changes

What’s a Major Version?

DocStd says:

> *The [major version] number increments to 1 for the first public version, and to 2, 3, . . . , for subsequent versions that are not backward compatible and/or require substantial revisions to implementations.*

Making a first-principles definition actually *is* hard. For this session, I’m suggesting:

A major version is when two things that could talk to each other before no longer can when only one is updated.

Desiderata

Desiderata

If things stop talking to each other, that's called "broken" in the VO (or the internet, for that matter).

Users don't like broken things.

Are there methods to minimise the exposure of our users to broken things when we do make breaking changes?

Let's dream: What would an ideal major version transition look like?

Desideratum: No VO splits

The VO should look the same regardless of what client is in use.

Counterexample: If a legacy client only sees version 1, another, possibly newer, client sees both version 1 and 2, and yet another client sees only version 2, then users will see entirely different sets of data collections depending on which client they currently use.

We had this big time with the pre-RegTAP Registry system (for different reasons), and it upset people quite a bit.

Desideratum: Server-Side Economy

Services should not be required to implement multiple versions of a standard.

If we required all services to have both version 1 and version 2, we'd almost be done. The Registry is built such that clients can pick and choose which version they understand and want to use with reasonable effort.

But of course that puts quite a bit of load on the service implementors, and it certainly doesn't help if the reason for the major version step is getting rid of mistakes made in the past.

Desideratum: No Flag Days

In a widely distributed system like the VO, we cannot force clients and servers to all move from one version to the next at one point in time.

If we could, we'd have a "flag day". However, the realistic half-life for updates in the VO is rather 24 months than 24 hours. 24 hours of confusion may be acceptable. 24 months is not.

Desideratum: Client-Side Economy

Clients should not be required to implement multiple major versions.

Requiring clients to support all major versions is probably a non-starter, because part of the problem is legacy software that's hard to upgrade. If we could upgrade to mixed-version client libraries, we could upgrade to the new version directly just as well.

Desideratum: Ecosystem Stability

A client written in year Y should not break because of our transition before year Y+N, where N is perhaps 5 or so.

If we actually stop supporting a given major version, we *will* break some hand-written custom code. Let's at least make it so we don't pull any rugs from underneath people who were just settling down on them.

Stable environments are important to foster the growth of software systems.

Conflicting Desiderata

When you review the desiderata, it is *almost* obvious they are conflicting:

- No VO splits
- Server-Side Economy
- No Flag Days
- Client-Side Economy
- Ecosystem Stability

Something's (almost certainly) got to give. But what?

How did we do so far?

Experiences With Breaking Changes

We’ve tried it before.

It never was pretty.

Let me tell you a few stories.

UCD1 to UCD1+

In the olden days, UCD atoms could not be concatenated. What is now pos.eq.ra;meta.main used to be POS_EQ_RA_MAIN.

Changing all the strings of course broke everything relying on UCDs.

I wasn't there at the time, but it seems there was not much turmoil (presumably there was not much UCD usage in these days).

But UCD1 was used in SCS and SIA1. *It still lives on in both places*, and both standards will break if we move them to UCD1+.

SIA1/SIA2

When we had Obscore as a good response schema for images and the like, there was strong pressure for an update to SIA.

Since the response schema needed to change, it was clear that SIA1 clients wouldn't be able to grok SIA2 responses; and we changed, perhaps somewhat wantonly, the input parameters, too.

We did not manage the transition at all. Right now, about 20% of the SIA1 services are younger than 30 months. Most data providers try to go dual-version.

SIA2 was made REC in December 2015.

VOTable BINARY2

In the early 2010s, people were tired of trying to juggle NULL values when writing VOTable BINARY tables. With VOTable 1.3, the much better BINARY2 was introduced, together with a suggestion for how clients could singal they'd like to see it.

12 years after VOTable 1.3-REC, here's a census among 130 TAP services in the VO as to their default serialisation:

tabledata	binary1	binary2
61	32	10

Winding Down

Lessons

- It's nasty. Let's not do major versions lightly.
- We'll have to compromise on the desiderata. I think it's server-side economy that will have to budge. SIA operators have done that anyway.
- We have to be clear that we want to migrate and give clear time frames ("SIA1 will no longer be an IVOA standard by 2022").
- We need a team to manage the change: talk to server operators, talk to client developers, watch the progress.

A Project?

The topic came up in the context of replacing VOTable with JSON and dropping x-www-urlencoded parameters to our protocols.

Could this be a model project for a major version transition?

What about SCS2 instead? I'd be in on that.

Also, please contribute to our brainstorming document: https://github.com/ivoa/major-version-transition