International
Virtual
Observatory
Alliance

Table Access Protocol Version 1.1

IVOA Working Draft 2016-04-28

Working group
Data Access Layer Working Group
This version
http://www.ivoa.net/documents/TAP/20160428
Latest version
http://www.ivoa.net/documents/TAP
Previous versions
TAP-1.0

Author(s)
Patrick Dowler, ...
Editor(s)
Patrick Dowler

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 (Ortiz et al., 2008) 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 Working Draft for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use IVOA Working Drafts as reference materials or to cite them as other than "work in progress".

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

Contents

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, G. Rixon, 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 (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

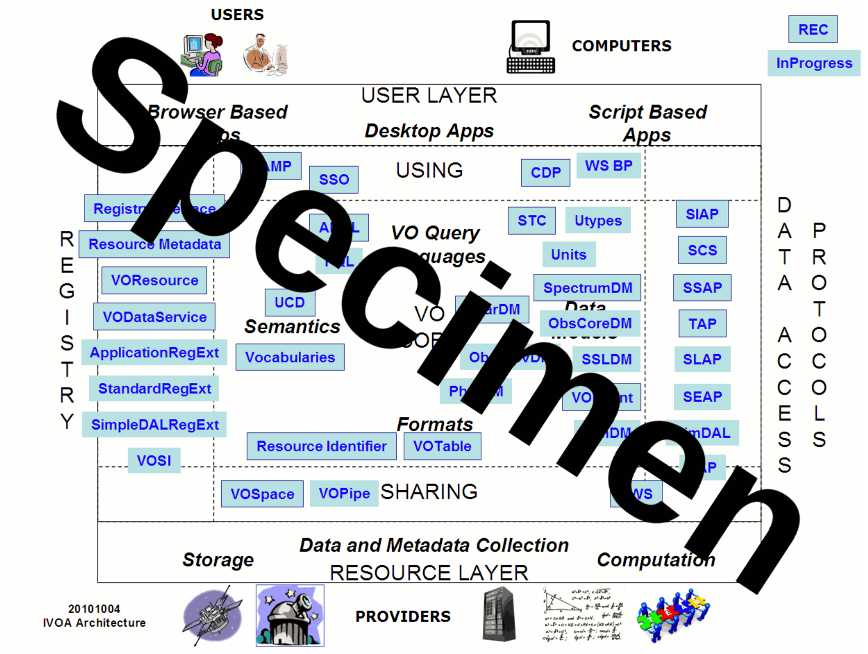

Figure 1: Architecture diagram for this document

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

NOTE: not in TAP-1.0

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

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.

1.2.5 ADQL Queries

The Astronomical Data Query Language (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.

1.2.6 Other Query Languages

A TAP service must be able to support use of other query languages, such pass-through of native SQL directly to an underlying DBMS or simple key-vale (parameter-based) constraints. 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.

1.2.7 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).

1.2.8 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
sync	/sync	must
async	/async	must
sync	service specific	may (alternate authentication method)
async	service specific	may (alternate authentication method)
VOSI-availability	service specific	must
VOSI-availability	service specific	may (alternate authentication method)
VOSI-capabilities	/capabilities	must
VOSI-capabilities	service specific	may (alternate authentication method)
VOSI-tables	/tables	should
VOSI-tables	service specific	may (alternate authentication method)
DALI-examples	/examples	should
DALI-examples	service specific	may (alternate authentication method)

At least one set of sync and async resources must be named /sync and /async respectively for backwards compatibility with TAP-1.0 (which required these names. Other sync and async resources may have service specific names. As required by (Dowler et al., 2013), all resources except the VOSI-availability must be siblings of the VOSI-capabilities resource.

If a TAP service allows anonymous access, the fixed name resources and VOSI-availability must be used for anonymous access; other resource names may be used for authenticated access. If the entire service requires authentication then the fixed names may be used for authenticated access.

The web resource at the root of the tree must represent the service as a whole. This specification defines no standard representation for this root resource. Implementations may provide a representation, or may return a '404 not found' response to requests for the root web-resource. One possible representation is an HTML page describing the scientific usage and content of the service. TAP clients must not depend on a specific representation of the root web-resource.

2.1 {sync}

A TAP service must provide one or more web resources that represents the results of synchronous query execution. The sync resources must conform to the general rules for DALI-sync (Dowler et al., 2013) 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. Representations of results of queries if 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 {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 (Dowler et al., 2013) resources and thus represent a job-list as specified in (Harrison and Rixon, 2010).

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 the 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 availability

The use of the VOSI-availability resource is described in (Dowler et al., 2013).

2.4 /capabilities

The TAP-1.0 standard is identified using

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

For TAP-1.1 we define new standard identifiers for each of the features. Asynchronous query resources (section 2.2) are described by standardID:

```
ivo://ivoa.net/std/TAP#async-1.1
```

Synchronous query resources (section 2.1) are described by standardID:

```
ivo://ivoa.net/std/TAP#sync-1.1
```

In TAP-1.0 the base URL was described with a single standard identifier; in TAP-1.1 and beyond, individual resources are described with their own standardID. This allows service providers to describe multiple resources that deliver the specified feature (e.g. with different authentication methods) in the VOSI-capabilities resource.

The use of the VOSI-capabilities resource is described in (Dowler et al., 2013).

For example, the returned capabilities document for a service supporting TAP might look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

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

  <vosi:capability standardID="ivo://ivoa.net/std/TAP">
    <interface xsi:type="vs:ParamHTTP" role="std" version="1.0">
      <accessURL use="base"> http://example.net/myTAP </accessURL>
    </interface>
    <interface xsi:type="vod:ParamHTTP" role="std" version="1.0">
      <accessURL use="base">https://example.net/myTAP</accessURL>
      <securityMethod standardID="ivo://ivoa.net/sso#tls-with-certificate" />
    </interface>

  </vosi:capability>

  <vosi:capability standardID="ivo://ivoa.net/std/TAP#async-1.1">
    <interface xsi:type="vs:ParamHTTP" role="std" version="1.1">
      <accessURL use="base"> http://example.net/myTAP/async </accessURL>
    </interface>
    <interface xsi:type="vod:ParamHTTP" role="std" version="1.1">
      <accessURL use="base">http://example.net/myTAP/auth−async</accessURL>
      <securityMethod standardID="http://www.w3.org/Protocols/HTTP/1.0/spec.html#BasicAA" />
    </interface>
    <interface xsi:type="vod:ParamHTTP" role="std" version="1.1">
      <accessURL use="base">https://example.net/myTAP/async</accessURL>
      <securityMethod standardID="ivo://ivoa.net/sso#tls-with-certificate" />
    </interface>
  </vosi:capability>

  <vosi:capability standardID="ivo://ivoa.net/std/TAP#sync-1.1">
    <interface xsi:type="vs:ParamHTTP" role="std" version="1.1">
      <accessURL use="base"> http://example.net/myTAP/sync </accessURL>
    </interface>
    <interface xsi:type="vod:ParamHTTP" role="std" version="1.1">
      <accessURL use="base">http://example.net/myTAP/auth-sync</accessURL>
      <securityMethod standardID="http://www.w3.org/Protocols/HTTP/1.0/spec.html#BasicAA" />
    </interface>
    <interface xsi:type="vod:ParamHTTP" role="std" version="1.1">
      <accessURL use="base">https://example.net/myTAP/sync</accessURL>
      <securityMethod standardID="ivo://ivoa.net/sso#tls-with-certificate" />
    </interface>
  </vosi:capability>

  <vosi:capability standardID="ivo://ivoa.net/std/VOSI#capabilities">
    <interface xsi:type="vs:ParamHTTP">
      <accessURL use="full">
        http://example.net/myTAP/capabilities </accessURL>
    </interface>
  </vosi:capability>

  <vosi:capability standardID="ivo://ivoa.net/std/VOSI#availability">
    <interface xsi:type="vs:ParamHTTP">
      <accessURL use="full">
        http://example.net/myTAP/availability
      </accessURL>
    </interface>
  </vosi:capability>

  <vosi:capability standardID="ivo://ivoa.net/std/VOSI#tables">
    <interface xsi:type="vs:ParamHTTP">
      <accessURL use="full"> http://myarchive.net/myTAP/tables </accessURL>
    </interface>
    <interface xsi:type="vs:ParamHTTP">
      <accessURL use="full"> https://myarchive.net/myTAP/tables </accessURL>
        <securityMethod standardID="ivo://ivoa.net/sso#tls-with-certificate" />
    </interface>
    <interface xsi:type="vs:ParamHTTP">
      <accessURL use="full"> http://myarchive.net/myTAP/auth-tables </accessURL>
```

```
      <securityMethod standardID="http://www.w3.org/Protocols/HTTP/1.0/spec.html#BasicAA" />
    </interface>
  </vosi:capability>

</vosi:capabilities>
```

The service capabilities must be accessible from a web resource with relative URL /capabilities that is a direct child of the root web resource. The /capabilities resource must be accessible via the http GET method. The content is described by [8].

2.5 /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 must be accessible via the http GET method. The content is described by (Plante et al., 2010) and is equivalent to the metadata from the TAP_SCHEMA described in section 4.

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

2.6 /examples

A 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 (Dowler et al., 2013). 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". Suitable HTML elements holding these include span, or a (which would allow linking to further information on the table).

When using elements with src or href attributes to carry the property attributes, note that the element content must be repeated in a content attribute, as otherwise RDFa clients would interpret the embedded link rather than the element content as the object in the triple.

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 (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. For example, if a request carries LANG=ADQL then the SELECT parameter (from PQL) is spurious. 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 LANG and the client must provide a value with REQUEST=doQuery. The only standard values for the LANG parameter is ADQL (a required language). Support for other languages and the LANG value to use with them is described in the service capabilities.

For example, an ADQL query would be performed with

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

A PQL query would be performed with

```
LANG=PQL
<PQL-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 the ADQL query. It may also be used to specify the query for other values of LANG (e.g. LANG=<some RDBMS-specific SQL variant>) which are not specified in this document but may be described in the service capabilities.

A service must support the QUERY parameter because ADQL is a required language. The case sensitivity of the query string is defined solely by the query language specification. In the case of ADQL 2.0, for example, the query is not case sensitive except for character literals; schema, table, and column names, function names, and other ADQL keywords are not case sensitive.

Within the ADQL query, the service must support the use of timestamp values as described in (Dowler et al., 2013).

TAP-1.0 text about ADQL region functions:

If the tables that are queried through a service contain columns with spatial coordinates and the service supports spatial querying via the ADQL "region" constructs, the service must support the INTERSECTS function and it must support the following geometry functions: REGION, POINT, BOX, CIRCLE, COORD1, COORD2, COORDSYS. Support for the AREA, CONTAINS, and POLYGON functions are optional. If the service supports the REGION function, it must support region encoding in STC-S format (see section 6); the extent of STC-S support within the REGION function is left up to the implementation. Coordinate system specification for POINT, BOX, CIRCLE, and POLYGON must use values from STC-S as described in ???.

Proposed TAP-1.1 text:

INTERSECTS and CONTAINS are required. POINT, CIRCLE, and POLYGON are required. POINT (and point-valued columns) cannot be used as an argument to INTERSECTS. AREA, CENTROID, COORDSYS, COORD1, and COORD2 are optional. REGION is not supported (and hopefully gone from ADQL-2.1).

Note: Although it is allowed by the ADQL 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.

2.7.3 FORMAT and RESPONSEFORMAT

The RESPONSEFORMAT parameter is fully described in (Dowler et al., 2013). For backwards compatibility, TAP-1.1 must also accept the FORMAT parameter as equivalent to RESPONSEFORMAT.

2.7.4 MAXREC

The MAXREC parameter and its effect on the query result is fully described in (Dowler et al., 2013). 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 FORMAT=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 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 (Dowler et al., 2013).

2.7.6 UPLOAD

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

If table upload 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 (Ortiz et al., 2008)). Uploaded tables must be referred to in queries as TAP_UPLOAD.<tablename>, where <tablename> is the 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.

The (Dowler et al., 2013) 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 (TBD: how to declare this?), 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.

3 Use of VOTable

The (Ochsenbein et al., 2013) format 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).

3.1 INFO elements

The RESOURCE element must contain an INFO element with attribute name="QUERY_STATUS" indicating the success of the operation. For RESOURCE elements that contain a TABLE element, this INFO element must appear lexically before the TABLE. The following values are defined for this INFO element's value attribute:

"OK", meaning that the query executed successfully and a result table is included in the resource

"ERROR", meaning that an error was detected at the level of the TAP protocol or the query failed to execute

The content of the INFO element conveying the status should be a message suitable for display to the user describing the status.

```
<INFO name="QUERY_STATUS" value="OK"/>

<INFO name="QUERY_STATUS" value="OK">Successful query</INFO>
```

```
<INFO name="QUERY_STATUS" value="ERROR">
   value out of range in POS=45,91
</INFO>
```

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_STATUS" value="ERROR">
    unrecognized operation
</INFO>
<INFO name="SPECIFICATION" value="TAP"/>
<INFO name=’’VERSION’’ value=’’1.0’’/>
<INFO name="REQUEST" value="doQuery"/>
<INFO name="baseUrl" value="http://webtest.aoc.nrao.edu/ivoa-dal"/>
<INFO name="serviceVersion" value="1.0"/
...
</RESOURCE>
```

If an overflow occurs (result exceeds MAXREC), the service must close the table and append another INFO element to the RESOURCE (after the TABLE) with name="QUERY_STATUS" and the value="OVERFLOW".

```
<RESOURCE type=’’results’’>
<INFO name="QUERY_STATUS" value="OK"/>
...
<TABLE>...</TABLE>
<INFO name="QUERY_STATUS" value="OVERFLOW"/>
</RESOURCE>
```

In the above example, the TABLE should have exactly MAXREC rows.

If an error occurs while writing the rows of the VOTable, the service must close the table and append another INFO element to the RESOURCE, after the TABLE, with name="QUERY_STATUS" and the value="ERROR".

```
<RESOURCE type=’’results’’>
<INFO name="QUERY_STATUS" value="OK"/>
...
<TABLE>...</TABLE>
<INFO name="QUERY_STATUS" value="ERROR" />
</RESOURCE>
```

The content of these trailing INFO elements is optional and intended for users; client software should not depend on it.

Thus, one INFO element with name="QUERY_STATUS" and value="OK" or value="ERROR" must be included before the TABLE. If the TABLE does not contain the entire query result, one INFO element with value="OVERFLOW" or value="ERROR" must be included after the table.

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 FORMAT (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) from the query and the datatype, arraysize, and xtype attributes of FIELD elements must be set using the mapping specified in section 3.5. The xtype attribute in the output must match the datatype for the column in the TAP_SCHEMA.

VOTable structure follows the rules in section 2.9 and must be returned with an allowed VOTable MIME type (application/x-votable+xml or text/xml). If the RESPONSEFORMAT parameter (section 2.7.3) of the request specified a specific VOTable MIME type, the requested MIME type must be used in the HTTP response.

CSV 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 commas. 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 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, n.d.).

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 (University of Minnesota Gopher Team, n.d.). 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.

Errors in the use of the TAP protocol, including both invalid requests and failure of the service to complete valid requests.

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 joblist 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 for TAP errors must be VOTable documents; any resultformat specified in the request is ignored. 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. The exception condition must be described to the client using a status code in the VOTable header. Section 2.9 specifies the use of VOTable for error documents in TAP services.

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.

If the output format is VOTable, section 2.9.1 describes the method by which the overflow is reported. No method of reporting an overflow is defined for formats other than VOTable.

3.5 Mapping Table Datatypes

This section describes the bi-directional mapping between VOTable and RDBMS + geometric datatypes and extends the basic rules for serialising such values in VOTable described in (Dowler et al., 2013). These rules apply to input tables supplied via an UPLOAD parameter (see section 2.7.6) and to result tables after successful query execution.

For input tables, the name attribute of the FIELD element is used as the column name. This name must be a legal ADQL column name (i.e., a string following the regular identifier production (Ortiz et al., 2008). For result tables, the column name or an alias specified in the query is used to set the name attribute of the FIELD elements in the output table (the alias overrides the normal column name).

The mapping between VOTable datatype, arraysize, and xtype attributes and RDBMS datatypes are as follows:

datatype	arraysize	xtype	RDBMS datatype	required
char	[1]		CHAR or CHAR(1)	yes
char	*		VARCHAR(*) or CLOB	no
char	n		CHAR(n)	yes
char	n*		VARCHAR(n)	yes
unsignedByte	[1]		BINARY(1)	yes
unsignedByte	*		VARBINARY(*) or BLOB	no
unsignedByte	n		BINARY(n)	yes
unsignedByte	n*		VARBINARY(n)	yes
short	[1]		SMALLINT?	yes
short	*		implementation-specific	no
short	n		implementation-specific	no
short	n*		implementation-specific	no
int	[1]		INTEGER	yes
int	*		implementation-specific	no
int	n		implementation-specific	no
int	n*		implementation-specific	no
long	[1]		BIGINT	yes
long	*		implementation-specific	no
long	n		implementation-specific	no
long	n*		implementation-specific	no
float	[1]		REAL	yes
float	*		implementation-specific	no
float	n		implementation-specific	no
float	n*		implementation-specific	no
double	[1]		DOUBLE [PRECISION]	yes
double	*		implementation-specific	no
double	n		implementation-specific	no
double	n*		implementation-specific	no
char	n or n*	timestamp	implementation-specific	yes
double	2	point	implementation-specific	no
double	3	circle	implementation-specific	no
double	*	polygon	implementation-specific	no
float	2	point	implementation-specific	no
float	3	circle	implementation-specific	no
float	*	polygon	implementation-specific	no
short	2	interval	implementation-specific	no
int	2	interval	implementation-specific	no
long	2	interval	implementation-specific	no
float	2	interval	implementation-specific	no
double	2	interval	implementation-specific	no

The default mapping of data types are shown above (no arraysize or xtype). If the xtype attribute is set, this is the preferred internal datatype. If xtype is not set, then the datatype and arraysize indicate the most suitable internal datatype. Note that the RDBMS type is not normative. Implementations SHOULD try to make sure that the actual types chosen are at least signature-compatible with the recommended types (i.e., integers should remain integers, floating-point values floating-point values, etc.), such that clients can reliably write queries against uploaded tables.

In the arraysize column above, [1] means the arraysize is not set or is set to 1, n means arraysize is set to a specific value, * means arraysize="*", and n* means arraysize="n*" (variable size up to length n). Support for arrays of numeric values is not required but may be implemented and described. However, such values can only be selected unless the query language or a custom extension or function is provided.

Binary values (unsignedByte in VOTable, BINARY, VARBINARY, or BLOB in ADQL) can be expressed as specified by the VOTable standard. By default, VOTable allows them to be written as an array of decimal numbers, e.g. 12 56 0 255 0 0 255 (one number per byte value).

TIMESTAMP values are specified as described in (Dowler et al., 2013). The xtype=timestamp attribute must be specified in an uploaded VOTable in order for the values to be inserted in a column of type TIMESTAMP; without the xtype, the values would be inserted into a CHAR(n) or VARCHAR column.

POINT, CIRCLE, and POLYGON values are specified as arrays of double values as described in (Dowler et al., 2013). The xtype attribute must be specified in an uploaded VOTable in order for the array values to be parsed and treated as geometric values and thus to be used with some of the ADQL features.

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 the VODataService (Plante 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.

The qualified names in the tables of the TAP schema must follow the rules defined in section 2.4. The names must be stated in a form that is acceptable as an operand of a query.

All columns in the TAP_SCHEMA tables are of type VARCHAR except for size, principal, indexed, and std (in Columns) which are INTEGER values.

Implementors are permitted to 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 was last modified.

4.1 Schemas

The table TAP_SCHEMA.schemas must contain the following columns:

column name	datatype	not-null
schema_name	VARCHAR	true
utype	VARCHAR	false
description	VARCHAR	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	datatype	not-null
schema_name	VARCHAR	true
table_name	VARCHAR	true
table_type	VARCHAR	true
utype	VARCHAR	false
description	VARCHAR	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 (quoted identifier in (Ortiz et al., 2008)) then the value must include the quotes.

The table_index is used to recommended 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	datatype	not-null
table_name	VARCHAR	true
column_name	VARCHAR	true
datatype	VARCHAR	true
arraysize	INTEGER	false
"size"	INTEGER	false
description	VARCHAR	false
utype	VARCHAR	false
unit	VARCHAR	false
ucd	VARCHAR	false
indexed	BOOLEAN?	true
principal	BOOLEAN?	true
std	BOOLEAN?	true
column_index	false	

The table_name,column_name (pair) values must be unique.

Database data types and how they map to VOTable datatypes are described in Section xrefsec:vot-rdbms above. However, the datatype column in the TAP_SCHEMA.columns table contains the database type: either an xtype defined in (Dowler et al., 2013) or a TAPType defined in (Plante et al., 2010). [TODO: make this consistent with VOSI-tables] For columns with a datatype of BLOB or CLOB, most database systems support reference to these columns in the select clause but not in any other part of the query. Services may use these types to indicate that columns may only be selected. 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, they would specify a column with datatype CLOB in the TAP_SCHEMA and the column with URL(s) could be referenced in the SELECT clause of a query, but could not be used in the WHERE clause. The service could then process the query result and insert the URL(s) or, more likely, transform a column value (an identifier) into a URL while writing the results.

The arraysize column gives the length of variable length datatypes, for example varchar(256); this arraysize does not map exactly to the VOTable arraysize attribute because the latter can specify the size and shape of a multi-dimensional array as well as the variable size. The "size" column is retained for backwards compatiblity to TAP-1.0 and must contain the same value as arraysize. To use the size column in a query, it 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 DDL for TAP_SCHEMA) use lower case exclusively. In the next major version of TAP, the "size" column will be removed.

The “principal” flag indicates that the column is considered a core part 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 services 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” 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	datatype	not-null
key_id	VARCHAR	true
from_table	VARCHAR	true
target_table	VARCHAR	true
description	VARCHAR	false
utype	VARCHAR	false

The key_id values are unique and used only to join with the TAP_SCHEMA.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	datatype	not-null
key_id	VARCHAR	true
from_column	VARCHAR	true
target_column	VARCHAR	true

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

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 (Harrison and Rixon, 2010) and its application to TAP in section section 2.2. 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 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 fo 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. Once the returned value is COMPLETED, 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 et al., 2013)), 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 (Dowler et al., 2013)). 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 be 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:

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

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 (Harrison and Rixon, 2010)). 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.

5.1.4 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
REQUEST=doQuery
LANG=ADQL
QUERY=SELECT * FROM magnitudes as m where m.r>=10 and m.r<=16
```

In this examples, the output format defaults to VOTable; the FORMAT 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 reponse code 303) and another URL for the results, e.g.:

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

As described in (Dowler et al., 2013), clients must be prepared to follow such redirects to obtain the result.

5.1.5 Example: DALI-examples Document

TODO

A Changes from Previous Versions

A.1 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.

A.2 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.

A.3 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 quoted identifier becayse 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 pamaters 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.
URL: *http://www.ivoa.net/documents/Notes/IVOAArchitecture*

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

Dowler, P., Demleitner, M., Taylor, M. and Tody, D. (2013), ‘Data access layer interface, version 1.0’, IVOA Recommendation.
URL: *http://www.ivoa.net/documents/DALI*

Harrison, P. and Rixon, G. (2010), ‘Universal worker service pattern, version 1.0’, IVOA Recommendation.
URL: *http://www.ivoa.net/documents/UWS*

Ochsenbein, F., Williams, R., Davenhall, C., Demleitner, M., Durand, D., Fernique, P., Giaretta, D., Hanisch, R., McGlynn, T., Szalay, A., Taylor, M. and Wicenec, A. (2013), ‘Votable format definition, version 1.3’, IVOA Recommendation.
URL: *http://www.ivoa.net/documents/VOTable/*

Ortiz, I., Lusted, J., Dowler, P., Szalay, A., Shirasaki, Y., Nieto-Santisteban, M. A., Ohishi, M., O’Mullane, W., Osuna, P., the VOQL-TEG and the VOQL Working Group (2008), ‘IVOA astronomical data query language’, IVOA Recommendation.
URL: *http://www.ivoa.net/documents/latest/ADQL.html*

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 and services version 1.1’, IVOA Recommendation.
URL: *http://www.ivoa.net/documents/VODataService/*

Shafranovich, Y. (n.d.), ‘Common Format and MIME Type for Comma-Separated Values (CSV) Files’, IETF RFC 4180.
URL: *https://tools.ietf.org/html/rfc4180*

University of Minnesota Gopher Team (n.d.), ‘Tab separated values’, IANA, MIME Media Types.
URL: *https://www.iana.org/assignments/media-types/text/tab-separated-values*