International
Virtual
Observatory
Alliance

TAPRegExt: A VOResource Schema Extension for Describing TAP Services

Version 1.1

IVOA Working Draft 2015-11-17

Working Group
Registry

This version
https://www.ivoa.net/documents/TAPRegExt/20151117

Latest version
https://www.ivoa.net/documents/TAPRegExt

Previous versions
REC-1.0
PR-20110727

Author(s)
Markus Demleitner, Patrick Dowler, Ray Plante, Guy Rixon, Mark Taylor

Editor(s)
Markus Demleitner

Version Control
Revision 7165975, 2025-10-02 12:36:28 +0200

Abstract

This document describes an XML encoding standard for metadata about services implementing the table access protocol TAP (Dowler and Rixon et al., 2019), referred to as TAPRegExt. Instance documents are part of the service's registry record or can be obtained from the service itself. They deliver information to both humans and software on the languages, output formats, and upload methods supported by the service, as well as data models implemented by the exposed tables, optional language features, and certain limits enforced by the service.

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 in the IVOA document repository[1].

Contents

[1] https://www.ivoa.net/documents/

1 Introduction

The Table Access Protocol TAP (Dowler and Rixon et al., 2019) allows VO clients to send queries to remote database servers and receive the results in standard formats. In addition, it defines means to discover database schemata on the remote side, to upload data from the local disk or third-party hosts, and more. TAP builds upon a variety of other standards, premier among which is the Universal Worker Service (Harrison and Rixon, 2016), which describes how client and server can negotiate the execution of a query and the retrieval of results without having to maintain a continuous connection.

To accommodate a wide variety of requirements, the TAP specification offers implementors many choices on optional features, resource limits, or locally defined functionality. One purpose of TAPRegExt is to allow the service to communicate such choices to remote clients using the mechanisms laid down in the VO Service Interfaces standard (Graham and Rixon et al., 2017).

Clients also need to discover TAP services offering certain kinds of data. Central to this is the concept of a registry in which resources can be described and consequently discovered by users and applications in the VO. Registries receive resource descriptions as defined in the IVOA standard (Plante and Demleitner et al., 2018). In this schema, support for a standard service protocol is described as a service's capability; the associated metadata is contained within the service resource description's `<capability>` element.

TAPRegExt defines this capability element for TAP services. In the context of registering TAP services, an important role filled by TAPRegExt is the communication of supported data models to the Registry.

The specification comes with a non-normative example document[2] showing this specification's major features. It also declares a separate capability for the VOSI capabilities endpoint. The example is written as a response from a TAP service's capabilities endpoint. When embedded in a VOResource record, the capability elements would be direct children of the *vr:Resource* element.

1.1 TAPRegExt within the VO Architecture

This specification directly relates to other IVOA standards in the following ways:

[2] https://www.ivoa.net/documents/TAPRegExt/20151117/sample.xml

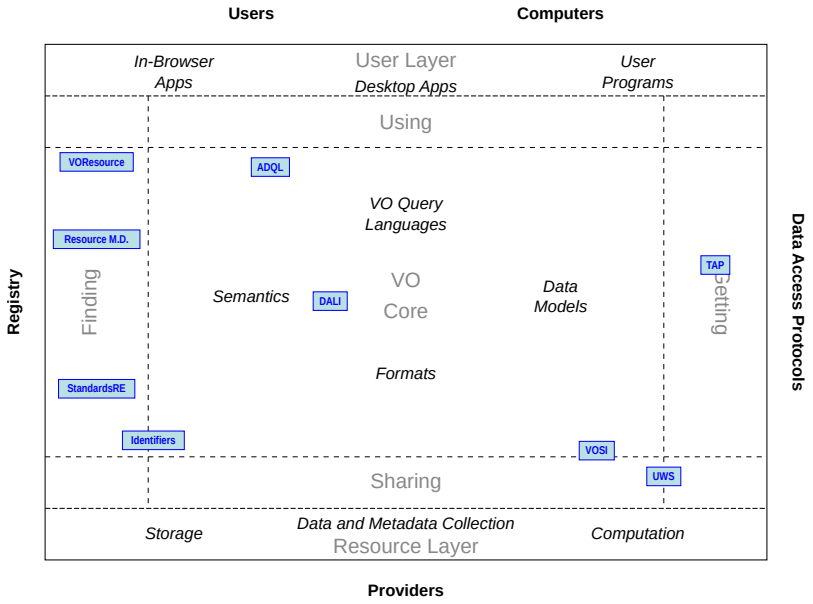

Figure 1: IVOA Architecture diagram with TAPRegExt and the related standards marked up

VOResource, v1.1 (Plante and Demleitner et al., 2018) Descriptions of services that support TAP are encoded using the VOResource XML schema. TAPRegExt is an extension of the VOResource core schema.

TAP, v1.1 (Dowler and Rixon et al., 2019) The TAP standard defines some of the concepts that TAPRegExt deals with. The TAP standard document indirectly refers to this document in the specification of its capabilities endpoint.

UWS, v1.1 (Harrison and Rixon, 2016) The UWS standard describes additional parameters the choices of which are communicated using TAPRegExt.

StandardsRegExt (Harrison and Burke et al., 2012) TAPRegExt uses the StandardKeyEnumeration mechanism introduced in StandardsRegExt to define controlled vocabularies.

This standard also relates to other IVOA standards:

IVOA Support Interfaces, v1.1 (Graham and Rixon et al., 2017) VOSI describes the standard interfaces to discover metadata about services; this document defines the response TAP services should provide on the `capabilities` endpoint described by VOSI.

IVOA defined data models Data models specified by the IVOA can define the structure of database tables holding instances of those data models. The first examples of such definitions are ObsCore (Louys and Tody et al., 2017) and RegTAP (Demleitner and Harrison et al., 2019). Services providing access to such tables declare that fact within TAPRegExt instance documents.

2 The Extension

2.1 The Schema Namespace and Location

The namespace associated with the TAPRegExt VOResource extension is

`http://www.ivoa.net/xml/TAPRegExt/v1.0`.

The namespace is unchanged from version 1.0 of this standard as no changes that could break clients are introduced.

Just like the namespace URI for the VOResource schema, the TAPRegExt namespace URI can be interpreted as a URL. Resolving it returns the current XML schema document that defines the TAPRegExt schema (cf. Appendix A).

Authors of VOResource instance documents may choose to provide a location for the VOResource XML schema document and its extensions using the *xsi:schemaLocation* attribute. While generators are free to provide any schema location (e.g., a local mirror), this specification recommends using the TAPRegExt namespace URI as its location URL, as in,

```
xsi:schemaLocation="http://www.ivoa.net/xml/TAPRegExt/v1.0
                    http://www.ivoa.net/xml/TAPRegExt/v1.0"
```

Note that you must give the `xsi:schemaLocation` of the TAPRegExt schema when the capability defined here is part of a published registry resource record as per the IVOA Registry Interface standard (Dower and Demleitner et al., 2018). This does not apply to the use in a TAP server's capabilities endpoint.

2.2 Mode-dependent Declarations

Several TAPRegExt elements have a *forMode* attribute. This allows operators to declare different behaviour of their services depending on the "mode" used to access the service, which currently means choosing between sync and async. For instance, certain output formats may produce more than one physical file, in which case it would only be available in async mode. More commonly, services may give async services much more liberal limits for total execution time or the number of rows retrievable.

Elements without a *forMode* attribute apply to all modes. The semantics of non-empty *forMode* attributes depends on what elements they are on:

- For *outputFormat* and *uploadMethod*, specifications are accumulative (i.e., you cannot retract formats or methods per mode, just add to them).
- For *retentionPeriod*, *executionDuration*, *outputLimit*, and *uploadLimit*, limits given in elements with a *forMode* attribute override limits given in elements without one for the mode specified. For these elements, only zero or one occurrences are allowed each without *forMode* and per *forMode* value[3].

This is orthogonal to limits and features that depend on the authentication status. Services varying these depending on who runs a query should adjust their capability documents accordingly (cf. sect. 2.7).

2.3 Declaring Instantiated Data Models

The IVOA defines certain data models that can be instantiated in database tables exposed by a TAP service. This allows a query built exclusively on a data model or a set of data models to work on all TAP services exposing tables instantiating the data model(s).

In TAPRegExt, a data model is identified by its IVOA identifier (Demleitner and Plante et al., 2016).

tr:DataModelType Type Schema Documentation

An IVOA defined data model, identified by an IVORN intended for machine consumption and a short label intended for human comsumption.

tr:DataModelType Type Schema Definition

```
<xs:complexType name="DataModelType" >
  <xs:simpleContent >
    <xs:extension base="xs:token" >
      <xs:attribute name="ivo-id" type="xs:anyURI" use="required" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

tr:DataModelType Attributes

ivo-id

Type	a URI: *xs:anyURI*
Meaning	The IVOID of the data model.
Occurrence	required

[3]This constraint is hard to express in XML Schema and is thus not enforced by it; its purpose is to make sure that clients will find unique limits.

2.4 Languages Supported

TAP services may offer a variety of query languages. In TAPRegExt, the *language* element allows the communication of what languages are available on a service. TAP defines values of the LANG parameter to have either the form <name>-<version> or the form <name>, where the latter form leaves the choice of the version to the server. Therefore, a language is defined using a name and one or more versions.

The recommended way to associate larger amounts of documentation with a language entry in a capability element is via registration of the language using the mechanisms defined in StdRegExt (Harrison and Burke et al., 2012) and associating the registry record with the language element through the latter's *ivo-id* attribute. The IVOID for the only language mandatory for TAP services, ADQL 2.0, is ivo://ivoa.net/std/ADQL#v2.0. .

The type of the *ivo-id* attribute on version is *xs:anyURI* as opposed to *vr:IdentifierURI* since the latter does not allow fragment identifiers in VOResource 1.0. The description constrains the value to be an IVOID (i.e., a URI with a schema of ivo:), though. The same reasoning applies to the *ivo-id* attributes of *outputFormat* and *uploadMethod*.

tr:Language Type Schema Documentation

A query language supported by the service.

Each language element can describe one or more versions of a language. Either name alone or name-version can be used as values for the server's LANG parameter.

tr:Language Type Schema Definition

```
<xs:complexType name="Language" >
  <xs:sequence >
    <xs:element name="name" type="xs:NCName" />
    <xs:element name="version" type="tr:Version" minOccurs="1"
             maxOccurs="unbounded" />
    <xs:element name="description" type="xs:token" minOccurs="0" />
    <xs:element name="languageFeatures"
             type="tr:LanguageFeatureList"
             minOccurs="0"
             maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
```

tr:Language Metadata Elements

Element *name*

Type	a prefixless XML name
Meaning	The name of the language without a version suffix.
Occurrence	required

Element *version*

Type	a string with optional attributes
Meaning	A version of the language supported by the server.

Occurrence required; multiple occurrences allowed.

Element *description*

Type string: *xs:token*

Meaning A short, human-readable description of the query language.

Occurrence optional

Element *languageFeatures*

Type composite: *tr:LanguageFeatureList*

Meaning Optional features of the query language, grouped by feature type.

Occurrence optional; multiple occurrences allowed.

Comment This includes listing user defined functions, geometry support, or similar concepts.

tr:Version Type Schema Documentation

One version of the language supported by the service.

If the service supports more than one version of the language, include multiple version elements. It is recommended that you use a version numbering scheme like MAJOR.MINOR in such a way that sorting by ascending character codes will leave the most recent version at the bottom of the list.

tr:Version Type Schema Definition

```
<xs:complexType name="Version" >
  <xs:simpleContent >
    <xs:extension base="xs:token" >
      <xs:attribute name="ivo-id" type="xs:anyURI" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

tr:Version Attributes

ivo-id

Type a URI: *xs:anyURI*

Meaning An optional IVOID of the language.

Occurrence optional

Comment To more formally define a language supported by a service, a resource record for the language can be created, either centrally on the Registry of Registries or by other registry operators. When such a record exists, the ivo-id attribute of language should point to it.

Query languages may support optional features. For ADQL, the most prominent of those are user-defined functions, i.e., functions not defined in the language standard but added by the operators of the service, and geometry functions. Such optional features may be communicated to the service client in *tr:languageFeatures* elements.

Each such list is labelled with a *type* attribute indicating the type of language option being described. This string should be an IVOID whose semantics in this context, along with the semantics of the content of its descendant

feature/form elements, can be documented in association with the language in question.

TAPRegExt itself defines the following feature types:

ivo://ivoa.net/std/TAPRegExt#features-udf

Each feature declares a user-defined ADQL (or similar) function supported. The content of the *form* element must be the signature of the function, written to match the `signature` nonterminal in the following grammar:

```
signature ::= <funcname> <arglist> "->" <type_name>
funcname  ::= <regular_identifier>
arglist   ::= "(" <arg> { "," <arg> } ")"
arg       ::= <regular_identifier> <type_name>
```

The `type_name` nonterminal is not defined by the ADQL grammar in version 2.0. For the purposes of TAPRegExt, it is sufficient to assume it expands to "some sort of SQL type specifier" (which may include spaces and parentheses). For an enumeration of common types in ADQL, refer to the last column of the table in section 2.5 of the TAP standard (Dowler and Rixon et al., 2019).

Example:

```
<languageFeatures type="ivo://ivoa.net/std/TAPRegExt#features-udf">
  <feature>
    <form>match(pattern TEXT, string TEXT) -> INTEGER</form>
    <description>
      match returns 1 if the POSIX regular expression pattern
      matches anything in string, 0 otherwise.
    </description>
  </feature>
</languageFeatures>
```

ivo://ivoa.net/std/TAPRegExt#features-adqlgeo

Each feature declares support for one of the geometry functions defined by ADQL (support for these functions is in general optional for ADQL implementations, though TAP imposes some constraints on what combinations of support are permitted).

The signature of these functions, where supported, is fixed by ADQL; the content of the *form* element is just the name of the function.

Example:

```
<feature>
  <form>CONTAINS</form>
</feature>
```

tr:LanguageFeatureList Type Schema Documentation

An enumeration of non-standard or non-mandatory features of a specific type implemented by the language.

A feature type is a language-dependent concept like "user defined function", "geometry support", or possibly "units supported". A featureList gives all features of a given type applicable for the service. Multiple featureLists are possible.

All feature in a given list are of the same type. This type is declared using the mandatory type attribute, the value of which will typically be an IVOID. To see values defined in TAPRegExt, retrieve the ivo://ivoa.net/std/TAPRegExt resource record and look for keys starting with "features-".

tr:LanguageFeatureList Type Schema Definition

```
<xs:complexType name="LanguageFeatureList" >
  <xs:sequence >
    <xs:element name="feature" type="tr:LanguageFeature" minOccurs="0"
             maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="type" type="xs:anyURI" use="required" />
</xs:complexType>
```

tr:LanguageFeatureList Attributes

type

Type	a URI: ***xs:anyURI***
Meaning	The type of the features given here.
Occurrence	required
Comment	This is in general an IVOID. TAPRegExt itself gives IVOIDs for defining user defined functions and geometry support.

tr:LanguageFeatureList Metadata Elements

Element ***feature***

Type	composite: ***tr:LanguageFeature***
Meaning	A language feature of the type given by the type attribute.
Occurrence	optional; multiple occurrences allowed.

tr:LanguageFeature Type Schema Documentation

A non-standard or non-mandatory feature implemented by the language..

tr:LanguageFeature Type Schema Definition

```
<xs:complexType name="LanguageFeature" >
  <xs:sequence >
    <xs:element name="form" type="xs:token" />
    <xs:element name="description" type="xs:string" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
```

tr:LanguageFeature Metadata Elements

Element *form*

Type	string: ***xs:token***
Meaning	Formal notation for the language feature.
Occurrence	required
Comment	The syntax for the content of this element is defined by the type attribute of its parent language list.

Element *description*

Type	string: ***xs:string***
Meaning	Human-readable freeform documentation for the language feature.
Occurrence	optional

2.5 Output Formats

A TAP service may offer a variety of output formats. What output formats are available is defined using ***outputFormat*** elements. They declare an RFC 2046 media type (Freed and Borenstein, 1996) as well as aliases (the shorthand forms the server also accepts in the FORMAT parameter). If desired, the format can be further described with an IVOID in the ivo-id attribute; TAPRegExt provides keys for some variants of VOTables which are not interoperably distinguishable by their MIME types so far:

`output-votable-td` A VOTable in which all DATA elements contain a TABLEDATA element

`output-votable-binary` A VOTable in which all DATA elements contain a STREAM element with a BINARY child

`output-votable-binary2` A VOTable in which all DATA elements contain a STREAM element with a BINARY2 child

Operators can restrict individual output formats to either sync or async requests using the ***forMode*** attribute as explained in 2.2.

tr:OutputFormat **Type Schema Documentation**

An output format supported by the service.

All TAP services must support VOTable output, with media types as requested by the FORMAT parameter if applicable (cf. section 2.7.1 of the TAP standard).

The primary identifier for an output format is the RFC 2046 media type. If you want to register an output format, you must use a media type (or make one up using the x- syntax), although the concrete media syntax is not enforced by the schema.

For more detailed specification, an IVOID may be used.

tr:OutputFormat **Type Schema Definition**

```
<xs:complexType name="OutputFormat" >
  <xs:sequence >
    <xs:element name="mime" type="xs:token" />
    <xs:element name="alias" type="xs:token" minOccurs="0"
             maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attributeGroup ref="tr:mode-dependent" />
  <xs:attribute name="ivo-id" type="xs:anyURI" />
</xs:complexType>
```

tr:OutputFormat Attributes

ivo-id

Type a URI: ***xs:anyURI***

Meaning An optional IVOID of the output format.

Occurrence optional

Comment When the media type does not uniquely define the format (or a generic media type like application/octet-stream or text/plain is given), the IVOID can point to a key or StandardsRegExt document defining the format more precisely. To see values defined in TAPRegExt, retrieve the ivo://ivoa.net/std/TAPRegExt resource record and look for keys starting with "output-".

forMode

Type string

Meaning This declaration only applies to the named mode.

Occurrence optional

Allowed Values

sync
: This declaration is for synchronous queries.

async
: This declaration is for asynchronous queries.

tr:OutputFormat Metadata Elements

Element ***mime***

Type string: ***xs:token***

Meaning The media type of this format.

Occurrence required

Comment The format of this string is specified by RFC 2046. The service has to accept this string as a value of the FORMAT parameter.

Element ***alias***

Type string: ***xs:token***

Meaning Other values of FORMAT ("shorthands") that make the service return documents with the media type.

Occurrence optional; multiple occurrences allowed.

2.6 Upload Methods

TAP services should allow the upload of VOTables. They can support various methods to do this: HTTP upload, retrieval by URL, but also VOSpace or possibly retrieval using Grid protocols. Since an actual specification of the details of such protocols is far beyond the scope of a registry document and probably would not benefit clients anyway, the upload methods are given as IVOIDs.

IVOIDs for the standard upload methods are provided within the resource record `ivo://ivoa.net/std/TAPRegExt`. The IVOIDs are built by using the keys as fragments after the TAPRegExt IVOID.

It is permitted to register upload methods under authorities other than ivoa.net. The registry records can then provide more in-depth information. For the upload methods defined in the TAP specification, however, the IVOIDs of the keys in the TAPRegExt resource record must be used to enable clients to identify supported methods using string comparisons.

This document defines the following protocol identifiers:

- `upload-inline` – HTTP upload as per section 2.5.2 of the TAP standard (Dowler and Rixon et al., 2019).
- `upload-http` – retrieval from an http URL.
- `upload-https` – retrieval from an https URL.
- `upload-ftp` – retrieval from an ftp URL.

Thus, a service offering upload by retrieving from ftp and http URLs would say:

```
<uploadMethod ivo-id="ivo://ivoa.net/std/TAPRegExt#upload-http"/>
<uploadMethod ivo-id="ivo://ivoa.net/std/TAPRegExt#upload-ftp"/>
```

Operators can restrict individual upload methods to either sync or async requests using the *forMode* attribute as explained in 2.2.

tr:UploadMethod Type Schema Documentation

An upload method as defined by IVOA.

Upload methods are always identified by an IVOID. Descriptions can be obtained by dereferencing this IVOID. To see values defined in TAPRegExt, retrieve the ivo://ivoa.net/std/TAPRegExt resource record and look for keys starting with "upload-".

You can register custom upload methods, but you must use the standard IVOIDs for the upload methods defined in the TAP specification.

tr:UploadMethod Type Schema Definition

```
<xs:complexType name="UploadMethod" >
  <xs:complexContent >
    <xs:restriction base="xs:anyType" >
      <xs:attribute name="ivo-id" type="xs:anyURI" />
      <xs:attributeGroup ref="tr:mode-dependent" />
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

***tr:UploadMethod* Attributes**

ivo-id

Type a URI: ***xs:anyURI***

Meaning The IVOID of the upload method.

Occurrence optional

forMode

Type string

Meaning This declaration only applies to the named mode.

Occurrence optional

Allowed Values

sync
: This declaration is for synchronous queries.

async
: This declaration is for asynchronous queries.

2.7 Resource Limits

TAP services usually impose certain limits on resource usage by clients, e.g., a maximum run time per query, or a maximum number of rows in the result set. Services assign such limits to newly created jobs and may allow raising the limits by means of queries or query parameters (e.g., the size of the result set is limited by the `MAXREC` parameter, whereas the date of job destruction may be changed by posting to the `destruction` parameter). Services may put some limit to how far the resource limitations may be raised.

TAPRegExt's ***capabilities*** element allows the declaration of such limits. These declarations are primarily intended for human consumption and should give conservative guidelines. Thus, the operators of a service implementing a complex, possibly dynamic limits policy should give lower estimates here.

It is rather common that operators enforce different limits depending on the access mode (sync or async). Therefore, most limits have a ***forMode*** attribute as per sect. 2.2; for convenience and later extensibility, that is even true for ***retentionPeriod***, which admittedly cannot apply to sync requests.

If a service supports authentication and has different limits depending on what user is authenticated, it should make an effort to guess the limits applying to a given client (e.g., when authentication tokens are present in the request). Limits reported to the Registry should reflect limits for unauthenticated use unless the service does not admit unauthenticated requests.

The resource limits applying to newly created jobs are given in *default* elements, the limits beyond which users cannot raise the limits are given in *hard* elements.

Note that the absence of a specification of limits does not imply that no limits are enforced.

2.7.1 Limits on Time

This document defines two time-like resource limits:

- *retentionPeriod* – the time from job creation until `destruction`.
- *executionDuration* – the maximal run time given to a query.

All values in time-like limits are given in seconds. Both *retentionPeriod* and *executionDuration* are of type *tr:TimeLimits*.

tr:TimeLimits Type Schema Documentation

Time-valued limits, all values given in seconds.

tr:TimeLimits Type Schema Definition

```
<xs:complexType name="TimeLimits" >
  <xs:sequence >
    <xs:element name="default" type="xs:integer" minOccurs="0"
             maxOccurs="1" />
    <xs:element name="hard" type="xs:integer" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
  <xs:attributeGroup ref="tr:mode-dependent" />
</xs:complexType>
```

tr:TimeLimits Attributes

forMode

Type string

Meaning This declaration only applies to the named mode.

Occurrence optional

Allowed Values

sync
: This declaration is for synchronous queries.

async
: This declaration is for asynchronous queries.

tr:TimeLimits Metadata Elements

Element *default*

Type integer

Meaning The value of this limit for newly-created jobs, given in seconds.

Occurrence optional

Element *hard*

Type integer

Meaning The value this limit cannot be raised above, given in seconds.

Occurrence optional

2.7.2 Limits on Data

Limits on data are expressed much like time limits in that they give *default* and a *hard* value as well. Both those values have a unit attribute that can either be `byte` or `row` for data limits.

This document defines two resource limits on data:

- *outputLimit* – if *unit* is `row` here, the *default* gives the value of TAP's `MAXREC` parameter the service will use when none is specified.
- *uploadLimit* – the maximum size of uploads. This is not a TAP adjustable parameter. The *default* value advises clients about the server's wishes as to a limit above which some sort of acknowledgement should be requested from the user. The *hard* limit gives the maximum size of an upload to the server.

Data limits are defined using the *tr:DataLimits* and *tr:DataLimit* types:

tr:DataLimits Type Schema Documentation

Limits on data sizes, given in rows or bytes.

tr:DataLimits Type Schema Definition

```
<xs:complexType name="DataLimits" >
  <xs:sequence >
    <xs:element name="default" type="tr:DataLimit" minOccurs="0"
             maxOccurs="1" />
    <xs:element name="hard" type="tr:DataLimit" minOccurs="0"
             maxOccurs="1" />
  </xs:sequence>
  <xs:attributeGroup ref="tr:mode-dependent" />
</xs:complexType>
```

tr:DataLimits Attributes

forMode

Type string

Meaning This declaration only applies to the named mode.

Occurrence optional

Allowed Values

sync
: This declaration is for synchronous queries.

async
: This declaration is for asynchronous queries.

tr:DataLimits Metadata Elements

Element ***default***

Type	an integer with optional attributes
Meaning	The value of this limit for newly-created jobs.
Occurrence	optional

Element ***hard***

Type	an integer with optional attributes
Meaning	The value this limit cannot be raised above.
Occurrence	optional

tr:DataLimit Type Schema Documentation

A limit on some data size, either in rows or in bytes.

tr:DataLimit Type Schema Definition

```
<xs:complexType name="DataLimit" >
  <xs:simpleContent >
    <xs:extension base="xs:integer" >
      <xs:attribute name="unit" use="required" >
        <xs:simpleType >
          <xs:restriction base="xs:token" >
            <xs:enumeration value="byte" />
            <xs:enumeration value="row" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

tr:DataLimit Attributes

unit

Type	string with controlled vocabulary
Meaning	The unit of the limit specified.
Occurrence	required
Allowed Values	

byte

row

2.8 Interface Declaration

Version 1.0 of this specification used a VODataService ***vs:ParamHTTP*** interface to declare the TAP service's base URL. This was mainly done to maintain common service discovery patterns – other DAL protocols use ***vs:ParamHTTP***-typed interfaces as well –, but it ignored the semantics of the type, which in particular is that the access URL of such an interface is what clients use in

queries. Against that, the TAP specification actually leaves open what directly dereferencing the base URL should result in. Instead, all functionality is offered through children of the base URL. The abuse of *vs:ParamHTTP* later had a severe fallout as discussed in Demleitner (2019).

However, existing TAP clients rely on the interface declaration for discovery, and hence all capability elements with a TAPRegExt version 1 *standardID* must include a *vs:ParamHTTP*-typed interface with

- a child element *accessURL* with its content the base URL of the TAP service (the parent of sync, async, etc.) and its *use* attribute set to `base`.
- its *role* attribute set to `std`
- its *version* attribute set to the version of the TAP specification implemented.

In order to open the path to advanced use cases (e.g., involving authenticated access), services implementing TAPRegExt 1.1 must include another interface element of type *tr:DALIInterface*.

tr:DALIInterface Type Schema Documentation

An interface for a complex, multi-endpoint interfaces as regulated by DALI.

In addition to the standard vr:Interface elements, DALIInterfaces have endpoints, listed by name; that name doubles as a path segment to append to the interface's access URL, yielding the URI at which the endpoint is operated.

tr:DALIInterface Type Schema Definition

```
<xs:complexType name="DALIInterface" >
  <xs:complexContent >
    <xs:extension base="vr:Interface" >
      <xs:sequence >
        <xs:element name="endpoint" type="tr:Endpoint" minOccurs="0"
                  maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

tr:DALIInterface Extension Metadata Elements

Element *endpoint*

Type	composite: *tr:Endpoint*
Meaning	An endpoint accessible through this interface.
Occurrence	optional; multiple occurrences allowed.

The *tr:DALIInterface*-typed interface must satisfy the same requirements as the *vs:ParamHTTP*-typed one as regards its *accessURL* child and *role* and *version* attributes. Furthermore, this interface now enumerates the various endpoints exposed below the base URI in *endpoint* elements.

We may want to support vocab on endpoint; or do we want an implicit prefix as in datalink?

tr:Endpoint Type Schema Documentation

An endpoint of a complex interface.

An endpoint is characterised and addressed by its name; they can further be defined through RDF triples. This is a generic extension mechanism for endpoint-specific metadata, primarily intended for custom, vendor-specific extensions.

tr:Endpoint Type Schema Definition

```
<xs:complexType name="Endpoint" >
  <xs:sequence >
    <xs:element name="name" type="xs:token" minOccurs="1" maxOccurs="1" />
    <xs:element name="meta" type="tr:MetaTriple" minOccurs="0"
            maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
```

tr:Endpoint Metadata Elements

Element ***name***

Type string: ***xs:token***

Meaning The endpoint name, which is also the last component of the path of its URI.

Occurrence required

Comment Names without dashes are reserved for IVOA use and are expected to work the same way on all services. Well-known examples for such endpoint names include sync, async, and tables.

Element ***meta***

Type a string with optional attributes

Meaning Auxiliary information on this endpoint.

Occurrence optional; multiple occurrences allowed.

For TAP, at least the `sync`, `async`, and `tables` endpoints must be declared, as well as `examples` if present. The VOSI capabilities URL, while in TAP also a child of the base URL, continues to be declared in separate capability in this version rather than a DALI interface endpoint. TAPRegExt specifically discourages returning different capability contents from different interfaces (e.g., with and without authentication) of a service.

While in general, not overly much configurability should be enabled on the level of endpoints, operators can attach extra per-endpoint metadata using their ***meta*** children. These support a subset of the attributes defined by RDFa lite (Sporny, 2012), enough to make statements consisting of a subject (***about***), a predicate (***property***), and an object (content or ***resource***):

tr:MetaTriple Type Schema Documentation

A container for an RDFa triple giving information related to an endpoint.

tr:MetaTriple Type Schema Definition

```
<xs:complexType name="MetaTriple" >
  <xs:simpleContent >
    <xs:extension base="xs:token" >
      <xs:attribute name="about" type="xs:anyURI" use="optional" />
      <xs:attribute name="property" type="xs:anyURI" use="required" />
      <xs:attribute name="resource" type="xs:anyURI" use="optional" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

tr:MetaTriple Attributes

about

Type	a URI: ***xs:anyURI***
Meaning	The subject of the statement.
Occurrence	optional
Comment	If missing, the endpoint itself is assumed as the subject.

property

Type	a URI: ***xs:anyURI***
Meaning	The property of the statement.
Occurrence	required
Comment	This is a reference to an RDF resource. IVOA standards may define semantics for scheme-less URI; non-IVOA properties must use full URIs with at least scheme and authority; in this version, no vocab attributes are supported.

resource

Type	a URI: ***xs:anyURI***
Meaning	The object of the statement.
Occurrence	optional
Comment	If missing, the text content of the element is used as the object.

This is intended to give operators the possibility to communicate extra information to custom clients primarily by defining properties, which are simply URIs controlled by the operator and should dereference to documents explaining the semantics of the properties, ideally authored in HTML. The object of the statement is typically given by the element contents, but where the object cannot be inlined, the ***meta***'s ***resource*** attribute can be used instead.

For instance, to convey a custom method to upload tables on an (authenticated) tables endpoint, an operator could say:

```
<endpoint>
  <name>tables</name>
  <meta property="http://operator.example.edu/supports-our-extension"
    >table-upload</meta>
</endpoint>
```

To prevent a fragmentation of the TAP standard, such extensions should be restricted to single service providers and custom clients and be considered for addition to the TAP standard as other parties take them up.

Because it models the most essential aspects of RDF, the *meta* element, in principle, allows operators to build almost arbitrarily complex metadata structures when other *meta* elements are used as subjects. Continuing the example above, the operator could declare media types and labels for upload formats like this:

```
<endpoint>
  <name>tables</name>
  <meta property="http://operator.example.edu/supports-our-extension"
    id="custom-upload"
    >table-upload</meta>

  <meta about="#custom-upload"
    property="http://operator.example.edu/supports-format"
    id="fits-binary"/>
  <meta about="#fits-binary"
    property="http://operator.example.edu/upload-label"
    >FITS Binary</meta>
  <meta about="#fits-binary"
    property="http://operator.example.edu/upload-media-type"
    >application/ fits </meta>

  <meta about="#custom-upload"
    property="http://operator.example.edu/supports-format"
    id="votable"/>
  <meta about="#votable"
    property="http://operator.example.edu/upload-label"
    >VOTable</meta>
  <meta about="#votable"
    property="http://operator.example.edu/upload-media-type"
    >application/x-votable+xml</meta>
</endpoint>
```

Again, this facility is intended to support prototyping features as well as facilitate provider-specific extensions aiding custom tooling on top of standard interfaces. Interoperable TAP functionality should probably not need to rely on per-endpoint metadata of this complexity.

2.9 The Capability Record

Using the types defined above, the *tr:TableAccess* type can be defined. Note that it is a type, not a (global) element. In instance documents, you will typically place it in a capability element with an explicit type specification, like this:

```
<capability
  xmlns:tr="http://www.ivoa.net/xml/TAP/v1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  standardID="ivo://ivoa.net/std/TAP"
  xsi:type="tr:TableAccess">
  ...
```

tr:TableAccess-typed capabilites can be used with arbitrary standardIDs; this is different from previous TAPRegExt versions, which fixed it to ivo://ivoa.net/std/TAP. This URI should still be given for TAP 1.0 services.

Services having other capabilities (e.g., TAP 1.1 or even UWS) will use other standardIDs.

tr:TableAccess Type Schema Documentation

The capabilities of a TAP server.

The capabilities attempt to define most issues that the TAP standard leaves to the implementors ("may", "should").

tr:TableAccess Type Schema Definition

```
<xs:complexType name="TableAccess" >
  <xs:complexContent >
    <xs:extension base="vr:Capability" >
      <xs:sequence >
        <xs:element name="dataModel" type="tr:DataModelType" minOccurs="0"
                maxOccurs="unbounded" />
        <xs:element name="language" type="tr:Language" minOccurs="1"
                maxOccurs="unbounded" />
        <xs:element name="outputFormat" type="tr:OutputFormat" minOccurs="1"
                maxOccurs="unbounded" />
        <xs:element name="uploadMethod" type="tr:UploadMethod" minOccurs="0"
                maxOccurs="unbounded" />
        <xs:element name="retentionPeriod" type="tr:TimeLimits" minOccurs="0"
                maxOccurs="unbounded" />
        <xs:element name="executionDuration" type="tr:TimeLimits"
                minOccurs="0"
                maxOccurs="unbounded" />
        <xs:element name="outputLimit" type="tr:DataLimits" minOccurs="0"
                maxOccurs="unbounded" />
        <xs:element name="uploadLimit" type="tr:DataLimits" minOccurs="0"
                maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

tr:TableAccess Extension Metadata Elements

Element ***dataModel***

Type a string with optional attributes

Meaning Identifier of IVOA-approved data model supported by the service.

Occurrence optional; multiple occurrences allowed.

Element ***language***

Type composite: ***tr:Language***

Meaning Language supported by the service.

Occurrence required; multiple occurrences allowed.

Element ***outputFormat***

Type composite: ***tr:OutputFormat***

Meaning Output format supported by the service.

Occurrence required; multiple occurrences allowed.

Element *uploadMethod*

Type composite: *tr:UploadMethod*

Meaning Upload method supported by the service.

Occurrence optional; multiple occurrences allowed.

Comment The absence of upload methods indicates that the service does not support uploads at all.

Element *retentionPeriod*

Type composite: *tr:TimeLimits*

Meaning Limits on the time between job creation and destruction time.

Occurrence optional; multiple occurrences allowed.

Element *executionDuration*

Type composite: *tr:TimeLimits*

Meaning Limits on executionDuration.

Occurrence optional; multiple occurrences allowed.

Element *outputLimit*

Type composite: *tr:DataLimits*

Meaning Limits on the size of data returned.

Occurrence optional; multiple occurrences allowed.

Element *uploadLimit*

Type composite: *tr:DataLimits*

Meaning Limits on the size of uploaded data.

Occurrence optional; multiple occurrences allowed.

A Obtaining the Schema

The recommended way to obtain the TAPRegExt schema is to retrieve the namespace URI, http://www.ivoa.net/xml/TAPRegExt/v1.0, which will usually involve one or two redirects. Despite to suggestive name, this will always return the latest version of the TAPRegExt schema in major version 1; see Harrison and Demleitner et al. (2018) for the background of this somewhat unfortunate circumstance.

The schema accompanying this specific version is published alongside this document[4]. The document at this URI is only intended for use in review or debugging.

B Changes from Previous Versions

B.1 Changes from REC-1.0

- Adding tr:DALIInterface to the schema.

[4] https://www.ivoa.net/documents/TAPRegExt/20151117/TAPRegExt-v1.0.xsd

- Allowing a forMode attribute on outputFormat, uploadMethod, retentionPeriod, executionDuration, outputLimit, and uploadLimit; to make that work, allowing more than one of the limit elements now.
- Dropping the appendix with an example document. Using a standalone, validated file and auxiliaryurl instead.

B.2 Changes from WD-20110127

- userDefinedFunction was generalized to feature within languageFeatures.
- The uploadmethods StandardKeyEnumeration was replaced by a resource record for TAPRegExt as a whole. This now includes keys of output formats and features as well; therefore, upload method names in their new IVOIDs are prefixed with upload-
- Schema version was bumped to 1.0 (yes, we indulge in unversioned schema changes before this becomes REC).
- uploadLimit interpretation was changed: The default limit is now "advisory" and to be interpreted as such by clients, the hard limit is what is actually required by the server.
- There's now an optional ivo-id attribute on the version element within language.
- There's now an optional ivo-id attribute on output formats.

B.3 Changes from WD-20110727

- The namespace in the schema is now http://www.ivoa.net/xml/TAPRegExt/v1.0 consistent with what has already been stated in the text.
- The IVOID for ADQL is now ivo://ivoa.net/std/ADQL#v2.0; it is defined here to be in ADQL's record since we do not want to wait for the ADQL standard to be fixed, but ADQL versioning should really not be done here, so a TAPRegExt IVOID is out of the question.
- The IVOID of the TAPRegExt standard is now `ivo://ivoa.net/std/TAPRegExt` to conform with other standard IVOIDs. Unfortunately, this changes all other IVOIDs dependent on this.
- We now allow AnyURI on the ivo-id of language to allow fragment identifiers as, e.g., in ADQL.

B.4 Changes from PR-20120812

- Fixed units in limits to `"row"` and `"byte"`.

B.5 Changes from REC-1.0

- No longer restricting TableAccess' standardID to the (old) TAP standardID. TAPRegExt capabilities are now allowed with arbitrary standardIDs
- Changed former use of the term "IVORN" to "IVOID" (or Registry part, as appropriate) to comply with a proposed clarification of terminology in Identifiers 2.0
- The example now shows an entire capabilities response
- Repaired obscore data model URI in the example
- dataModel/@ivo-id is now typed xs:anyURI to allow fragment identifiers on it
- Migrated to ivoatex

References

Demleitner, M. (2019), 'On the use of capabilities in the VO', IVOA Note. http://ivoa.net/documents/caproles/.

Demleitner, M., Harrison, P., Molinaro, M., Greene, G., Dower, T. and Perdikeas, M. (2019), 'IVOA Registry Relational Schema Version 1.1', IVOA Recommendation 11 October 2019. doi:10.5479/ADS/bib/2019ivoa.spec.1011D, https://ui.adsabs.harvard.edu/abs/2019ivoa.spec.1011D.

Demleitner, M., Plante, R., Linde, T., Williams, R. and Noddle, K. (2016), 'IVOA Identifiers Version 2.0', IVOA Recommendation 23 May 2016, arXiv:1605.07501. doi:10.5479/ADS/bib/2016ivoa.spec.0523D, https://ui.adsabs.harvard.edu/abs/2016ivoa.spec.0523D.

Dower, T., Demleitner, M., Benson, K., Plante, R., Auden, E., Graham, M., Greene, G., Hill, M., Linde, T., Morris, D., O'Mullane, W., Rixon, G., Stébé, A. and Andrews, K. (2018), 'Registry Interfaces Version 1.1', IVOA Recommendation 23 July 2018. doi:10.5479/ADS/bib/2018ivoa.spec.0723D, https://ui.adsabs.harvard.edu/abs/2018ivoa.spec.0723D.

Dowler, P., Rixon, G., Tody, D. and Demleitner, M. (2019), 'Table Access Protocol Version 1.1', IVOA Recommendation 27 September 2019. doi:10.5479/ADS/bib/2019ivoa.spec.0927D, https://ui.adsabs.harvard.edu/abs/2019ivoa.spec.0927D.

Freed, N. and Borenstein, N. (1996), 'Mulitpurpose internet mail extensions', IETF RFC. http://www.ietf.org/rfc/rfc2046.txt.

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. doi:10.5479/ADS/bib/2017ivoa.spec.0524G, https://ui.adsabs.harvard.edu/abs/2017ivoa.spec.0524G.

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

Harrison, P., Burke, D., Plante, R., Rixon, G., Morris, D. and IVOA Registry Working Group (2012), 'StandardsRegExt: a VOResource Schema Extension for Describing IVOA Standards Version 1.0', IVOA Recommendation 08 May 2012, arXiv:1402.4745. doi:10.5479/ADS/bib/2012ivoa.spec.0508H, https://ui.adsabs.harvard.edu/abs/2012ivoa.spec.0508H.

Harrison, P., Demleitner, M., Major, B. and Dowler, P. (2018), 'XML Schema Versioning Policies Version 1.0', IVOA Endorsed Note 29 May 2018. doi:10.5479/ADS/bib/2018ivoa.spec.0529H, https://ui.adsabs.harvard.edu/abs/2018ivoa.spec.0529H.

Louys, M., Tody, D., Dowler, P., Durand, D., Michel, L., Bonnarel, F., Micol, A. and IVOA DataModel Working Group (2017), 'Observation Data Model Core Components, its Implementation in the Table Access Protocol Version 1.1', IVOA Recommendation 09 May 2017. doi:10.5479/ADS/bib/2017ivoa.spec.0509L, https://ui.adsabs.harvard.edu/abs/2017ivoa.spec.0509L.

Plante, R., Demleitner, M., Benson, K., Graham, M., Greene, G., Harrison, P., Lemson, G., Linde, T. and Rixon, G. (2018), 'VOResource: an XML Encoding Schema for Resource Metadata Version 1.1', IVOA Recommendation 25 June 2018. doi:10.5479/ADS/bib/2018ivoa.spec.0625P, https://ui.adsabs.harvard.edu/abs/2018ivoa.spec.0625P.

Sporny, M. (2012), 'RDFA lite 1.1', W3C Recommendation.