Table Access Protocol
Version 1.1

IVOA Proposed Recommendation 2019-06-26

Working group
Data Access Layer Working Group
This version
http://www.ivoa.net/documents/TAP/20190626
Latest version
http://www.ivoa.net/documents/TAP
Previous versions
PR-TAP-1.1-20181024
PR-TAP-1.1-20180830
PR-TAP-1.1-20180416
PR-TAP-1.1-20170830
WD-TAP-1.1-20170707
WD-TAP-1.1-20160428
TAP-1.0
Author(s)
Patrick Dowler, Guy Rixon, Doug Tody, Markus Demleitner
Editor(s)
Patrick Dowler
Version Control
Revision 5515, 2019-06-28 11:21:31 +0200 (Fri, 28 Jun 2019)
https://volute.g-vo.org/svn/trunk/projects/dal/TAP/TAP.tex

Abstract

The table access protocol (TAP) defines a service protocol for accessing general table data, including astronomical catalogs as well as general database tables. Access is provided for both database and table metadata as well as for actual table data. This version of the protocol includes support for multiple query languages, including queries specified using the Astronomical Data Query Language ADQL within an integrated interface. It also includes support for both synchronous and asynchronous queries. Special support is provided for spatially indexed queries using the spatial extensions in ADQL. A multi-position query capability permits queries against an arbitrarily large list of astronomical targets, providing a simple spatial cross-matching capability. More sophisticated distributed cross-matching capabilities are possible by orchestrating a distributed query across multiple TAP services.

Status of this document

This is an IVOA Proposed Recommendation made available for public review. It is appropriate to reference this document only as a recommended standard that is under review and which may be changed before it is accepted as a full Recommendation.

A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/documents/.

Contents

1 Introduction **5**
- 1.1 Role within the VO Architecture 5
- 1.2 Motivating Use Cases 7
  - 1.2.1 Discover Metadata 7
  - 1.2.2 Query Custom Tables 8
  - 1.2.3 Query Standard Tables 8
  - 1.2.4 Query Standard Data Models 8
- 1.3 Query Languages 8
  - 1.3.1 ADQL Queries 8
  - 1.3.2 Other Query Languages 9
- 1.4 Query Execution 9
  - 1.4.1 Asynchronous Queries 9

1.4.2 Synchronous Queries . . . 9

2 Resources **9**
2.1 TAP-sync . . . 10
2.2 TAP-async . . . 11
2.3 VOSI-availability . . . 12
2.4 VOSI-capabilities . . . 12
2.5 VOSI-tables . . . 15
2.6 DALI-examples . . . 15
2.7 Parameters . . . 16
2.7.1 LANG . . . 17
2.7.2 QUERY . . . 17
2.7.3 FORMAT and RESPONSEFORMAT . . . 18
2.7.4 MAXREC . . . 18
2.7.5 RUNID . . . 19
2.7.6 UPLOAD . . . 19

3 Use of VOTable **20**
3.1 INFO elements . . . 21
3.2 Successful Queries . . . 21
3.3 Errors . . . 22
3.4 Overflows . . . 22
3.5 Mapping Table Datatypes . . . 23

4 Metadata: TAP_SCHEMA **23**
4.1 Schemas . . . 24
4.2 Tables . . . 25
4.3 Columns . . . 25
4.4 Foreign Keys . . . 27

5 Examples **28**
5.1 Example: Asynchronous Query . . . 29
5.1.1 Creating and Executing a Simple Query . . . 29
5.1.2 Modify a Query Job Before Execution . . . 30
5.1.3 Running a Query . . . 32
5.2 Example: Synchronous Query . . . 33
5.3 Example: DALI-examples Document . . . 34

A Changes from TAP-1.0 to TAP-1.1 **35**
A.1 General Improvements and Clarifications 35
A.2 New Features (including changes from related specifications) . 36
A.3 Removed specifications 36
A.4 ADQL Clarifications 36
A.5 Datatype mapping 36
A.6 TAP_SCHEMA Changes 37
A.7 VOSI/UWS related changes 37

B Detailed Changes from Previous Versions **38**
B.1 PR-TAP-20190420 38
B.2 PR-TAP-1.1-20180830 38
B.3 PR-TAP-1.1-20180416 38
B.4 PR-TAP-1.1-20170830 39
B.5 WD-TAP-1.1-20170707 39
B.6 WD-TAP-1.1-20161011 40
B.7 WD-TAP-1.1-20160428 40
B.8 WD-TAP-1.1-20150930 40
B.9 Changes from TAP-1.0 40

Acknowledgments

The authors would like to acknowledge all contributors to this and previous versions of this standard, especially: K. Andrews, J. Good, R. Hanisch, G. Lemson, T. McGlynn, K. Noddle, F. Ochsenbein, I. Ortiz, P. Osuna, R. Plante, J. Salgado, A. Stebe, and A. Szalay.

Conformance-related definitions

The words “MUST”, “SHALL”, “SHOULD”, “MAY”, “RECOMMENDED”, and “OPTIONAL” (in upper or lower case) used in this document are to be interpreted as described in IETF standard, Bradner (1997).

The *Virtual Observatory (VO)* is general term for a collection of federated resources that can be used to conduct astronomical research, education, and outreach. The International Virtual Observatory Alliance (IVOA) is a global collaboration of separately funded projects to develop standards and infrastructure that enable VO applications.

1 Introduction

The Table Access Protocol (TAP) is a web-service protocol that gives access to collections of tabular data referred to collectively as a tableset. TAP services accept queries posed against the tableset available via the service and return the query response as another table, in accord with the relational model. Queries may be submitted using various query languages and may execute synchronously or asynchronously. Support for the Astronomical Data Query Language ADQL (Osuna and Ortiz et al., 2008) is mandatory; support for other query languages is supported but optional.

The result of a TAP query is another table, normally returned as a VOTable. Support for VOTable output is mandatory; all other formats are optional.

The table collections made accessible via TAP are typically stored in relational database management systems (RDBMS). A TAP service exposes the database schema to client applications so that queries can be posed directly against arbitrary data tables available via the service.

Multi-table operations such as joins or cross matches are possible provided the tables are all managed by the local TAP service, and provided the service supports these capabilities. Larger scale operations such as a distributed cross match are also possible, but require combining the results of multiple TAP services.

1.1 Role within the VO Architecture

Fig. section 1 shows the role this document plays within the IVOA architecture (Arviset and Gaudet et al., 2010).

TAP depends on the following other IVOA standards:

UWS (Harrison and Rixon, 2016) TAP services can be queried asynchronously; the Universal Worker Service UWS defines the corresponding communication pattern. Note that while TAP 1.1 does not require the use of any particular minor version of the UWS standard, UWS 1.1 can significantly streamline the communication, and implementors of TAP 1.1 are encouraged to support UWS 1.1 or later.

ADQL (Osuna and Ortiz et al., 2008) A standards-compliant TAP service must support queries written in the Astronomical Data Query Language.

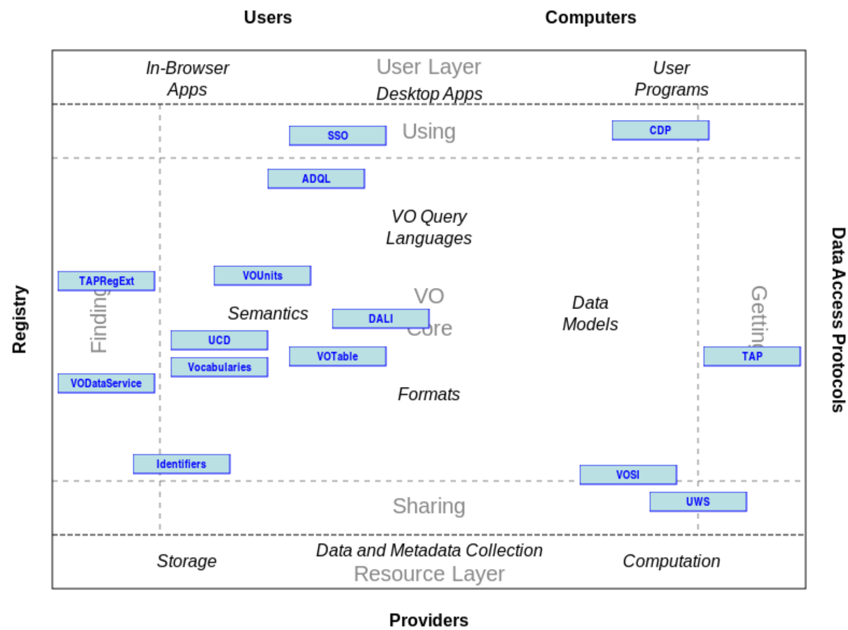

Figure 1: Architecture diagram for this document

VOSI (Graham and Rixon et al., 2017) The VO Support Interfaces standard defines endpoints for metadata discovery; for TAP, this is the tables, capabilities, and availability endpoints. Note that while TAP 1.1 does not require the use of any particular minor version of the VOSI standard, the VOSI-tables resource in VOSI 1.1 provides important usability features, and implementors of TAP 1.1 are encouraged to support VOSI 1.1 or later.

VOTable (Ochsenbein and Taylor et al., 2013) All TAP services must be able to serve query results in the VOTable format. Note that while TAP 1.1 does not require the use of any particular minor version of the VOTable standard, older versions are missing features that are required and may be unusable in practice. For example, the overflow reporting and xtype attribute were introduced in VOTable 1.2 so that is the minimum viable version that must be used.

SSO (Major and Rixon et al., 2016) TAP services that support authenticated access declare this in their capabilities using recommendations and security method identifiers from the SSO standard.

TAP is related in other ways to the following IVOA standards:

DALI (*Dowler and Demleitner et al., 2017*) The Data Access Layer Interface standard gives general rules for the construction of DAL standards. TAP has been written against version 1.1 of DALI. In particular, TAP directly references DALI 1.1's serialisation rules for geometries and timestamps and recommends implementing the examples endpoint defined by DALI.

Obscore (*Tody and Micol et al., 2011*) The Obscore data model facilitates the publication of metadata of observational data products. TAP is used to access compliant metadata collections.

RegTAP (*Demleitner and Harrison et al., 2014*) The relational model for the VO Registry provides a data model for publishing service metadata. TAP is used to access compliant metadata collections.

TAPRegExt (*Demleitner and Dowler et al., 2012a*) While there is no formal requirement to that effect, the response on a TAP service's capabilities endpoint should contain instances of `tr:TableAccess`-typed capabilities in order to allow clients to discover several important aspects of a TAP service's capabilities (e.g., resource limits, output formats, user defined functions).

CDP (*Plante and Graham et al., 2010*) TAP services that support authenticated requests may require delegation of user credentials in order for some features to work; such services will have an associated credential service and could use delegated credentials for remote calls that require authentication. For example, a URL specified in the UPLOAD parameter may require authentication and the service would use delegated credentials to perform the retrieval.

1.2 Motivating Use Cases

Below are some of the more common use cases that have motivated the development of the TAP specification. While this is not complete, it helps to understand the problem area covered by this specification.

1.2.1 Discover Metadata

Since content in relational databases is often custom and project-specific, users of a TAP service must be able to discover the names of tables and

columns, datatypes, units, and other information necessary to construct meaningful correct queries.

1.2.2 Query Custom Tables

A large amount of astronomical data and metadata is stored in tables in relational databases. Historically, users could query these tables through custom user interfaces (usually web page forms), but such approaches could not provide support for truly ad-hoc querying. A TAP service should enable users to discover and query custom tables with a flexible and expressive input that supports ad-hoc querying: selecting output, filtering the result, joining multiple tables, computing aggregate quantities, etc.

1.2.3 Query Standard Tables

A TAP service should enable users to query externally defined standard tables in a uniform way such that the same web service request can be sent to multiple services. Services must be able to declare their support for standard tables in the service metadata.

1.2.4 Query Standard Data Models

A TAP service should enable users to query (parts of) externally defined data models that are (partially or fully) implemented by the service. Services must be able to declare their support for data models as well as the way that model elements are mapped to tables and columns.

1.3 Query Languages

1.3.1 ADQL Queries

The Astronomical Data Query Language ADQL (Osuna and Ortiz et al., 2008) is the standard query language for the IVOA. Support for ADQL queries is mandatory. ADQL can be used to specify queries that access one or more tables provided by the TAP service, including the standard metadata tables. In general, the client must access table metadata in order to discover the names of tables and columns and then formulate queries. ADQL queries provide a direct (low-level) access to the tables; a query will be written for a specific TAP service and will not be usable with other services unless the query refers only to common tables and columns. It is also possible that

the service registration (in an IVOA Registry) may include sufficient table metadata to enable queries to be written directly.

1.3.2 Other Query Languages

A TAP service implementor must be able to include support for other query languages, such as pass-through of native SQL directly to an underlying DBMS or simple key-value (parameter-based) constraints, without making their service non-compliant with the specification. The service interface must allow for this and the service capabilities must be able to describe it. This mechanism also allows future developments within and outside the IVOA to be used without revising the TAP specification.

1.4 Query Execution

1.4.1 Asynchronous Queries

Asynchronous queries allow for long running queries to complete without the client maintaining a connection to the service. Results are stored by the service for later retrieval by the client. Asynchronous query execution is generally more robust and not susceptible to time-outs or other transient failures. They are especially suited to queries that run for a long time before producing output (e.g. queries that compute or aggregate values).

1.4.2 Synchronous Queries

Synchronous queries execute immediately and the client must wait for the query to finish. Synchronous query execution is generally simpler and provides a faster (low latency) response and should be adequate when the query will execute and start returning results quickly. Even with large query results, synchronous queries are a good approach as long as the service can stream the output and consume modest internal resources.

2 Resources

An implementation of a TAP service provides the following RESTful resources under the base URL.

resource type	resource name	required
TAP-sync	/sync	must
TAP-async	/async	must
VOSI-availability	service specific	must (must be anonymous)
VOSI-capabilities	/capabilities	must (must be anonymous)
VOSI-tables	/tables	should
DALI-examples	/examples	should

A TAP service provides a single base URL with child resources for the various features in tbe table above. As required by DALI (Dowler and Demleitner et al., 2017), all resources (including the optional VOSI-tables resource) except the VOSI-availability must be siblings of the VOSI-capabilities resource.

The fixed name resources above (async, sync, tables, and examples) are used for both anonymous and authenticated access to the service; the consequences of having a single base URL are detailed below in section 2.4. The VOSI-availabilty and VOSI-capabilities resources must allow anonymous access as they can be used by clients to determine if the service is available and which resources to use with available security (authentication) methods.

The web resource at the root of the tree represents the service as a whole. This specification defines no standard representation for this root resource. Implementations should return a human readable document (HTML) describing the service as a whole. TAP clients should not depend on a specific representation of the root resource.

2.1 TAP-sync

A TAP service must provide one or more web resources that represent the results of synchronous query execution. The sync resources must conform to the general rules for DALI-sync resources. The exact form of the query, and hence the representation of the resource, is defined by the query parameters as listed in section 2.7. The representation of results of queries is defined in section 2.7.3 and section 3.

For query languages that produce a single result (e.g. ADQL) executed using the /sync endpoint, the result of a successful query is returned in the response or the response includes an HTTP redirect (303: See Other) to a resource from which the result may be retrieved.

An HTTP-GET request to the /sync web resource may return a cached copy of the representation. This cached copy might come from an HTTP

cache between the client and the service, and the service may also maintain its own cache. Clients which require an up-to-date representation of volatile data or metadata must use HTTP POST.

2.2 TAP-async

A TAP service must provide one or more web resource representing controls for asynchronous queries. Specifically, the web resource must conform to the general rules for DALI-async resources and thus represent a job-list as specified in UWS (Harrison and Rixon, 2016).

The child web resources of the /async resource are as specified by UWS. These are descendants of the /async web-resource, and they include a web resource that represents the eventual result of an asynchronous query, e.g.:

```
http://example.com/tap/async/42/results/result
```

where the base URL for the TAP service is:

```
http://example.com/tap
```

the UWS job list is:

```
http://example.com/tap/async
```

and the job resource is

```
http://example.com/tap/async/42
```

where 42 is an example job identifier. A client making an asynchronous request must use the UWS facilities to monitor or control the job. In addition to the job list and job resource above, UWS specifies the name and semantics of a small set of child resources used to view and control the job, e.g.:

```
http://example.com/tap/async/42/phase
http://example.com/tap/async/42/quote
http://example.com/tap/async/42/executionduration
http://example.com/tap/async/42/destruction
http://example.com/tap/async/42/error
http://example.com/tap/async/42/parameters
http://example.com/tap/async/42/results
http://example.com/tap/async/42/owner
```

Successful TAP queries produce results which must be accessible as resources under the UWS result list, e.g.:

```
http://example.com/tap/async/42/results/
```

Failed TAP queries produce an error document (see section 3.3) which must be accessible as the error resource, e.g.:

```
http://example.com/tap/async/42/error
```

For query languages that produce a single result executed using the /async endpoint, the result of a successful query can be found within the result list specified by UWS; the result must be named result and thus clients are able to access it directly, e.g.:

```
http://example.com/tap/async/42/results/result
```

Access of this resource must deliver the result, either directly or as an HTTP redirect (303: See Other) to a resource from which the result may be retrieved.

For query languages that may produce multiple result resources, the names of the results are not specified (they may be specified in the specification for the language). The client can always access the result list resource as specified by UWS.

If the query returned no rows, the result resource must exist and contain no data rows. Details on interacting with these resources are specified in the UWS standard; for examples specific to TAP see section 5 below.

2.3 VOSI-availability

The use of the VOSI-availability resource is described in DALI.

2.4 VOSI-capabilities

The TAP-1.0 standard is identified using

```
ivo://ivoa.net/std/TAP
```

For TAP-1.1 we use the same standardID value. The version attribute of the interfaces must use the minor version (e.g. `version="1.1"`) of the TAP standard; clients should treat a missing version attribute is equivalent to `version="1.0"`.

The interface element within the TAP capability specifies the base URL of the service under which the fixed-name resources specified above (section 2) are located. In addition to the accessURL element, the interface element may also contain zero or more securityMethod elements that specify

the supported authentication methods. Zero such elements and a securityMethod with no standardID attribute are both interpreted as meaning anonymous; the latter should only be used if other securityMethod(s) are also specified and zero such elements is preferable in an anonymous-only service.

Here is an example for a document as it might be returned from a VOSI capabilities endpoint of a TAP service:

```
<vosi:capabilities
   xmlns:vosi="http://www.ivoa.net/xml/VOSICapabilities/v1.0"
   xmlns:vs="http://www.ivoa.net/xml/VODataService/v1.1"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
    "http://www.ivoa.net/xml/VOSICapabilities/v1.0
     http://www.ivoa.net/xml/VOSICapabilities/v1.0
     http://www.ivoa.net/xml/VODataService/v1.1
     http://www.ivoa.net/xml/VODataService/v1.1">

  <!-- base URL for clients that apppend /async, /sync, etc -->
  <capability standardID="ivo://ivoa.net/std/TAP">
    <interface xsi:type="vs:ParamHTTP" role="std" version="1.1">
      <accessURL use="base"> https://example.net/srv </accessURL>
      <!-- anonymous or cookie or client certificate -->
      <securityMethod/>
      <securityMethod standardID="ivo://ivoa.net/sso#cookie"/>
      <securityMethod standardID="ivo://ivoa.net/sso#tls-with-certificate"/>
    </interface>
  </capability>

  <capability standardID="ivo://ivoa.net/std/VOSI#capabilities">
        <!-- VOSI capabilities endpoint as per VOSI 1.1 -->
    <interface xsi:type="vs:ParamHTTP" role="std" version="1.0">
      <accessURL use="full"> https://example.net/srv/capabilities </accessURL>
      <!-- anonymous only -->
    </interface>
  </capability>

  <capability standardID="ivo://ivoa.net/std/VOSI#availability">
        <!-- VOSI availability endpoint as per VOSI 1.1 -->
    <interface xsi:type="vs:ParamHTTP" role="std" version="1.0">
      <accessURL use="full"> https://example.net/srv/availability </accessURL>
      <!-- anonymous only -->
    </interface>
  </capability>

  <capability standardID="ivo://ivoa.net/std/VOSI#tables-1.1">
        <!-- VOSI table endpoint as per VOSI 1.1 -->
    <interface xsi:type="vs:ParamHTTP" role="std" version="1.1">
```

```
      <accessURL use="base"> https://example.net/srv/tables </accessURL>
      <!-- anonymous or cookie or client certificate -->
      <securityMethod/>
      <securityMethod standardID="ivo://ivoa.net/sso#cookie"/>
      <securityMethod standardID="ivo://ivoa.net/sso#tls-with-certificate"/>
    </interface>
  </capability>

</vosi:capabilities>
```

This specification requires that exactly one capability with a standardID of `ivo://ivoa.net/std/TAP` is present. In addition to this basic service description, production services should use the TAPRegExt (Demleitner and Dowler et al., 2012b) extension to provide additional metadata within the TAP capability. Clients should create the endpoint URLs by appending the fixed endpoint names to the access URL given in the interface defined in the TAP capability. The TAP capability normally has a single interface; multiple interfaces are permitted in order to support multiple versions (different version attribute on the interface element and different accessURL).

This specification does not require that the capabilities document includes a capability description for the VOSI features themselves. However, these are needed in the document and in registry records until such time that another standard (DALI or VOResource) provides an alternative mechanism to convey required information to clients (standardID or equivalent, version, accessURL(s), supported securityMethod(s), etc.). In the example above, the VOSI-tables capability conveys the version (1.1) and that multiple securityMethod(s) are supported. The version tells clients that a returned tableset may not contain column metadata and that the detail parameter is supported. The securityMethod(s) tell clients that authenticating may effect the output such as exposing metadata for protected tables.

The example above describes a TAP service that can be accessed anonymously or by authenticating with a cookie (`ivo://ivoa.net/sso#cookie`) or an X509 client certificate (`ivo://ivoa.net/sso#tls-with-certificate`). The same URL (e.g. `https://example.net/tap/sync` for a synchronous query) would be used for both anonymous and authenticated access.

As a consequence of using a single base URL with fixed name child resources, all supported authentication methods must be able to co-exist on the same URL. At this time, both anonymous and other authentication methods described in SSO can be made to work in this way, with the exception of HTTP Basic and Digest authentication. With a single base URL, one can provide an anonymous-only TAP service with http or https; once

support for `ivo://ivoa.net/sso#tls-with-certificate` is added, however, the entire service must use https only. While one can provide a service with `securityMethod="ivo://ivoa.net/sso#BasicAA"`, this authentication method uses the HTTP protocol to implement it's challenge-response and thus cannot co-exist with other methods (including anonymous); such a service would only be able to describe that single method. However, the SSO standard recommends that username-password authentication (including BasicAA) be used to login and obtain another type of credential (e.g. a cookie) and that services should not directly use username-password for authentication; the single base URL design here is consistent with the recommendations of SSO.

In the example above, the VOSI-availability and VOSI-capabilities interfaces are anonymous (no securityMethod). The VOSI-tables interface is typically also anonymous-only, but in the example we show that it may also support anonymous and/or authenticated access. In general, clients that support authentication should be prepared to discover and use anonymous-only endpoints for some requests.

2.5 VOSI-tables

The table metadata should be accessible from a web resource with relative URL /tables that is a direct child of the root web resource. The /tables resource implements the VOSI-tables capability and output described in VOSI. The content is equivalent to the metadata from the TAP_SCHEMA described in section 4. The use of VOTableType (rather than TAPType) in the VOSI-tables output is recommended because the values map directly; TAPType may be used when VOTableType does not provide a suitable alternative.

Services which do not implement the /tables resource must respond with an HTTP response code of 404 when this resource is accessed.

2.6 DALI-examples

A successful GET from this endpoint MUST yield a document with a MIME type of either application/xhtml+xml or text/html. A service that does not provide examples MUST return a 404 HTTP status on accessing this resource.

If present, the endpoint must be represented in a capability in the TAP service's registry record. The capability's *`standardID`* is defined by DALI. A capability element could hence look like this:

```
<capability standardID="ivo://ivoa.net/std/DALI#examples">
   <interface xsi:type="vr:WebBrowser">
      <accessURL use="full"
        >http://myarchive.net/myTAP/examples</accessURL>
   </interface>
</capability>
```

TAP defines two additional properties for the examples vocabulary:

- query – each example MUST have a unique child element with simple text content having a `property` attribute valued *query*. It contains the query itself, preferably with extra whitespace for easy human consumption and editing. This will usually be a HTML `pre` element.

- table – examples MAY also have descendants with `property` attributes having the value *table*. These must have pure text content and contain fully qualified table names to which the query is somehow "pertaining".

Although it might be tempting, examples authors should not put table names into HTML `a` elements (e.g., to link to the table descriptions). As discussed in DALI 1.1, sect. 2.3, this would result in invalid RDF statements.

An example of a response from a TAP service's examples endpoint is given in section 5.3.

2.7 Parameters

The {async} and {sync} web-resources must accept the parameters listed in the following sub-sections. In a synchronous request, the parameters select the representation returned in the response message. In an asynchronous request, the parameters select the representation of the eventual query result rather than the response to the initial request.

Requirements on the presence and values of parameters described below are enforced only when the TAP request is executed (not when individual HTTP requests are handled). Thus, for asynchronous TAP queries, the parameter requirements must be satisfied (and errors returned if not) only when the query is run (in the sense of UWS job execution). Specifically, asynchronous queries may be created with with no parameters and multiple, subsequent HTTP POST actions may specify the parameters in any order.

Not all combinations of the parameters are meaningful. If a service receives a spurious parameter in an otherwise correct request, then the service must ignore the spurious parameter, must respond to the request normally and must not report errors concerning the spurious parameter.

2.7.1 LANG

The LANG parameter specifies the query language. The service must support the LANG parameter and the client must provide a value. The only standard value for the LANG parameter is ADQL. Support for other languages and the LANG value to use with them can be described in TAPRegExt service capabilities (Demleitner and Dowler et al., 2012a).

For example, an ADQL query would be performed with

```
LANG=ADQL
QUERY=<ADQL query string>
```

A query with a custom query language would be performed with

```
LANG=MySecretLang
<MySecretLang-specific parameters>
```

The value of LANG is a string specifying the language and optionally the language version used for the query parameter(s), as defined by the service capabilities. The client may specify the version of the query language, e.g. LANG=ADQL-2.0 (the syntax should be as shown) or it may omit the version, e.g. LANG=ADQL. The service should return an "unknown query language" error as described in section 3.3 if an unsupported language or an incompatible language version is specified.

2.7.2 QUERY

The QUERY parameter is used to specify queries that are serialised as a single character string, such as an ADQL query (with LANG=ADQL or some version thereof). The interpretation of the value depends on the value of the LANG parameter. This parameter should also be used to specify the query for other values of LANG (e.g. LANG=<some RDBMS-specific SQL variant>) when appropriate.

A service must support the QUERY parameter because ADQL is a required language.

If timestamp comparisons are supported within ADQL queries, they must use the syntax defined in DALI. Timestamp values are usable if there are columns with timestamp values, including in uploaded tables if table upload is supported.

If the service supports the use of spatial coordinates in ADQL queries, the output of geometry values should use the syntax defined in DALI. Services may output geometry values using the STC-S convention described in the

previous version of this standard, but we strongly recommend switching to the DALI syntax.

If table upload is supported, values using the DALI syntax must be supported and values using the previous STC-S convention may be supported for backwards compatibility. Input and output of all values must be supported (e.g. selecting all columns from an uploaded table) for all types even if comparisons are not supported.

Note: Although it is allowed by the ADQL-2.0 syntax, clients should be careful when mixing constants and column references for coordinate system and coordinate values. For example, POINT('ICRS', t.ra, t.dec) does not cause t.ra and t.dec to be transformed to ICRS; it simply tells the service to treat the values as being expressed in that coordinate system. Clients should avoid using the coordinate system argument to geometric functions (supply the empty string, or use an alternate function without the coordinate system argument if available).

2.7.3 FORMAT and RESPONSEFORMAT

The RESPONSEFORMAT parameter is fully described in DALI. For backwards compatibility, TAP-1.1 must also accept the FORMAT parameter as equivalent to RESPONSEFORMAT. Specifying both FORMAT and RESPONSEFORMAT is undefined.

If both the FORMAT and RESPONSEFORMAT parameters are omitted, the default format is VOTable. A TAP service must support VOTable as an output format, should support CSV and TSV output, and may support other formats.

2.7.4 MAXREC

The MAXREC parameter and its effect on the query result is fully described in DALI. If the result set is truncated in this fashion, it must include an overflow indicator as specified in section 3.4.

For the special value of MAXREC=0, the service is not required to execute the query; a successful MAXREC=0 request does not necessarily mean that the query is valid and the overflow indicator does not necessarily mean that there is at least one row satisfying the query. The service may perform validation and may try to execute the query, in which case a MAXREC=0 request can fail. A query with MAXREC=0 can be used with a simple query (e.g. SELECT * FROM some_table) to extract and examine the VOTable metadata (assuming RESPONSEFORMAT=votable). Note: in this version

of TAP, this is the only mechanism to learn some of the detailed metadata, such as coordinate systems used.

The output truncation caused by non-zero values of the MAXREC parameter occurs after any limitations imposed by the query and the overflow indicator is only added if the query result is actually truncated. For example:

```
MAXREC=A
QUERY=select TOP B * from foo
```

for integer values A and B. Assuming the table contains many rows, if A > B then the result table will contain B rows and no overflow indicator. If A < B then the result table will contain A rows and an overflow indicator. If the table contains A or fewer rows then the result will not contain an overflow indicator.

2.7.5 RUNID

The RUNID parameter is fully described in DALI.

2.7.6 UPLOAD

The UPLOAD parameter is described in DALI. Services should support the upload of temporary tables in VOTable (Ochsenbein and Taylor et al., 2013) format via the standard UPLOAD parameter. The table-name(s) must be legal table names as defined in ADQL (Osuna and Ortiz et al., 2008) but restricted as described in section 4.2. URIs may be simple URLs (e.g. with a URI scheme of http or https) or URIs that must be resolved (e.g. with a URI scheme of vos or param) to give the location of the table content.

If table upload is supported, the service must accept tables in VOTable format. The client specifies the name of the uploaded table; this name must be a legal ADQL table name with no catalog or schema (i.e., a string following the regular identifier production of ADQL). Uploaded tables must be referred to in queries as TAP_UPLOAD.<tablename>, where <tablename> is the name specified by the user. Tables in the TAP_UPLOAD schema are transient and persist only for the lifetime of the query (although caching might be used behind the scenes) and are never visible in the TAP_SCHEMA metadata.

For uploaded tables, the name attribute of the FIELD element is used as the column name. Services must support delimited identifiers so that FIELD names that are not valid ADQL column names work correctly.

The DALI UPLOAD parameter supports both external resources and in-line content. For external resources, one provides a URI (usually an HTTP URL) the TAP service can use to obtain the table content. For example,

```
HTTP POST http://example.com/tap/async/42
UPLOAD=mytable,http://otherplace.com/path/votable.xml
```

The service would retrieve the table from the provided URL and make it visible to the query as TAP_UPLOAD.mytable.

If the TAP service supports VOSpace URIs, one may specify an upload table using a URI to a table stored in a VOSpace, for example:

```
HTTP POST http://example.com/tap/async/42
UPLOAD=mytable,vos://space/path/votable.xml
```

The service would resolve the URI, contact the VOSpace, retrieve the table, and make it visible to the query as TAP_UPLOAD.mytable.

UPLOADs are accumulating, i.e., each UPLOAD parameter given will create one or more tables in TAP_UPLOAD. When the table names from two or more upload items agree after case folding, the service behaviour is unspecified. Clients thus cannot reliably overwrite uploaded tables; to correct errors, they have to tear down the existing job and create a new one. In principle, any number of tables can be uploaded using the UPLOAD parameter and any combination of URI schemes supported by the service as long as they are assigned unique table names within the query. Services may limit the size and number of uploaded tables; if the service refuses to accept the entire table it must respond with an error as described in section 3.3.

Table upload must support all valid VOTable content even if they do not support all features and uses of extended data types; clients must be able to upload and then query a valid table and round-trip all values. Services should store extended types (e.g. timestamp) in an appropriate database column type in order to facilitate predictable use in queries.

3 Use of VOTable

VOTable (Ochsenbein and Taylor et al., 2013) is the standard format for output (query results) and input (table upload) in a TAP service so most of this section deals with how VOTable is used. However, rules about serialising column values also apply to other formats (e.g. CSV and TSV).

The use of VOTable in TAP services is described in DALI, with additional clarifications or advice below.

3.1 INFO elements

The RESOURCE element must contain INFO element(s) as described in DALI.

Additional INFO elements may be provided, e.g., to echo the input parameters back to the client in the query response (a useful feature for debugging or to self-document the query response), but clients should not depend on these.

```
<RESOURCE type="results">
...
<INFO name="QUERY">
    select * from stuff.items
</INFO>
...
</RESOURCE>
```

3.2 Successful Queries

The result of a query depends on the query language used and may be one or more tables in one or more resources. Unsupportable combinations of query result and RESPONSEFORMAT (e.g. queries that produce multiple tables and an inherently single-table format like CSV) will cause the request to fail. Currently, an ADQL query result must be a single table (in a single file).

The output table must include the same number and order of columns as specified in the SELECT clause of the query. For VOTable output, the name attribute of FIELD elements must be the same as the column names (or aliases if specified) in the query and the datatype, arraysize, and xtype attributes of FIELD elements must be set using the values from the TAP_SCHEMA. In cases where items in the select list do not have names (e.g. expression or function invocation without an alias) the service must generate a name; generated names must be unique (within the output table) and should be valid ADQL identifiers.

CSV formatted data should represent the output table with one row of text per table row, with the table column values rendered as text. If a column value contains a comma, the entire column value should be enclosed in double quotes. Text lines may be arbitrarily long. The first data row should give the column name(s) as the data value. CSV data must be returned with a MIME type of text/csv; if the optional header line (with column names) is included, the MIME type must be text/csv;header=present. Full details of CSV format are defined in Shafranovich (2005).

TSV formatted data should represent the output table with one row of text per table row, with the table column values rendered as text and separated by the TAB character. TSV data must be returned with a MIME type of text/tab-separated-values as described in University of Minnesota Gopher Team (1993). Column values may not contain the TAB character.

3.3 Errors

If the service detects an exceptional condition, it must return an error document with an appropriate HTTP-status code. TAP distinguishes three classes of exceptions.

- Errors in the use of the HTTP protocol.
- TAP-level errors: Invalid TAP requests
- TAP-level errors: Failure of the service to complete a valid request.

Error documents for HTTP-level errors are not specified in the TAP protocol. Responses to these errors are typically generated by service containers and cannot be controlled by TAP implementations. There are several cases where a TAP service could return an HTTP error. First, the /async endpoint could return a 404 (not found) error if the client accesses a job within the UWS job list that does not exist. Second, access to a resource could result in an HTTP 401 (not authorized) error if authentication is required or an HTTP 403 (forbidden) error if the client is not allowed to access the resource.

Error documents should be in a format that matches the requested format where possible; see DALI for details. If the error document is being retrieved from the /async/<jobid>/error resource (specified by UWS) after an asynchronous query, the HTTP status code should be 200. If the error document is being returned directly after a synchronous query, the service may use an appropriate HTTP status code, including 200 (successfully returning a response to the request) and various 4xx and 5xx values.

3.4 Overflows

If a query is executed by a TAP service, the number of rows in the table of results may exceed a limit requested by the user (using the MAXREC parameter) or a limit set by the service implementation (the default or maximum value of MAXREC). In these cases, the query is said to have "overflowed".

Typically, a TAP service will not detect an overflow until some part of the table of results has been sent to the client.

If an overflow occurs, the TAP service must produce a table of results that is valid, in the required output format, and which contains all the results up to the point of overflow. Since an output overflow is not an error condition, the MIME type of the output must be the same as for any successful query and the HTTP status-code must be as for a successful, complete query.

Reporting of overflow depends on the output format and is described in DALI.

3.5 Mapping Table Datatypes

This section describes the bi-directional mapping between VOTable on the one and the RDBMS (including its geometric datatypes) on the other side. It extends the basic rules for serialising such values in VOTable described in DALI. These rules apply to input tables supplied via an UPLOAD parameter (see section 2.7.6) and to result tables after successful query execution.

The mapping to and from VOTable makes use of the datatype, arraysize, and xtype attributes. Mapping for primitive types (numbers and strings) is straightforward; services should ensure that input values behave as expected in query processing and output values should have correct and complete metadata. Mapping for specially structured values use xtype(s) specified in DALI. The behaviour of such structured values in queries depends on the query language (section 2.7.1) being used.

4 Metadata: TAP_SCHEMA

There are several approaches to getting metadata for a given TAP service. All TAP services must support a set of tables in a schema named TAP_SCHEMA that describe the tables and columns included in the service. In addition to the TAP_SCHEMA, there are two other ways to get metadata from a TAP service. First, the VOSI tables resource provides metadata on all tables and columns; this resource is described in section 2.5. The VOSI tables resource provides the same metadata as the TAP_SCHEMA but in a rigorously controlled format; the information in the TAP_SCHEMA is equivalent to that defined by VODataService (Plante and Stébé et al., 2010). Second, the client may specify a query of one or more tables setting the MAXREC parameter to 0 so that only the metadata regarding the requested fields is returned. Use of MAXREC is described in section 2.7.4.

The TAP_SCHEMA provides access to table, column, and join key metadata through the TAP query mechanisms themselves. Users can discover tables or columns that meet their specific criteria by querying the tables described below. The service may enhance the TAP_SCHEMA with additional metadata where that seems appropriate; since it is self-describing, the TAP_SCHEMA may be queried to determine if any extended schema metadata is defined by the service. Services must provide these tables and make them accessible by all supported query mechanisms.

Column datatypes in the TAP_SCHEMA are specified using the same concepts used in VOTable: datatype, arraysize, and xtype. For backwards compatibility, implementors must also include the `"size"` column and populate it where possible. In the details below, TAP_SCHEMA column types are either string or integer; implementers may choose an appropriate data type that behaves the same way in queries and output (e.g. varchar(16) or varchar(64) for string and smallint, int, or bigint for integer. Implementors should use datatype and arraysize values that best describe their implementation of the TAP_SCHEMA tables (e.g., datatype char and arraysize 64* to describe a column with database type varchar(64)).

Implementors may include additional tables in the TAP_SCHEMA to describe additional aspects of their service not covered by this specification. Implementors may also include additional columns in the standard tables described below. For example, one could include a column with a timestamp saying when metadata values were last modified.

4.1 Schemas

The table TAP_SCHEMA.schemas must contain the following columns:

column name	type	not-null
schema_name	string	true
utype	string	false
description	string	false
schema_index	integer	false

The schema_name values must be unique and may be qualified by the catalog name or not depending on the implementation requirements. The fully qualified schema name is defined by the ADQL language and follows the pattern [catalog.]schema. The schema metadata are included for reference and are not used directly to construct queries.

4.2 Tables

The table TAP_SCHEMA.tables must contain the following columns:

column name	**type**	**not-null**
schema_name	string	true
table_name	string	true
table_type	string	true
utype	string	false
description	string	false
table_index	integer	false

The table_name values must be unique. The value of the table_name should be the string that is recommended for use in querying the table; it may or may not be qualified by schema and catalog name(s) depending on the implementation requirements. The fully qualified table name is defined by the ADQL language and follows the pattern [[catalog.]schema.]table. If the table name is such that the name must be quoted (delimited identifier in ADQL) then the value must include the quotes.

The table_type value must be either table or view.

The table_index is used to recommend table ordering for clients. Clients may order by table_index (ascending) so lower index tables would appear earlier in a listing.

4.3 Columns

The table TAP_SCHEMA.columns must contain the following columns:

column name	type	not-null
table_name	string	true
column_name	string	true
datatype	string	true
arraysize	string	false
xtype	string	false
“size”	integer	false
description	string	false
utype	string	false
unit	string	false
ucd	string	false
indexed	integer	true
principal	integer	true
std	integer	true
column_index	integer	false

The table_name,column_name (pair) values must be unique. If the column name is such that the name must be quoted (delimited identifier in ADQL) then the value must include the quotes.

The type of a database column is described in the TAP_SCHEMA.columns table using three columns with an additional (deprecated) column from TAP-1.0 for backwards compatibility. The allowed values for datatype and the syntax for arraysize are specified in VOTable (Ochsenbein and Taylor et al., 2013). Values for xtype are not restricted per se but implementors should use standard values such as those defined in DALI before inventing new xtype(s).

The arraysize column gives the length of fixed and variable length datatypes using the VOTable array shape syntax. For example, a database column of type varchar(256) would be described with datatype “char” and arraysize “256*”. Arrays, including multi-dimensional arrays, are permitted for all VOTable primitive types. The `"size"` column is retained for backwards compatibility to TAP-1.0 and must contain the integer value equivalent to arraysize when possible (ignoring the variable-size indicator in arraysize) and must be null if arraysize is null or represents a multi-dimensional array. For example, a column description with arraysize=“256*” must also have `"size"`=“256”. Both arraysize and `"size"` must be null for scalar numeric columns.

To use the `"size"` column in a query, the column name must be put in double quotes since it collides with an ADQL reserved word. Since delimited identifiers are case-sensitive, for the `"size"` column both clients and servers

MUST always (in particular, in the creation of the TAP_SCHEMA) use lower case exclusively. In the next major version of TAP, the `"size"` column will be removed.

For columns with a database type equivalent to BLOB or CLOB, most database systems support reference to these columns in the select clause but not in every other part of the query where character columns may be used. In addition, services may want to define generated columns where the output is dynamically generated but the content is not stored in the databasase in a form that supports querying. For example, if service implementors want to make URL(s) available as column values in the results, but do not actually store the URL(s) in the database, the column with URL(s) could be referenced in the SELECT clause of a query, but could not sensibly be used in the WHERE clause. In general, if a query references a column in an inappropriate part of the query, the job should fail with a suitable error message.

The principal, indexed, and std columns are boolean values implemented as integers. As such, the value must be 0 or 1; no other values are allowed.

The principal flag indicates that the column is considered a core part of the content; clients can use this hint to make the principal column(s) visible, for example by selecting them by default in generating an ADQL query. In cases where the service selects the columns to return (such as a query language without an explicit output selection), the principal column indicates those columns that are returned by default.

The indexed flag indicates that the column is indexed, potentially making queries run much faster if this column is used in a constraint.

The std flag is included for compatibility with the registry, which uses this value to indicate that a given column is defined by some standard, as opposed to a custom column defined by a particular service.

The column_index is used to recommend column ordering for clients. Clients may order by column_index (ascending) so lower index columns would appear earlier in a listing. This is useful for keeping related columns together in output or display.

4.4 Foreign Keys

The table TAP_SCHEMA.keys must contain the following columns to describe foreign key relations between tables:

column name	type	not-null
key_id	string	true
from_table	string	true
target_table	string	true
description	string	false
utype	string	false

The key_id values are unique and used only to join with the key_columns table below. There may be one or more rows with different key_id values and a pair of tables to denote one or more ways to join the tables.

The table TAP_SCHEMA.key_columns must contain the following columns to describe the columns that make up a foreign key:

column name	type	not-null
key_id	string	true
from_column	string	true
target_column	string	true

There may be one or more rows with a specific key_id to denote single or multi-column keys.

For the TAP_SCHEMA itself, services should enforce and describe the following foreign keys:

```
TAP_SCHEMA.tables.schema_name -> TAP_SCHEMA.schemas.schema_name
TAP_SCHEMA.columns.table_name -> TAP_SCHEMA.tables.table_name
TAP_SCHEMA.keys.from_table -> TAP_SCHEMA.tables.table_name
TAP_SCHEMA.keys.target_table -> TAP_SCHEMA.tables.table_name
TAP_SCHEMA.key_columns.key_id -> TAP_SCHEMA.keys.key_id
```

In addition to the above constraints, the from_table and from_column (in TAP_SCHEMA.keys and TAP_SCHEMA.key_columns respectively) must refer to an entry in the TAP_SCHEMA.columns table. Likewise, the target_table and target_column must also refer to an entry in TAP_SCHEMA.columns.

A TAP service must provide the tables listed above and may provide other tables in the TAP_SCHEMA namespace.

5 Examples

The UWS pattern is specified in the UWS specification (Harrison and Rixon, 2016) and its application to TAP in section 2.2. TAP services may implement

UWS 1.0 or a later version. This section gives examples of the exchange of messages between a TAP client and service when using UWS to run an asynchronous query.

5.1 Example: Asynchronous Query

Consider a TAP service at http://example.com/tap. TAP mandates that the asynchronous requests be directed to http://example.com/tap/async (e.g. for anonymous queries). This URL points to the list of "jobs", i.e., the list of queries currently executing or recently executed.

5.1.1 Creating and Executing a Simple Query

Asynchronous queries are created in the same way as synchronous, using one of the async endpoints, for example:

```
HTTP POST http://example.com/tap/async
LANG=ADQL
QUERY=SELECT * FROM magnitudes AS m WHERE m.r>=10 AND m.r<=16
```

The service's response to this request is an HTTP redirect with a URL for the query job:

```
HTTP status 303 'See other'
Location: http://example.com/tap/async/42
```

The query result or an error document can then be retrieved from a URL associated with the job. This is an application of the UWS pattern. The query is then executed with a separate request to run the job URL:

```
HTTP POST http://example.com/tap/async/42/phase
PHASE=RUN
```

The state of the job can be retrieved from the phase resource:

```
HTTP GET http://example.com/tap/async/42/phase
```

The client may have to check the phase multiple times until the job finishes. If the service implements UWS 1.1 (Harrison and Rixon, 2016) or later, a blocking call can be used instead of polling. Once the job reaches the COMPLETED phase, the results can be obtained from the results resource:

```
HTTP GET http://example.com/tap/async/42/results/result
```

5.1.2 Modify a Query Job Before Execution

To create a new query, the client POSTs a request to the job list:

```
HTTP POST http://example.com/tap/async
LANG=ADQL
```

The response with the job URL:

```
HTTP status 303 'See other'
Location: http://example.com/tap/async/42
```

While the job is in the PENDING phase, the job parameters may be modified by additional POST(s) to the parameters resource (see Dowler and Demleitner et al. (2017)), for example:

```
HTTP POST http://example.com/tap/async/42/parameters
UPLOAD=mytable,http://a.b.c/mytable.xml
QUERY=select * from TAP_UPLOAD.mytable t join magnitudes
  m on t.target = m.target
```

Here we have specified with the UPLOAD parameter that the service create a temporary table named mytable with content from the VOTable at the specified URL. The QUERY parameter can then reference the uploaded table with the specified name (but in the TAP_UPLOAD schema).

Parameter-value pairs accumulate when POSTed to the parameters resource, so an additional POST of the UPLOAD parameter in this example would add another parameter-value pair (essentially a multi-valued parameter as described in DALI). There is no mechanism to replace or remove a parameter in a PENDING job.

After each such POST, the service issues an HTTP redirection to the job's URL, where the modified state may be accessed:

```
HTTP status 303 'See other'
Location: http://example.com/tap/async/42
```

All TAP-specific parameters are stored using the parameter list mechanism of UWS and are included in the XML representation of the job:

```
HTTP GET http://example.com/tap/async/42
```

or directly from the parameters resource:

```
HTTP GET http://example.com/tap/async/42/parameters
```

Individual parameters cannot be accessed as separate web resources.

The UWS pattern requires the following resources to describe and control the job:

```
http://example.com/tap/async/42/phase
http://example.com/tap/async/42/quote
http://example.com/tap/async/42/executionduration
http://example.com/tap/async/42/destruction
http://example.com/tap/async/42/results
http://example.com/tap/async/42/error
```

The quote resource specifies the predicted completion time for the job (query), assuming it is started immediately. In practice, it is very hard to estimate the time a query will take; for TAP services it is recommended that this is set to the current time plus the maximum amount of time the query will be allowed to run. The executionduration resource specifies the amount of time (in seconds) the job (query) will be allowed to run before being aborted by the service. The execution duration is set by the service and can be read from the job or directly from the executionduration resource:

```
HTTP GET http://example.com/tap/async/42/executionduration
```

The service may allow the client to change the duration:

```
HTTP POST http://example.com/tap/async/42/executionduration
EXECUTIONDURATION=600
```

The destruction resource specifies when the service will destroy the job. The service is only required to keep a job for a finite period of time, after which it may destroy the job, including the result. After this time, the client will receive an HTTP 404 'not found' status if it tries to get any information about the job. The destruction time of the job is chosen by the service and the client can read it from the job or directly from the destruction resource:

```
HTTP GET http://example.com/tap/async/42/destruction
```

The service may allow the client to change the destruction time:

```
HTTP POST http://example.com/tap/async/42/destruction
DESTRUCTION=2008-11-11T11:11:11Z
```

In general, clients should fully specify the query job parameters and then check and possibly negotiate the UWS job control parameters.

5.1.3 Running a Query

The phase URL shows the progress of the job. When the job is created by the service it will normally be set to PENDING, but might be set to ERROR if the service has rejected the job. If the phase is ERROR, then the error URL should lead to a an error document explaining the problem. If the phase is PENDING, then the client needs to commit the job for execution.

The client runs the job by posting to the phase URL:

```
HTTP POST http://example.com/tap/async/42/phase
PHASE=RUN
```

The service replies with a redirection to the job URL

```
HTTP status 303 'see other'
Location: http://example.com/tap/async/42
```

The phase will now have changed to either QUEUED or EXECUTING, depending on the service implementation. The client tracks the execution by polling the phase URL (UWS-1.0):

```
HTTP GET http://example.com/tap/async/42/phase
```

or by performing a blocking GET of the job url (UWS-1.1 or later):

```
HTTP GET http://example.com/tap/async/42?WAIT=30
```

The blocking GET will block until something changes in the job (usually the phase change) and then return, with a maximum wait time specified by the WAIT parameter. Since services may impose a limit on the maximum wait time and may return before the job reaches a final phase, clients must examine the job state (returned by the GET) and possibly perform additional requests.

A job in the QUEUED or EXECUTING phase may be aborted by posting to the phase URL:

```
HTTP POST http://example.com/tap/async/42/phase
PHASE=ABORT
```

When the query job is complete, the phase changes will normally be one of COMPLETED, ABORTED, or ERROR (although there are other less used phases defined in UWS). The client then retrieves the result from the results list:

```
HTTP GET http://example.com/tap/async/42/results/result
```

The client knows that the table of results is at the URL /result relative to the results list because the TAP protocol requires this naming. A generic UWS client could find the name of the result and retrieve it by examining either the job description:

```
HTTP GET http://example.com/tap/async/42
```

or by looking specifically at the result list:

```
HTTP GET http://example.com/tap/async/42/results
```

If the service cannot run the query, then the final phase is ERROR and there is no table of results. In this case, the client should expect an HTTP 404 'not found' status if it tries to retrieve the result. The client should look instead at the error resource to find out what went wrong:

```
HTTP GET http://example.com/tap/async/42/error
```

If the job was aborted (by the client or the service), the final phase will be ABORTED and there is no table or results. As with errors, the client should look at the error resource to find out what went wrong.

5.2 Example: Synchronous Query

Synchronous queries return the table of results in the HTTP response to the initial request. This is an example of a synchronous ADQL query on r magnitude:

```
HTTP POST http://example.com/tap/sync
LANG=ADQL
QUERY=SELECT * FROM magnitudes as m where m.r>=10 and m.r<=16
```

In this example, the output format defaults to VOTable; the RESPONSEFORMAT parameter could be added to select a different format.

Many implementations will implement synchronous query execution using the common POST-redirect-GET pattern. If this is the case, the response from the initial POST will be a redirect (HTTP response code 303) and another URL for the results, e.g.:

```
Location: http://example.com/tap/sync/53/go
```

As described in DALI, clients must be prepared to follow such redirects to obtain the result.

5.3 Example: DALI-examples Document

The following is a full document a service might serve from its `/examples` endpoint; note that you can add arbitrary styling or further HTML material without impacting the document's functionality.

The document also shows how operators can add links to table descriptions without breaking RDFa semantics.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.1//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-2.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" version="XHTML+RDFa 1.1">
<head>
  <title>An Example for /examples</title>
</head>
<body vocab="http://www.ivoa.net/rdf/examples#">
  <h1>An Example for /examples</h1>

  <p>This document illustrates how to write a DALI-compliant examples
  document using TAP's special table and query properties.  Apart from the
  actual examples blocks, you can add arbitrary additional material, for
  instance, a table of contents:</p>

  <ul id="toc">
    <li>
      <a href="#QueryforCALIFAobjectproperties"
        >Query for CALIFA object properties</a>
    </li>
    <li>
     <a href="#Queryagainstcoverage"
       >Query against coverage</a>
    </li>
  </ul>

  <p>Use the <a href="https://www.w3.org/2012/pyRdfa/Validator.html"
  >W3C RDFa validator</a> to see what semantics your RDFa actually
  conveys.</p>

  <div typeof="example"
      id="QueryforCALIFAobjectproperties"
      resource="#QueryforCALIFAobjectproperties">
    <h2 property="name">Query for CALIFA object properties</h2>
    <p>This example shows how to combine the
    <a href="/tableinfo/califadr3.objects">&#x2197;</a
    ><em property="table">califadr3.objects</em> table
    of properties of CALIFA target galaxies with the
    <a href="/tableinfo/califadr3.cubes">&#x2197;</a
    ><em class="dachs-ex-taptable" property="table">califadr3.cubes</em>
    table of data cubes to select bright, early-type spirals:</p>
```

```
    <pre class="dachs-ex-tapquery literal-block" property="query">
SELECT target_name, accref, hubtyp, magg
FROM califadr3.cubes
  NATURAL JOIN califadr3.objects
WHERE hubtyp in ('S d', 'S cd', 'S c')
  AND magg&lt;13
    </pre>
  </div>

  <div typeof="example"
      id="Queryagainstcoverage"
      resource="#Queryagainstcoverage">
    <h2 property="name">Query against coverage</h2>
    <p>When querying against geometric columns, in particular coverage,
    use ADQL's contains or intersect functions, like this:</p>
    <pre class="dachs-ex-tapquery literal-block" property="query">
SELECT accref, seeing
  FROM cars.images
      WHERE 1=INTERSECTS(coverage, circle('', 34, -4, 2))
      ORDER BY seeing
    </pre>
    <p>Of course, this concerns all SIAP and SSAP tables
    (<a href="/tableinfo/cars.images">&#x2197;</a
    ><em class="dachs-ex-taptable" property="table">cars.images</em> only
    standing as an example here) as well as
    <a href="/tableinfo/ivoa.obscore">&#x2197;</a
    ><em property="table">ivoa.obscore</em>
    </p>
  </div>
</body>
</html>
```

A Changes from TAP-1.0 to TAP-1.1

A.1 General Improvements and Clarifications

- Added updated IVOA architecture diagram showing role of TAP (1.1).
- Clarified dependency on minor versions of related standards with some recommendations on using newer versions (1.1).
- Removed text that duplicates material from DALI.
- Rewrote the introduction with some basic use cases to help define the scope and tell readers what TAP is supposed to accomplish (1).

A.2 New Features (including changes from related specifications)

- Added example use of UWS-1.1 blocking requests (5.1.3).
- Added an example for `examples` (5.3).

A.3 Removed specifications

- Removed remaining uses and examples of PQL; replaced one example with something more clearly non-standard.
- Removed REQUEST and VERSION parameters from interface. Removed obsolete REQUEST=doQuery from examples (2.7).

A.4 ADQL Clarifications

- Clarified that the QUERY param is intended for use with other values of LANG and is not reserved for ADQL only. Removed text concerning the case sensitivity of QUERY value. (2.7.2)
- Defer to other standards concerning required and optional ADQL geometric functions (ADQL specifies, TAPRegExt describes). Clarified that DALI timestamp format support is required only in services where it can be used in queries. (2.7.2)
- Clarified the relationship of MAXREC and TOP (in ADQL) and the overflow indicator and that MAXREC always overrides limitations in the query (e.g. TOP in an ADQL query). (2.7.4)
- Removed language that somehow defined or restricted usage of ADQL constructs in favour of just referring to the AQDL spec. Clarified use of serialisation rules for extended types defined in DALI.

A.5 Datatype mapping

- Added arraysize and xtype to TAP_SCHEMA to allow the complete VOTable arraysize syntax and specified that the effective datatype in TAP is specified as in VOTable using datatype,arraysize, and xtype (4.3).
- Recommend use of VOTableType in VOSI-tables output (2.5).

A.6 TAP_SCHEMA Changes

- Added schema_index, table_index and column_index to TAP_SCHEMA(4).
- Added paragraph specifying allowed values for TAP_SCHEMA.tables.table_type (4.2).
- Removed explicit datatype/arraysize/xtype from TAP_SCHEMA description in favour of string and integer. Specified which integers are actually booleans (0 or 1). Added list of foreign keys for TAP_SCHEMA relational model. Clarified use of delimited identifiers for column names in TAP_SCHEMA.
- Clarified "size" can be used for 1-dimension arrays but just does not carry any info about them being variable-length (which is only in the new arraysize column). (4.3)
- Added advice that the "size" column TAP_SCHEMA.columns must always be used as a delimited identifier because it is a reserved word in many RDBMS servers. Note: "size" is now deprecated (will be removed in the next major version). (4.3)
- Improved guidance for allowed table names for UPLOAD, clarified that multiple UPLOAD parameters accumulate. Clarified that services must support delimited identifiers. Advise that services should assign unique column names in cases where they generate the name. (2.7.6)

A.7 VOSI/UWS related changes

- Added details and VOSI-capabilities example to clarify that there is one accessURL for a TAP service and multiple authentication methods must co-exist in that single URL (2.4). Clients are expected to append the TAP-specific resource names (async and sync) to execute queries (2).
- Clarified that VOSI-availability is no longer restricted to a specific name or location (2).
- Added explicit reference to VOSI-tables. Updates capabilities example to describe a tables-1.1 capability (2).

B Detailed Changes from Previous Versions

B.1 PR-TAP-20190420

Modified the resources section to only allow the fixed name resources on a single base URL. Adapted the VOSI-capabilities example and explanation to demonstrate the single base URL design and specify the meaning of having zero or more securityMethod(s). Removed all use to the previously proposed UWSRegExt.

Added explicit mention of SSO-2.0 in related standards.

B.2 PR-TAP-1.1-20180830

Added reference to UWSRegExt Note and clarified the use of this prototype in VOSI-capabilities.

B.3 PR-TAP-1.1-20180416

Added updated IVOA architecture diagram showing role of TAP.

Removed language that somehow defined or restricted usage of ADQL constructs in favour of just referring to the AQDL spec. Clarified use of serialisation rules for extended types defined in DALI.

Removed explicit datatype/arraysize/xtype from TAP_SCHEMA description in favour of string and integer. Specified which integers are actually booleans (0 or 1). Added list of foreign keys for TAP_SCHEMA relational model. Clarified use of delimited identifiers for column names in TAP_SCHEMA. Added schema_index column. Clarified when the `"size"` column in TAP_SCHEMA can contain a value. Clarified that both arraysize and `"size"` must be null for scalar numeric columns.

Fixed use of double-quotes which misbehaved inside tables.

Fixed BasicAA security method in capabilities example. Included use of prototype UWSRegExt interface tags in capabilities example and removed use of separate standardID values for async and sync.

Fixed numerous typos and gramnmatical errors.

Removed obsolete REQUEST=doQuery from examples.

Added explicit reference to VOSI-tables. Updated capabilities example to describe a tables-1.1 capability.

Clarified dependency on minor versions of related standards with some recommendations on using newer versions.

B.4 PR-TAP-1.1-20170830

Added an example for `examples`.

Clarified that the QUERY param is intended for use with other values of LANG and is not reserved for ADQL only. Removed text concerning the case sensitivity of QUERY value.

Removed remaining uses of PQL; replaced one example with something more clearly non-standard.

Removed restriction from previous WD that the “size” column must be null for variable length arrays. In fact, “size” can be used for 1-dimension arrays but just does not carry any info about them being variable-length (which is only in the new arraysize column).

Changed language about mandatory ADQL geometry function support back to optional (should in the case where the tables contain spatial coordinates) so TAP now recommends a set of functions to support and notes others are simply optional (or not supported in the case of REGION). Removed the comment about use of point args to INTERECTS (belongs in ADQL). Clarified that DALI timestamp format support is required only in services where it can be used in queries.

Removed some VOTable content and reference DALI. Explicitly relaxed datatype and arraysize used in TAP_SCHEMA tables. Fixed various cross-references and typos.

Added example use of UWS-1.1 blocking requests.

B.5 WD-TAP-1.1-20170707

Changed arraysize in TAP_SCHEMA to allow the complete VOTable arraysize syntax and specified that the effective datatype in TAP is specified as in VOTable using datatype,arraysize, and xtype. Recommend use of VOTableType in VOSI-tables output.

Clarified required and optional ADQL geometric functions.

Format tables so column headers are bold.

Added paragraph specifying allowed values for TAP_SCHEMA.tables.table_type.

Changed the principal, indexed, and std columns in TAP_SCHEMA.columns to boolean since we now use the VOTabletype system.

Fixed URL to schema in VOSI-capabilities example.

B.6 WD-TAP-1.1-20161011

Removed details of mapping database and VOTable data types and refer to DALI instead.

Strongly recommend that VOSI resources allow anonymous access.

Relax restrictions on column names in uploaded tables; clarify that services must support delimited identifiers. Advise that services should assign unique column names in cases where they generate the name.

B.7 WD-TAP-1.1-20160428

Completed the mapping table from VOTable to RDBMS datatypes using DALI-1.1 xtype values.

Added details and VOSI-capabilities example for providing multiple resources with different authentication requirements. Clarified that VOSI-availability is no longer restricted to a specific name or location.

B.8 WD-TAP-1.1-20150930

Clarified that MAXREC always overrides limitations in the query (e.g. TOP in an ADQL query).

Clarified that services are not required to support queries that reference tables in different schema. This is primarily to allow the TAP_SCHEMA to be implemented in a different server from the content.

Completed the references section.

B.9 Changes from TAP-1.0

Added table_index and column_index to TAP_SCHEMA.

Clarified the relationship of MAXREC and TOP (in ADQL) and the overflow indicator.

Added advice that the size column TAP_SCHEMA.columns must always be used as a delimited identifier because it is a reserved word in many RDBMS servers. Added arraysize column to TAP_SCHEMA.columns to replace size and deprecated size (which will be removed in the next major version).

Removed REQUEST and VERSION parameters from interface.

Restructured the document and removed text that duplicates material from DALI. Rewrite the overly long introduction with some basic use cases to help define the scope and tell readers what TAP is supposed to accomplish.

Made clarifications: restricted allowed table names for UPLOAD, clarified that multiple UPLOAD parameters accumulate, deprecated the size column in TAP_SCHEMA.columns and added advice to quote it as a delimited identifier, made presence of a TABLE element on VOTable output only required for successful queries, added optional DALI-examples endpoint (text TBD).

Defined standardID values for the async and sync resource types and explicitly allow for multiple of each resource (typically to support authentication). The fixed paths /async and /sync are still required and are to provide anonymous query access, which should be compatible with existing services.

References

Arviset, C., Gaudet, S. and the IVOA Technical Coordination Group (2010), 'IVOA architecture', IVOA Note.
http://www.ivoa.net/documents/Notes/IVOAArchitecture

Bradner, S. (1997), 'Key words for use in RFCs to indicate requirement levels', RFC 2119.
http://www.ietf.org/rfc/rfc2119.txt

Demleitner, M., Dowler, P., Plante, R., Rixon, G. and Taylor, M. (2012a), 'TAPRegExt: a VOResource Schema Extension for Describing TAP Services Version 1.0', IVOA Recommendation 27 August 2012, arXiv:1402.4742.
http://doi.org/10.5479/ADS/bib/2012ivoa.spec.0827D

Demleitner, M., Dowler, P., Plante, R., Rixon, G. and Taylor, M. (2012b), 'TAPRegExt: a VOResource schema extension for describing TAP services, version 1.0', IVOA Recommendation.
http://www.ivoa.net/documents/TAPRegExt

Demleitner, M., Harrison, P., Molinaro, M., Greene, G., Dower, T. and Perdikeas, M. (2014), 'IVOA Registry Relational Schema Version 1.0', IVOA Recommendation 08 December 2014, arXiv:1510.02275.
http://doi.org/10.5479/ADS/bib/2014ivoa.spec.1208D

Dowler, P., Demleitner, M., Taylor, M. and Tody, D. (2017), 'Data Access Layer Interface Version 1.1', IVOA Recommendation 17 May 2017.
https://ui.adsabs.harvard.edu/abs/2017ivoa.spec.0517D

Graham, M., Rixon, G., Dowler, P., Major, B., Grid and Web Services Working Group (2017), 'IVOA Support Interfaces Version 1.1', IVOA Recommendation 24 May 2017.
http://doi.org/10.5479/ADS/bib/2017ivoa.spec.0524G

Harrison, P. A. and Rixon, G. (2016), 'Universal Worker Service Pattern Version 1.1', IVOA Recommendation 24 October 2016.
http://doi.org/10.5479/ADS/bib/2016ivoa.spec.1024H

Major, B., Rixon, G., Schaaff, A. and Taffoni, G. (2016), 'IVOA single-sign-on profile: Authentication mechanisms, version 2.0', IVOA Proposed Recommendation.
http://www.ivoa.net/documents/SSO/20160930/PR-SSOAuthMech-2.0-20160930.html

Ochsenbein, F., Taylor, M., Williams, R., Davenhall, C., Demleitner, M., Durand, D., Fernique, P., Giaretta, D., Hanisch, R., McGlynn, T., Szalay, A. and Wicenec, A. (2013), 'VOTable Format Definition Version 1.3', IVOA Recommendation 20 September 2013.
http://doi.org/10.5479/ADS/bib/2013ivoa.spec.0920O

Osuna, P., Ortiz, I., Lusted, J., Dowler, P., Szalay, A., Shirasaki, Y., Nieto-Santisteban, M. A., Ohishi, M., O'Mullane, W., VOQL-TEG Group and VOQL Working Group. (2008), 'IVOA Astronomical Data Query Language Version 2.00', IVOA Recommendation 30 October 2008, arXiv:1110.0503.
http://doi.org/10.5479/ADS/bib/2008ivoa.spec.1030O

Plante, R., Graham, M., Rixon, G. and Taffoni, G. (2010), 'IVOA Credential Delegation Protocol Version 1.0', IVOA Recommendation 18 February 2010, arXiv:1110.0509.
http://doi.org/10.5479/ADS/bib/2010ivoa.spec.0218P

Plante, R., Stébé, A., Benson, K., Dowler, P., Graham, M., Greene, G., Harrison, P., Lemson, G., Linde, T. and Rixon, G. (2010), 'VODataService: a VOResource Schema Extension for Describing Collections, Services Version 1.1', IVOA Recommendation 02 December 2010, arXiv:1110.0516.
http://doi.org/10.5479/ADS/bib/2010ivoa.spec.1202P

Shafranovich, Y. (2005), 'Common Format and MIME Type for Comma-Separated Values (CSV) Files', IETF RFC 4180.
https://tools.ietf.org/html/rfc4180

Tody, D., Micol, A., Durand, D., Louys, M., Bonnarel, F., Schade, D., Dowler, P., Michel, L., Salgado, J., Chilingarian, I., Rino, B., de Dios Santander, J. and Skoda, P. (2011), 'Observation Data Model Core Components, its Implementation in the Table Access Protocol Version 1.0', IVOA Recommendation 28 October 2011, arXiv:1111.1758.
http://doi.org/10.5479/ADS/bib/2011ivoa.spec.1028T

University of Minnesota Gopher Team (1993), 'Tab separated values', IANA, MIME Media Types.
https://www.iana.org/assignments/media-types/text/tab-separated-values