FDRI Sensor Things API

Investigation and implementation report

Dave Reynolds, Mihajlo Milosavljevic, 22 October 2025

1 Summary

This report summarizes an investigation and implementation of an OGC Sensor Things API (STA) for the FDRI data service. The motivation was to enhance data and metadata access for downstream users, encourage provenance metadata download, and facilitate data exchange, ideally based on open standards.

STA is the best fitting open standard for the FDRI data but there were concerns regarding its data model limitations, implementation complexity, and usability. This has led us to adopt a dual strategy. We are offering a partial STA implementation in parallel with a tailored, simple REST API. The STA implementation offers some standards compliance and a route to exploiting future tools that might develop in the STA ecosystem whereas the REST API gives us complete coverage of the FDRI data model in an easy to use format.

The main challenge with the STA option is that STA is sensor-centric. Datastreams are associated with individual sensors that might be relocated and the datastream is expected to follow the sensor, with each observation potentially concerning a different feature of interest. In contrast FDRI, and other sensor networks, are focussed on sites and features of interest where we want a continuous datastream for a given observed property at some site and feature of interest, even if the individual sensor might be changed. This history of sensor deployments should be available as provenance metadata but should not dictate the structure of datastreams. We handled this by mapping the STA Sensor entities to EnvironmentalMonitoringPlatforms on which physical sensors are deployed.

The STA implementation utilized the Hydroserver 2 project's Python STA implementation as a shim over existing FDRI metadata APIs and observational data stored in Amazon S3. Elements of the FDRI data model that are outside of STA are exposed using the extension *properties* field on each FDRI entity. Though some further fine tuning of this will be needed.

The STA implementation has several query limitations due to the underlying REST API's limited expressiveness, some issues with the chosen base library and the performance characteristics of querying partitioned Parquet files in S3. Specifically, complex boolean combinations, arithmetic, and many built-in functions are not supported for metadata queries. Observation queries across many datastreams are supported but constrained by performance.

Despite these limitations, the current STA implementation is sufficient for evaluation. We will maintain it alongside the REST API, seeking user feedback. Future improvements could include exposing full i-Adopt facets, richer sensor history, supporting 'or' conjunctions on observations, and improving error reporting. If proven useful, work towards standardizing an STA profile should be considered.

2 Contents

3 Motivation and background

This work is part of a series of investigations into API options for the FDRI data service that can support both data and metadata access. At a high level the requirements are:

1. Single service able to return both data and metadata
2. Support downstream data users to help them discover the datasets of interest and access the data to support visualisation and local analysis
3. Encourage data users to download the provenance metadata associated with the data of interest
4. Ease of use so that the API is not a barrier to access
5. Facilitate data exchange with other relevant institutions
6. Ideally, based on shared API standards supporting an open ecosystem of associated tools
7. Initially support timeseries data but extensible to user data types

Among the standards-based options that would particularly meet (6) the OGC Sensor Things Sensing API version 1.1[1] (henceforth, STA) stands out. It is a reasonable match to the FDRI data, it is simpler than other OGC hydrology API standards and it has some traction with the FDRI community. In particular several NERC institutions are exploring data interoperation via STA under the Amplify project,[2] and STA is showing increasing uptake in European hydrology and environmental sensing groups.

[1] https://docs.ogc.org/is/18-088/18-088.html - only the read (GET) elements are of relevance here.
[2] *Couldn't find a good reference for Amplify.*

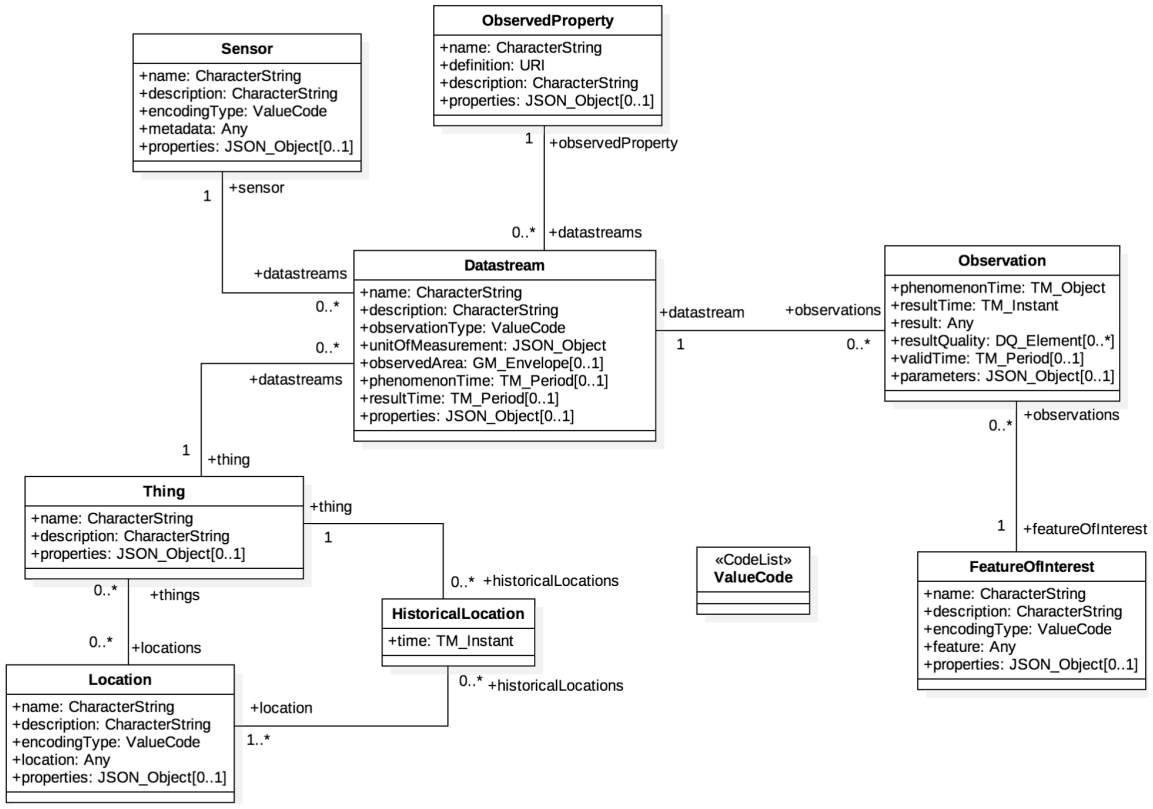
However, there were concerns over the limitations of the STA data model, the complexity of implementation and complexity of use. These concerns were amplified by feedback from some groups particularly USGS[3] who have decommissioned their STA service on grounds of performance, complexity of maintenance and steep learning curve for users.

After a number of investigations the decision was made to pursue a dual strategy in which an STA implementation is offered alongside a tailored, simple REST API. This allows us to gather feedback on the usability of STA, and to make use of other STA ecosystem tools if they emerge, but allows us to serve the full FDRI data and data model via the companion REST API without the compromises needed for STA.

4 Approach

4.1 Data model mapping

The STA data model is fairly simple and is shown below.



The main challenge with applying this data model to FDRI, and to many environmental sensing applications, is that it is based on the notion of an "Internet of Things" where a Thing is something with sensors that (potentially) moves to different locations. The datastreams are tied to the specific sensors on a Thing which might thus measure different Features of Interest at different times depending on where the Thing is deployed.

[3] https://waterdata.usgs.gov/blog/api-decom-fall-2025/

This is in contrast with FDRI which is focussed on environment sensing at specific sites (with associated features of interest) and creating datasets of long term measurements of variables (Observed Properties) at those sites. If the sensor for a given variable deployed at a site is swapped out for a new sensor then we wish to record that change as part of the provenance chain but the dataset should be stable and the timeseries should be unbroken.[4] If the old sensor is refurbished and then installed at a different site we might again wish to know that deployment history for provenance purposes but the data from the sensor is part of the record for the new site and not directly related to any measurement taken at the old site.

This disconnect makes it non-trivial to determine how to map the STA concepts of *Thing* and *Sensor* to a model like FDRI. It also has subtler impacts such as the cost of following the link from *Sensor* to *FeatureOfInterest* via *Observations*.

The mapping chosen for this implementation is:

STA Entity	FDRI Class	Comments
`Thing`	`EnvironmentalMonitoringSite`	Map descriptive properties of site
`Location`	`EnvironmentalMonitoringSite`	Map geo properties of site
`HistoricalLocation`		Not applicable, sites don't move
`Sensor`	`EnvironmentalMonitoringPlatform`	The platform on which sensors are mounted, so stable across swap of specific sensors. Platforms can be composed hierarchically and the finest grain platform should be used.
`Datastream`	`TimeSeriesDataset`	
`Observation`	data row	
`ObservedProperty`	`Variable`	
`FeatureOfInterest`		Future work

For a detailed breakdown of the property and relation mappings for these entities see code documentation.

4.2 Implementation Options

While the data model for STA is small and reasonably straightforward, the API itself is based on OASIS OData version 4. This is a complex standard to implement and implicitly assumes that queries operate over a fully featured database able to support nested expressions, arithmetic, builtin functions and arbitrary boolean combinators.

[4] Assuming both old and new sensors are correctly calibrated to some standard. The calibration actions, process and standards are all relevant metadata which should be captured and are included in the FDRI data model.

The reference implementation for STA, Fraunhofer FROST,[5] provides a full implementation but it is monolithic in design and assumes a postgresql database with its own table layout, offering no support for alternative backends nor API profiling. Maintaining a live copy of all data in a separate postgresql database via STA update was not seen as a practical option for FDRI on cost and performance grounds. The performance bottleneck of having to bulk load data to FROST only through the STA update API was part of the reason given by USGS for decommissioning their service.

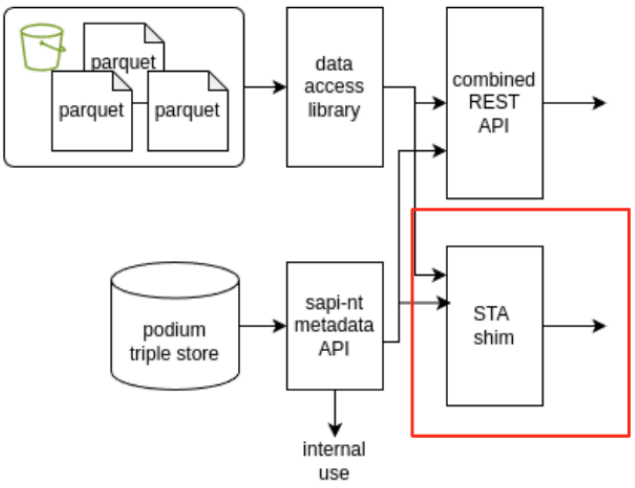
A ground up implementation is possible but there is little support for OData implementations outside of the Microsoft stack. There is some support in the form of Apache Olingo library but that does not go far enough to save much of the implementation effort.

However, recently the Utah Water Research Laboratory has released an implementation of STA in python as part of the Hydroserver 2 project. This implementation[6] is well suited to building an STA adapter for existing backends and we chose to use this to provide an STA compatibility layer over the FDRI service. We recognise that the Hydroserver implementation is not complete but it has enough coverage and, importantly, the flexibility to meet our needs.

5 Implementation

5.1 High level architecture

Our chosen approach is to offer two combined APIs from FDRI side-by-side - a REST style API fully matching the FDRI data model and an STA compatible API. Each of these runs as a shim or broker on top of an existing metadata API and tools for accessing the observational data (which is stored as parquet format in Amazon S3[7]).



Using HydroServer2-sensorthings as a framework makes this shim approach relatively straightforward.

[5] https://www.iosb.fraunhofer.de/en/projects-and-products/frost-server.html
[6] https://github.com/hydroserver2/hydroserver-sensorthings
[7] The parquet files follow a standard directory partitioning approach and are normally accessed via DuckDB.

For each metadata entity a query in the STA protocol is converted to a REST style query to the existing metadata API - translating the query to matching class endpoint and translating the STA filters to REST filters where possible.

Queries to *Observations* are mapped to a DuckDB query over the partitioned parquet files in S3.

STA queries that are chained by use of paths (e.g. `/Thing/Datastreams/ObservedProperty`) or by use of `$expand` sometimes require a series of calls to metadata API using the results of one as a filter for the next stage. Particularly when combining metadata and data queries.

5.2 Identity mapping

One key challenge in this mapping process is the handling of IDs. The FDRI model follows W3C Data on the Web best practice[8] by assigning URIs to each resource with the intention that these URIs should dereference.[9] This is compatible with a REST style API where the URI for the resource would return details of that resource and truncated URIs return lists of such resources. In contrast the STA approach is a pure API style where endpoints are abstract and the identity of a resource to retrieve is passed as a query parameter. Furthermore, the syntax rules for STA identifiers are very restrictive and do not permit URIs, nor even all characters allowed in URI stubs. We handle this by defining a mapping from FDRI URI space to legal STA identifiers which is reversible under certain assumptions:

- remove a standard base prefix: http://fdri.ceh.ac.uk/
- transform path separators / to three underscores: ___
- transform hyphens - to double underscores: __
- assume all other characters are legal

So for example the dataset resource http://fdri.ceh.ac.uk/id/dataset/cosmos-chobh-precip_30min_raw has STA identifier `id___dataset___cosmos__chobh__precip_30min_raw`.

This mapping is only safe and reversible if we assume both a restricted character set and absence of pathological patterns (such as "-_") in construction of FDRI identifiers. These assumptions are not too restrictive and are met by the current FDRI data pipeline.

5.3 Accessing extended FDRI data

The mapped FDRI classes typically have additional properties that we wish to expose. This is possible by use of the open `properties` field allowed for on each STA entity. We extract a json representation of the entire FDRI class, and some linked resources, and return this in the `properties` field of the STA entity.

For now we return the entire json representation of the mapped resource but this can be fine tuned in the future. In particular, the question of how best to expose the history of concrete FDRI sensors deployed at the platform mapped to an STA Sensor needs further work.

While this ability to deliver an arbitrary additional payload is useful, it makes the STA implementation specific to FDRI. There have been discussions on the need for a *profile* mechanism to agree common extensions, or at least enable automated discovery, but that is outside the work reported here.

[8] https://www.w3.org/TR/dwbp/
[9] Public access to the chosen base domain fdri.ceh.ac.uk is to be confirmed

5.4 Query limitations

The STA implementation is unable to support the full range of STA OData query patterns. There are several reasons for these limitations.

5.4.1 Metadata API limitations

STA assumes a rich database query language is available, with the ability to add functions to match the STA built-in query functions. Our implementation approach, a shim over a REST API, means that we are limited by the expressiveness of the REST API.

All of the query patterns `$top`, `$skip`, `$count`, `$orderby`, `$select`, `$expand` and `$filter` have an analogue in the REST API. However, the patterns available in the filter expressions are limited with no support for:

- `not`
- arbitrary boolean combinations with precedence grouping - we support `and` over different properties and `or` over values of a single property
- arithmetic - `add, sub, mul, div, mod`
- builtin functions - the string, date, maths and geospatial functions are not currently supported
- filters/counts/pagination within nested queries

5.4.2 Parquet query performance

For cost and scalability reasons the FDRI data is stored "Big Data" style as Apache Parquet files in Amazon S3 in a folder structure organised by site and date. These files are then accessed via a DuckDB query engine. This design is optimised for fast retrieval of individual time series over a bounded time period. However, STA allows users to query:

- Observations for a single observable property across multiple/all sites.
- Observations from a single site but across all observable properties.
- Observations from any number of different datastreams.
- All observations.[10]

These all require scanning multiple or all parquet files, sometimes extracting multiple columns at the same time while still supporting filtering, sorting and paging.

We explored[11] a number of query mapping approaches that could implement the STA queries with reasonable performance over parquet files. These included:

1. Separate queries for each datastream with merging, sorting and paging in python.
2. Union parquet scan returning wide rows then unpivot and sort/page in python.
3. Union parquet scan returning one column per row, push down sort and paging to the query.
4. Union parquet scan returning wide rows but pushing filtering and sorting to the query then final unpivot and page in python.

Based on our evaluations option (3) is the most scalable in terms of the API implementation, with all the work being done by the DuckDB engine. However, the size of the query grows with the number

[10] Supporting this is not a priority since it is impractical to deliver the entirety of the data this way.
[11] https://github.com/NERC-CEH/dri-sensor-things-api/blob/main/docs/observation-querying.md

of datastreams queried, becoming unmanageable when dealing with thousands or tens of thousands of datastreams.

For the current implementation we use option (3) since we anticipate common query patterns needing O(100) datastreams[12] or less. This does mean that attempts to e.g. return all datastreams at once will likely fail or timeout.

5.4.3 Hydroserver2 sensorthings limitations

A small number of limitations of the Hydrosever2 implementation have been noted.

In particular, when filtering on nested properties a user must explicitly walk down the expansion tree. For example, in order to apply this filter:

```
/Things?$filter=Datastreams/ObservedProperty/id eq
'ref___common___cop___precipitation'
```

the user has to first expand the Datastreams:

```
/Things?$expand=Datastreams($filter=ObservedProperty eq
'ref___common___cop___precipitation')
```

Additional query parameters on the second expansion are ignored. This is a limitation of the hydroserver's implementation.

Some limitations of of the Hydroserver2 sensorthings implementation were seen as blockers so we have forked the original code to address those:

- allow filters on the `properties` field
- allow multiple filter parameters in a nested `$expand` (separated by `;` as required by the OData spec)

Further extensions, such as adding text search support, could be implemented in this local fork.

6 Example query usage

We illustrate the STA implementation by following the steps in an example use case.[13]

a. Find an observed property of interest

We can list all observed properties with:

https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/ObservedProperties

Or can find one with a specific name:

https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/ObservedProperties?
$filter=name eq 'Relative humidity' 🔗 try it

[12] Queries such as requesting a single or small number of observed properties across all sites, or many properties from a single or small number of sites fall into this sweet spot.
[13] A modified version of use case 2 from FDRI API Requirements.docx, though using Relative Humidity as the example observed property to look up for better data coverage.

We can also filter on properties on the extension field such as:

https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/ObservedProperties?
$filter=properties/prefLabel eq 'Relative humidity' 🔗 try it

This will enable us to also filter on the i-Adopt facets of the (complex) observed properties once those are exposed through the base API.

This tells us the STA id of the observed property we are interested in: `ref___common___cop___rh` corresponding to a URI of `http://fdri.ceh.ac.uk/ref/common/cop/rh`

b. Find sites for which the observed property is available

In STA properties are attributes of the Datastreams, so this query has to follow that link using the expansion pattern noted above:

https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/Things?
$expand=Datastreams(
$filter=ObservedProperty eq 'ref___common___cop___rh') 🔗 try it

This indirection through Datastreams is necessary because of the STA data model. However, as we can see from the return, there are *many* Datastreams from each site so this query is lower performance and returns more data that we need.

In the FDRI data model we directly annotate sites with observed properties available from that site. We can exploit this to do a more efficient, but FDRI specific, version of the same query:

https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/Things?
$filter=properties/observes eq 'http://fdri.ceh.ac.uk/ref/common/cop/rh' 🔗 try it

c. Find the geolocations for those sites in order to display on a map

https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/Things?
$filter=properties/observes eq 'http://fdri.ceh.ac.uk/ref/common/cop/rh' &
$expand=Locations 🔗 try it

d. Find properties of the sensor for this property, such as mounting height, at specific sites

This is beyond the STA data model. In principle the mounting height would be an attribute of the EnvironmentalMonitoringPlatform and so would be available in the properties field of the STA Sensor. Though in the current FDRI sample data this is not populated and not exposed to the STA shim.

e. Retrieve data from this property from all the sites, or a set of sites, for a time period

For a single datastream from our list of relevant streams retrieval of data for a date range is easy:

https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/
Datastreams('id___dataset___cosmos__henfs__rh_30min_raw')/Observations?
$filter=phenomenonTime gt 2025-01-01T00:00:00Z
and phenomenonTime lt 2025-01-30T00:00:00Z 🔗 try it

The return does include navigation links allowing a data user to map to the source datastream and so its metadata, and provides a link for each individual result. This does lead to a large payload though.

{

```
  value: [
    {
      @iot.id: "id___dataset___cosmos__henfs__rh_30min_rawI20250127T190000",
      @iot.selfLink:
"https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/Observations('id___dataset___cosmos__henfs_
_rh_30min_rawI20250127T190000')",
      phenomenonTime: "2025-01-27T19:00:00+00:00",
      result: 79.61,
      Datastream@iot.navigationLink:
"https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/Observations('id___dataset___cosmos__henfs_
_rh_30min_rawI20250127T190000')/Datastream",
      FeatureOfInterest@iot.navigationLink:
"https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/Observations('id___dataset___cosmos__henfs_
_rh_30min_rawI20250127T190000')/FeatureOfInterest"
    },
    …
```


It is possible to select just the elements you want through `$select` so:

https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/
Datastreams('id___dataset___cosmos__henfs__rh_30min_raw')/Observations?
$filter=phenomenonTime gt 2025-01-01T00:00:00Z
and phenomenonTime lt 2025-01-30T00:00:00Z &
$select=result, phenomenonTime 🔗 try it

returns just:


```
{
  value: [
    {
      phenomenonTime: "2025-01-27T19:00:00+00:00",
      result: 79.61
    },
    {
      phenomenonTime: "2025-01-27T19:30:00+00:00",
      result: 81.9
    }, …
```


STA allows us to return all observations from a range of sites. For example, we can pick datastreams for Relative Humidity and fetch observations for those for a time span using:

https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/Datastreams?
$filter=properties/observedProperty eq 'http://fdri.ceh.ac.uk/ref/common/cop/rh' &
$top=5 &
$expand=Observations(
$filter=phenomenonTime gt 2025-01-25T00:00:00
and phenomenonTime lt 2025-01-30T00:00:00; $top=1000) 🔗 try it

Conversely we can start from Observations and filter them by the set of associated Datastreams as well as time span:

https://dri-sta.staging.eds.ceh.ac.uk/sta/v1.1/Observations?
$expand=Datastreams(
$filter=properties/observedProperty eq 'http://fdri.ceh.ac.uk/ref/common/cop/rh')&
$filter=phenomenonTime gt 2025-01-25T00:00:00Z
and phenomenonTime lt 2025-01-30T00:00:00Z 🔗 try it

This query covers all the Relative Humidity streams, not just the top 5 as above and so it is more expensive. It will sometimes time out and sometimes work and represents the boundary of what is

possible. Moving from S3 to EFS hosting of the parquet files is likely to make a significant difference for cases like this.

7 Status and next steps

Apart from the more fundamental limitations on query noted above, there are some elements of the implementation that could be improved or extended if there is interest. Specifically:

1. The ObservedProperty queries only return generic SKOS properties and do not yet include the full i-Adopt facets present in the underlying data. A small extension to the metadata API is planned to provide better access to these which can then be used for both the STA shim and the combined REST API.

2. The extension data returned in the `properties` field could be improved. For example, STA Sensors could show some part of the history of sensor deployments at the Platform and return more of the Platform metadata.

3. The `or` conjunction on data (Observations) is not yet supported.

4. There is no support for text search. This is not part of the STA specification in any case but will be needed to address discovery of observed properties, called for in several of the use cases.

5. Some of the STA filter functions might be implementable through post processing.

6. Query performance over datastreams is limited by the cost of scanning multiple files in Amazon S3, so some otherwise tractable queries timeout. It would be possible to maintain a copy of the parquet files on a faster distributed file system, such as Amazon EFS to mitigate this.

7. A number of the descriptive fields expected by STA are not populated in the current data, though provided for in the FDRI data model.

8. Error reporting and query validation by the Hydroserver2 implementation is poor with genuine parse errors reported without enough detail to guide debugging and some errors resulting in raw python stack traces. In some cases misspelling of STA properties is picked up but in other cases they are swallowed silently, resulting in unexpected returned data.

Despite these limitations, the current implementation is sufficiently complete to give a good basis for evaluating the approach.

We will maintain this initial implementation alongside the combined REST API[14] and seek user feedback as the service is opened up to end users and tool developers. No formal evaluation is currently planned.

If the STA interface proves useful then, apart from addressing some of the above limitations, consideration should be given to how to document and standardise this STA profile. Profiling here means both sub-setting (to limit the query patterns to those we can efficiently implement) and super-setting (to define the data model extensions exposed via the properties fields).

[14] Under development in dri-combined-apiwrite.