DRAFT – please do not distribute

International
Virtual
Observatory
Alliance

Data Access Layer Interface

Version 1.2

IVOA Working Draft 2023-07-12

Working Group
Data Access Layer Working Group

This version
https://www.ivoa.net/documents/DALI/20230712

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

Previous versions
DALI-1.1
DALI-1.0

Author(s)
Patrick Dowler, Markus Demleitner, Mark Taylor, Doug Tody

Editor(s)
Patrick Dowler

Abstract

This document describes the Data Access Layer Interface (DALI). DALI defines the base web service interface common to all Data Access Layer (DAL) services. This standard defines the behaviour of common resources, the meaning and use of common parameters, success and error responses, and DAL service registration. The goal of this specification is to define the common elements that are shared across DAL services in order to foster consistency across concrete DAL service specifications and to enable standard re-usable client and service implementations and libraries to be written and widely adopted.

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 https://www.ivoa.net/documents/.

Contents

PDF fallback: A conversion from SVG to PDF failed. This is probably because inkscape is not installed. While SVG is not supported by the major TeX engines, it is recommended to commit built PDFs to the VCS.

Figure 1: Architecture diagram for this document

1 Introduction

The Data Access Layer Interface (DALI) defines resources, parameters, and responses common to all DAL services so that concrete DAL service specifications need not repeat these common elements.

1.1 Role within the VO Architecture

DALI defines how DAL service specifications use other IVOA standards as well as standard internet designs and protocols. Fig. 1 shows the role this document plays within the IVOA architecture (Dowler and Evans et al., 2021).

Astronomical coordinate values accepted and returned by DAL services use a string representation of the Space-Time Coordinates (Rots, 2007) data model. The concrete DAL service specification defines whether the returned resources are serializations of a particular standard data model. For preserving backwards compatibility or to enable service-specific use cases, the concrete DAL service specification may explicitly specify the use of ad-hoc Utypes.

A registry extension schema, usually extending VODataService (Demleitner and Plante et al., 2021), may be used to describe the capabilities of a DAL service. This schema is used within the VOSI-capabilities (Graham and Rixon et al., 2017) resource and in registry records for the service.

1.2 Example Usage of the DALI Specification

The DALI specification defines common elements that make up Data Access Layer (DAL) services. DAL service specifications will refer to the sections in this document by name rather than include all the explanatory text. For example, suppose a document defines a service that stacks FITS images asynchronously, the specification could say that the service has the following resources:

- a DALI-async resource that accepts one or more UPLOAD parameters (section 4.3.5) where the resources are FITS images; the resource could also define a fixed set of error messages for anticipated failure modes
- a VOSI-availability resource (section 2.4)
- a VOSI-capabilities resource (section 2.5) conforming to a specified registry extension schema

and would have to define the registry extension schema to be used to register services and to implement the VOSI-capabilities resource. Most of the service specification would be in defining the semantics (possibly controllable via additional input parameters) of the computations to be performed and in defining the extension schema to describe service functionality and limits (e.g., maximum input or result image sizes, result retention time and policies). The registry extension schema may be part of the service specification or a separate document.

2 Resources

DAL services are normally implemented as HTTP REST (Fielding, 2000) web services, although other transport protocols could be used in the future. The primary resource in a DAL service is a job. A DAL job is defined by parameters (section 4) and can be executed either synchronously or asynchronously. A concrete service specification defines the job parameters and the manner of execution is defined by separate resources below.

In addition to job list resources, DAL services also implement several Virtual Observatory Support Interface (Graham and Rixon et al., 2017) resources to describe service availability, capabilities, and content.

A concrete DAL service must define at least one DALI-async or DALI-sync resource. It may define both with the same job semantics (e.g. TAP-1.0 (Dowler and Rixon et al., 2010)) or it may define one with one kind of job and the other with a separate kind of job (a service that does some things synchronously and others asynchronously).

The following table summarises the resources that are required in all concrete DAL service specifications (and thus in all DAL services) and which kinds of resources are defined and specified as required or optional in a concrete specification.

resource type	**resource name**	**required**
DALI-sync	service specific	service specific
DALI-async	service specific	service specific
DALI-examples	/examples	no
VOSI-availability	service specific	no
VOSI-capabilities	/capabilities	registered
VOSI-tables	service specific	service specific

The resource name is the path (relative to the base URL of the service). All implemented DALI and VOSI endpoints must be siblings, except for VOSI-availability (see below); concrete service specifications may constrain the names of these endpoints further. The relative path limitation enables a client with just the URL for a single endpoint to find the VOSI-capabilities endpoint and then discover all the capabilities provided by the service.

A VOSI-capabilities endpoint is required for services registered in the IVOA registry system; the VOSI-capabilities endpoint is optional for services that are not registered or only included as auxiliary capabilities (e.g. of a data collection resource).

The URL for the VOSI-availability is not constrained; it may be a sibling (e.g. /availability) or it may be hosted on a different server (e.g. VOSI-availability may be implemented as a completely external resource that tests the service from the user perspective).

A simple query-only DAL service like ConeSearch can be easily described as having a single DALI-sync resource where the job is a query and the response is the result of the query.

2.1 Asynchronous Execution: DALI-async

Asynchronous resources are resources that represent a list of asynchronous jobs as defined by the Universal Worker Service (UWS) pattern (Harrison and Rixon, 2016). Requests can create, modify, and delete jobs in the job list. UWS also specifies special requests to modify the phase of the job (cause the job to execute or abort).

As specified in UWS, a job is created by using the HTTP POST method to modify the job list. The response will always be an HTTP redirect (status code 303) and the Location (HTTP header) will contain the URL to the job.

```
POST http://example.com/base/async-jobs
```

The response will include the HTTP status code 303 (See Other) and a header named Location with a URL to the created job as a value, for example:

```
Location: http://example.com/base/async-jobs/123
```

The job description (an XML document defined by the UWS schema) can always be retrieved by accessing the job URL with the HTTP GET method:

```
GET http://example.com/base/async-jobs/123
```

```
<?xml version="1.0" encoding="UTF-8"?>
<uws:job xmlns:uws="http://www.ivoa.net/xml/UWS/v1.0">
  <uws:jobId>123</uws:jobId>
  <uws:runId>test</uws:runId>
  <uws:ownerId xsi:nil="true" />
  <uws:phase>PENDING</uws:phase>
  <uws:quote>2013-01-01T12:34:56</uws:quote>
  <uws:startTime/>
  <uws:endTime/>
  <uws:executionDuration>600</uws:executionDuration>
```

```
  <uws:destruction>2013-02-01T00:00:00</uws:destruction>
  <uws:parameters>
    <uws:parameter id="LANG">ADQL</uws:parameter>
    <uws:parameter id="REQUEST">doQuery</uws:parameter>
    <uws:parameter id="QUERY">select * from tab</uws:parameter>
  </uws:parameters>
  <uws:results/>
</uws:job>
```

In addition to the UWS job metadata, DAL jobs are defined by a set of parameter-value pairs. The client may include parameters in the initial POST that creates a job or it may add additional parameters by a POST to the current list of parameters, for example:

```
http://example.com/base/async-jobs/123/parameters
```

DALI-async resources may provide other ways to interact with jobs as specified in current or future UWS specifications, with the following exception: the UWS-1.0 standard may be interpreted to allow POSTing of job parameters to the job URL, but DALI-async resources must not accept job parameters at this URL.

Job parameters may only be POSTed while the job is in the PENDING phase; once execution has been requested and the job is in any other phase, job parameters may not be modified.

A concrete DAL service specification will specify zero or more asynchronous job submission resources and whether they are mandatory or optional. It may mandate a specific resource name to support simple client use, or it can allow the resource name to be described in the service metadata (Section 2.5).

2.2 Synchronous Execution: DALI-sync

Synchronous resources are resources that accept a request (a DAL job description) and return the response (the result) directly. Synchronous requests can be made using either the HTTP GET or POST method. If a specific type of job is exposed through both DALI-async and DALI-sync resources (e.g. TAP queries), then the parameters used to specify the job are the same for this pair of (synchronous and asynchronous) jobs. Service specifications may also specify different types of jobs on different resources, which would have different job parameters.

A synchronous job is created by a GET or POST request to a synchronous job list, executed automatically, and the result returned in the response. The

web service is permitted to split the operation of a synchronous request into multiple HTTP requests as long as it is transparent to standard clients. This means that the service may use HTTP redirects (status code 302 or 303) and the Location header to execute a synchronous job in multiple steps. For example, a service may

- immediately execute and return the result in the response, or
- the response is an HTTP redirect (status code 303) and the Location (HTTP header) will contain a URL; the client accesses this URL with the HTTP GET method to execute the job and get the result

Clients must be prepared to get redirects and follow them (using normal HTTP semantics) in order to complete requests.

A concrete DAL service specification will specify zero or more synchronous job submission resources and whether they are mandatory or optional. It may mandate a specific resource name to support simple client use, or it can allow the resource name to be described in the service capability metadata (Section 2.5).

2.3 DALI-examples

The DALI-examples resource returns a document with usage examples or similar material to the user. In DAL services, this resource is always accessed as a resource named examples that is a child of the base URL for the service. The following specification is intended to make sure the content is usable for both machines and humans. As such, the DALI-examples resource contains additional markup conforming to the RDFa 1.1 Lite (Sporny, 2012) specification, which defines the following attributes: *vocab*, *typeof*, *property*, *resource*, and *prefix* (although we do not include any use of the *prefix* attribute).

The DALI-examples capability identifier is:

ivo://ivoa.net/std/DALI#examples

DAL services may implement the /examples resource and include it in the capabilities described by the VOSI-capabilities resource (Section 2.5); if they do not, retrieving its URL must yield a 404 HTTP error code.

The document at /examples must be well-formed XML. This restriction is imposed in order to let clients parse the document using XML parsers

rather than much more complex parsers (e.g. HTML5 parsers). It is therefore advisable to author it in XHTML, although this specification does not prescribe any document types.

The document should be viewable with "common web browsers". Javascript or CSS must not be necessary to find and interpret the elements specified below. Apart from that, service operators are free to include whatever material or styling they desire in addition and within the example elements defined here.

The elements containing examples must be descendants of an element that has a *vocab* attribute with the value as shown below:

```
<div vocab="http://www.ivoa.net/rdf/examples#">
 ...
</div>
```

The URI in the *vocab* attribute resolves to an IVOA vocabulary of concepts useful for describing examples. That vocabulary complies to Vocabularies in the VO version 2 (Demleitner and Gray et al., 2021). The values of the *property* attributes below are described in it, and the concept URIs formed according to RDFa rules resolve to elements within it, which may be useful for documentation purposes. Clients purely interested in presenting the examples to their users usually have no reason to retrieve the vocabulary.

No other *vocab* attributes are allowed in the document. Each example resides in an element that has a *typeof* attribute with the value *example*. All such elements must have an *id* attribute to allow external referencing via fragments and a *resource* attribute with a reference pointing to the element itself. As an example,

```
<div id="x" resource="#x" typeof="example"> ... </div>
```

located inside the element having the *vocab* attribute would contain an example referable via the x fragment identifier. The *div* element is a suitable HTML element to hold an example.

The content of the example is expressed using the *property* attribute. For DALI-examples, we define the following values for the *property* attribute:

- *name*
- *capability*
- *generic-parameter*

- *continuation*.

Each example must include one name. DAL service specifications may define additional properties so they can mark up additional information in their examples using the procedures described in Vocabularies in the VO 2. For instance, TAP has introduced the notions of *query* and *table*.

In principle, any element permitted by the document type can include the RDFa attributes, so authors may re-use existing markup intended for display. Alternatively, the ***span*** element is a good choice when the example values are included in surrounding text and the author does not want any special rendering to be applied by the machine-readable additions.

To maintain compatibility with mainstream RDFa tools, extra care is necessary with elements that have ***src*** or ***href*** attributes. According to RDFa rules, in such cases the object of the relationship is the linked entity rather than the element content. While this is intended in some cases – see the continuation property below – this will lead to erroneous interpretations in the typical case.

For instance,

```
<!-- Wrong! -->
<div id="x" resource="#x" typeof="example">
<p>The case of <a property="name"
  href="http://object-resolver.edu/M42">Messier 42</a> is special.</p>
</div>
```

would imply that the name of the example `x` is http://object-resolver.edu/M42 rather than just "Messier 42". Full RDFa offers the ***content*** attribute to allow correct markup even in the presence of ***href*** attributes, but since DALI examples are restricted to RDFa lite, this cannot be used.

The rule of thumb is to only use elements with links when the relationship's object actually is a linked document or entity (for the terms given here, this is only true for continuation). If document authors wants to express a link with the relationship's object anyway, they will have to restructure their texts (which typically will also yield better link semantics). For instance, the example above could be written as:

```
<div id="x" resource="#x" typeof="example">
<p>The case of <span property="name">Messier 42</span> (<a
  href="http://object-resolver.edu/M42">M42 at object resolver</a>)
  is special.</p>
</div>
```

2.3.1 name property

The content of this element must be plain text (i.e., no child elements) and should be suitable for display within a space-limited label in user interface and still give some idea about the meaning of the example. In XHTML, a head element (*h2*, say) would usually be a good choice for the example name, for example:

```
<h2 property="name">Synchronous TAP query</h2>
```

2.3.2 capability property

The capability property for an example specifies which service capability the example is to be used with by giving, in plain text, the standards URI as given in the respective capability's *standardID* attribute. For example, if the text is describing how to use a SODA-1.0 service, the example could contain:

```
<span property="capability">ivo://ivoa.net/std/SODA#sync-1.0</span>
```

IVOA standard service capabilities are defined as URIs, so example documents may want to show the URI or show more user-friendly text depending on the expected audience for the document. For specifications that do not define specific capability identifiers, the IVOID for the specification itself should be used.

2.3.3 generic-parameter property

Request parameters are included within the example by using the generic-parameter property. The element must also be assigned a *typeof* attribute with value of *keyval*. Within this element, the document must include a pair of elements with *property* attributes valued key and value, where the plain-text content are the parameter name and value respectively. Multiple generic-parameter(s) are permitted, for example:

```
<span property="generic-parameter" typeof="keyval">
   <span property="key">REQUEST</span>
   <span property="value">doQuery</span>
</span>
<span property="generic-parameter" typeof="keyval">
   <span property="key">LANG</span>
   <span property="value">ADQL</span>
</span>
<span property="generic-parameter" typeof="keyval">
```

```
   <span property="key">QUERY</span>
   <span property="value">SELECT * from tap_schema.tables</span>
</span>
```

2.3.4 continuation property

If the examples are spread over multiple linked documents, the links to documents with additional examples must be within the parent element defining the *vocab* attribute and the link elements must contain the following additional attributes: a *property* attribute with the value *continuation*, a *resource* attribute with an empty value (referring to the current document), and the *href* attribute with the URL of another document formatted as above (i.e. another collection of examples that clients should read to collect the full set of examples).

```
<div vocab="http://www.ivoa.net/rdf/examples#">
  <div id="x" resource="#x" typeof="example">
    ...
  </div>
  <a property="continuation"
     href="simple_examples.html">Simple examples</a>
  <a property="continuation"
     href="fancy_examples.html">Fancy examples</a>
</div>
```

In the above example, the two linked documents would also contain some element with a vocab and examples as described above.

2.4 Availability: VOSI-availability

VOSI-availability (Graham and Rixon et al., 2017) defines a simple web resource that reports on the current ability of the service to perform.

If the VOSI-availability resource is implemented a description of this capability must be provided in the VOSI-capabilities document. The VOSI-availability resource is intended to respond with a dynamically generated document describing the current state of the service operation, e.g.:

```
<?xml version="1.0" encoding="UTF-8"?>
<vosi:availability
  xmlns:vosi="http://www.ivoa.net/xml/VOSIAvailability/v1.0">
  <vosi:available>true</vosi:available>
  <vosi:note>service is accepting queries</vosi:note>
</vosi:availability>
```

2.5 Capabilities: VOSI-capabilities

VOSI-capabilities (Graham and Rixon et al., 2017) defines a simple web resource that returns an XML document describing the service. In DAL services, this resource is always accessed as a resource named capabilities that is a child of the base URL for the service. The VOSI-capabilities should describe all the resources exposed by the service, including which standards each resource implements.

All registered DAL services must implement the /capabilities resource. The following capabilities document shows the capabilities and tables VOSI resources and a TAP base resource:

```
<?xml version="1.0" encoding="UTF-8"?>
<vosi:capabilities
   xmlns:vosi="http://www.ivoa.net/xml/VOSICapabilities/v1.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:vod="http://www.ivoa.net/xml/VODataService/v1.1">

  <capability standardID="ivo://ivoa.net/std/VOSI#capabilities">
    <interface xsi:type="vod:ParamHTTP" version="1.0">
      <accessURL use="full">
         http://example.com/tap/capabilities
      </accessURL>
    </interface>
  </capability>

  <capability standardID="ivo://ivoa.net/std/VOSI#tables">
    <interface xsi:type="vod:ParamHTTP" version="1.0">
      <accessURL use="full">
        http://example.com/tap/tables
      </accessURL>
    </interface>
  </capability>

  <capability xmlns:tr="http://www.ivoa.net/xml/TAPRegExt/v1.0"
    standardID="ivo://ivoa.net/std/TAP" xsi:type="tr:TableAccess">
    <interface xsi:type="vod:ParamHTTP" role="std" version="1.0">
      <accessURL use="full">
           http://example.com/tap/
      </accessURL>
    </interface>
    <!-- service details from TAPRegExt go here -->
  </capability>
</vosi:capabilities>
```

Note that while this example shows the use of a registry extension schema (the inline `xmlns:tr="http://www.ivoa.net/xml/TAPRegExt/v1.0"` in the

last capability element) this is not required; services may be registered and described without such an extension. The use of *standardID* – which should contain the IVOID of the standard a capability adheres to – does not imply a particular (or any) *xsi:type* be included.

2.6 Tables: VOSI-tables

VOSI-tables (Graham and Rixon et al., 2017) defines a simple web resource that returns an XML document describing the content of the service. The document format is defined by the VOSI (Graham and Rixon et al., 2017) standard and allows the service to describe their content as a tableset: schemas, tables, and columns.

A concrete DAL service specification will specify if the VOSI-tables resource is permitted or required and may restrict the resource name or location. Since DAL services with a VOSI-tables resource will specify in the capabilities which version they are using, DAL services can make use of new versions without change to the DAL service specification.

3 Data Types and Literal Values

In this section we specify how values are to be expressed. These literal values are used as input or output from DAL services: as parameter values when invoking simple services, as data values in response documents (e.g. VOTable), etc. We define some general purpose values for the *xtype* attribute of the VOTable *FIELD* and *PARAM* elements for simple structured values: *timestamp*, *interval*, *hms*, *dms*, *point*, *circle*, *range*, *polygon*, *moc*, *multipolygon*, *shape*, *uri*, and *uuid* (see below).

Services may use non-standard *xtype* values for non-standard datatypes, but if they do so they should include a simple prefix (a string followed by a colon followed by the non-standard xtype) so client software can easily determine if a value is standard or not. For example, an *xtype* for a non-standard 3D-vector might be *geom:vector3d*.

3.1 Numbers

Integer and real numbers must be represented in a manner consistent with the specification for numbers in VOTable (Ochsenbein and Taylor et al., 2019).

3.2 Boolean

Boolean values must be represented in a manner consistent with the specification for Boolean in XML Schema Datatypes (Biron and Malhotra, 2004). The values 0 and false are equivalent. The values 1 and true are equivalent.

```
FOO=1
FOO=true

BAR=0
BAR=false
```

3.3 Timestamp

Date and time values must be represented using the convention established for FITS (Hanisch and Farris et al., 2001) and STC (Rots, 2007) for astronomical times:

```
YYYY-MM-DD['T'hh:mm:ss[.SSS]]
```

where the T is a character separating the date and time components. The time component is optional, in which case the T separator is not used. Fractions of a second are permitted but not required. For example:

```
2000-01-02T15:20:30.456
2001-02-03T04:05:06
2002-03-04
```

are all legal date or date plus time values. Astronomical values never include a time zone indicator. However, values that are civil in nature (e.g. when some processing was completed, when some record was last modified) may include the time zone indicator Z to explicitly specify the UTC time zone. Civil times conform to:

```
YYYY-MM-DD['T'hh:mm:ss[.SSS]['Z']]
```

where the optional Z character indicates the value is UTC. For example:

```
2000-01-02T15:20:30.456Z
2000-01-02T15:20:30Z
```

are valid civil time values. In cases where time values may be expressed using Julian Date (JD) or Modified Julian Date (MJD), these follow the rules for double precision numbers above and may have additional metadata as described in the VOTable standard (Ochsenbein and Taylor et al., 2019). All date-time values (formatted string, JD, and MJD) shall be interpreted as referring to time scale UTC and time reference position UNKNOWN, unless either or both of these are explicitly specified to be different (Rots, 2007).

Note that the format used here is very close to the standard ISO8601 timestamp format except with respect to timezone handling. ISO8601 requires a Z character at the end of the string when the timezone is UTC; here, we follow the FITS (Hanisch and Farris et al., 2001) convention for astronomical values by omitting the Z but still defaulting to UTC.

Timestamp values serialised on VOTable or in service parameters must have the following metadata in the *FIELD* element: `datatype="char"`, `arraysize="*"`, `xtype="timestamp"`; the arraysize may be set to a more specific value if it is known (e.g. `arraysize="10"` for dates only).

3.4 Intervals

Numeric intervals are pairs of numeric values (integer and floating-point). For floating point intervals, special values for positive and negative infinity may be used to specify open-ended intervals. Finite bounding values are included in the interval. Open-ended floating-point intervals have one or both bounding values that are infinite. Intervals with two identical values are equivalent to a scalar value but must still be serialised as a pair of values.

The representation of an interval uses the numeric array serialisation from VOTable. For example:

```
0.5 1.0
-Inf 0.0
0.0 +Inf
-Inf +Inf
1.0 1.0
```

are all legal floating-point interval values and:

```
0 2
-5 5
0 0
```

are all legal integer interval values.

Floating point interval values serialised in VOTable or service parameters must have the following metadata in the *FIELD* element: `datatype="double"` or `datatype="float"`, `arraysize="2"`, `xtype="interval"`.

Integer interval values serialised in VOTable or service parameters must have the following metadata in the *FIELD* element: `datatype="short"` or `datatype="int"` or `datatype="long"`, `arraysize="2"`, `xtype="interval"`.

Interval values serialised in VOTable (*FIELD*) or service parameters (*PARAM*) with this xtype may include additional metadata like minimum or maximum value. These are specified using the standard scalar `MIN` and `MAX` child elements to describe the (minimum) lower bound and (maximum) upper bound of interval(s) respectively.

3.5 Sexagesimal Coordinates

Coordinate values expressed in sexagesimal form can be described using the following xtypes in both VOTable *FIELD* and *PARAM* elements:

- right ascension: `datatype="char" arraysize="*" xtype="hms"`
- declination: `datatype="char" arraysize="*" xtype="dms"`

(the arraysize may also be a fixed length or variable length with limit).

For `xtype="hms"`, the values are serialised as hours:minutes:seconds where hours and minutes are integer values and seconds is a real value. For `xtype="dms"`, the values are serialised as degrees:minutes:seconds where degrees and minutes are integer values and seconds is a real value. All hours must fall within [0,24], degrees (latitude) must fall within [-90,90], minutes must fall within [0,60), and seconds must fall within [0,60). Valid values for `xtype="hms"` are from 0:0:0 to 24:0:0. Valid values for `xtype="dms"` are from -90:0:0 to 90:0:0; an optional + sign at the start is allowed (e.g. +10:20:30) but not required. The upper bound on minutes and seconds is not part of the valid range; for example 12:34:60 is not allowed and must be expressed as 12:35:00 instead.

3.6 Point

Geometry values are two-dimensional; although they are usually longitude and latitude values in spherical coordinates this is specified in the coordinate metadata and not in the values.

Point values serialised in VOTable or service parameters must have the following metadata in the *FIELD* element: `datatype="double"` or `datatype="float"`, `arraysize="2"`, `xtype="point"`. For points in a spherical coordinate system, the values are ordered as: longitude latitude. For example:

```
12.3 45.6
```

Coordinate values are not limited to fall within a defined valid range; this is a change from DALI 1.1 where equatorial coordinates were explicitly limited. Software may have to perform range reduction in some coordinate systems (for example, spherical coordinates) in order to correctly interpret or use the coordinate values. Coordinate values are more likely to work as expected if they are expressed in the simplest form and do not require range reduction. For example, in spherical coordinates, `362.0 2.0` is equivalent to `2.0 2.0`, but the latter form is more likely to work as intended in all cases.

There is no general purpose definition of minimum and/or maximum point values, but specific services may define something that is applicable in a more limited context.

3.7 Circle

Circle values serialised in VOTable or service parameters must have the following metadata in the *FIELD* element: `datatype="double"` or `datatype="float"`, `arraysize="3"`, `xtype="circle"`. The values are ordered as a point followed by a radius. For example:

```
12.3 45.6 0.5
```

Valid coordinate value limits are specified by `xtype="point"` above.

Circle-valued service parameters may include additional metadata like minimum and or maximum value. These are specified using a custom interpretation of the `MAX` child element with a value that is the largest circle that makes sense for the operation. The value could be a maximum allowed by the service or simply the circle where larger circles and circles outside the specified maximum will not yield useful results. Since the maximum circle includes coordinates and radius, it is useful for describing parameters of a request related to a specific target location (for example, a SODA cutout of specific archival data).

There is no general purpose definition of a minimum circle value for parameters or a definition of a minimum or maximum circle to describe field

values (in a column of a table), but specific services may define something that is applicable in a more limited context.

3.8 Range

Range values serialised in VOTable or service parameters must have the following metadata in the *FIELD* element: `datatype="double"` or `datatype="float"`, `arraysize="4"`, `xtype="range"`. A range is a coordinate bounding box specified as two pairs of coordinate values: min-coordinate1 max-coordinate1 min-coordinate2 max-coordinate2. For example:

```
10.0 11.0 20.0 21.0
```

includes values from 10 to 11 (coordinate1) and from 20 to 21 (coordinate2).

Valid coordinate value limits are specified by `xtype="point"` above. This range form is used as part of the value of the POS parameter in (Dowler and Bonnarel et al., 2015) and (Bonnarel and Dowler et al., 2017) (see also "shape" below). For example, a range can span the meridian (longitude 0): 359 1 -1 1 is interpreted as the small (2x2 degree) coordinate range from 359 across the meridian to 1 degree longitude.

Range-valued service parameters may include additional metadata like minimum and or maximum value. These are specified using a custom interpretation of the `MAX` child element with a value that is the largest range that makes sense for the operation. The value could be a maximum allowed by the service or simply the range where larger ranges and ranges outside the specified maximum will not yield useful results.

There is no general purpose definition of a minimum range value for parameters or a definition of a minimum or maximum range to describe field values (in a column of a table), but specific services may define something that is applicable in a more limited context.

3.9 Polygon

Polygon values serialised in VOTable or service parameters must have the following metadata in the *FIELD* element: `datatype="double"` or `datatype="float"`, `arraysize="*"`, `xtype="polygon"` (where arraysize may also be fixed length or variable length with limit). The array holds a sequence of vertices (points) (e.g. longitude latitude longitude latitude ...) with an even number of values and at least three (3) points (six (6) numeric values). A polygon is always implicitly closed: there is an implied edge from the last point back to the

first point; explicitly including the first point at the end is highly discouraged because it creates an edge of length 0 that has negative side effects on some polygon computations. For example:

```
10.0 10.0 10.2 10.0 10.2 10.2 10.0 10.2
```

Valid coordinate value limits are specified by `xtype="point"` above. Vertices must be ordered such that the polygon winding direction is counterclockwise (when viewed from the origin toward the sky) as described in (Rots, 2007).

Polygon-valued service parameters may include additional metadata to describe minimum and/or maximum values. These are specified using a custom interpretation of the `MAX` child element with a value that is the largest polygon that makes sense for the operation. The value could be a maximum allowed by the service or simply the polygon that describes the target region. Since the maximum polygon includes coordinates, it is useful for describing parameters of a request related to a specific target location (for example, a SODA cutout of specific archival data).

There is no general purpose definition of a minimum polygon value for parameters or a definition of a minimum or maximum polygon to describe field values (in a column of a table), but specific services may define something that is applicable in a more limited context.

3.10 MOC

Spatial MOC (Multi Order Coverage) values serialised in VOTable or service parameters must have the following metadata in the *`FIELD`* element: `datatype="char"`, `arraysize="*"`, `xtype="moc"` (where arraysize may also be fixed length or variable length with limit). The value is the ascii serialisation of a MOC specified in Fernique and Nebot et al. (2022) section 4.3.2 and may be a one- or two-dimension (spatial) MOC.

Note: explicit time MOC and space-time MOC xtypes may be added in a future version.

3.11 Multi-Polygon

Multi-polygon values serialised in VOTable or service parameters must have the following metadata in the *`FIELD`* element: `datatype="double"` or `datatype="float"`, `arraysize="*"`, `xtype="multipolygon"` (where arraysize may also be fixed

length or variable length with limit). The array holds a sequence of non-overlapping polygon(s) separated by a pair of `NaN` values (a NaN point). For example:

```
10.0 10.0 10.2 10.0 10.2 10.2 10.0 10.2 NaN NaN
11.0 11.0 11.2 11.0 11.2 11.2 11.0 11.2
```

A multi-polygon without a separator is allowed, so all (simple) polygons are also valid multi-polygons. The component polygons in a multipolygon may touch (vertex of one on an edge of another, including sharing vertices) but may not have any common area.

Multi-polygon-valued service parameters can have additional metadata as described for polygon above, except that the maximum value may be a multipolygon.

3.12 Shape

Shape values serialised in VOTable or service parameters must have the following metadata in the *FIELD* element: `datatype="char"`, `arraysize="*"`, `xtype="shape"` (where arraysize may also be fixed length or variable length with limit). The value is a polymorphic shape made up of a type label (equivalent to an existing simple geometric xtype and the string serialisation of the value as described above.

The allowed shapes are: `circle`, `range`, `polygon`. For example:

```
circle 12.3 45.6 0.5

range 10.0 11.0 20.0 21.0

polygon 10.0 10.0 10.2 10.0 10.2 10.2 10.0 10.2
```

The interpretation and constraints on the coordinate values are as specified for the individual xtypes above.

The shape xtype provides a compatible description of the POS parameter in (Dowler and Bonnarel et al., 2015) and (Bonnarel and Dowler et al., 2017).

Shape-valued service parameters may include additional metadata to describe minimum and/or maximum values. These are specified using a custom interpretation of the `MAX` child element with a value that is the largest shape that makes sense for the operation. The value could be a maximum allowed by the service or simply the shape that describes the target region. Since the

maximum shape includes coordinates and radius, it is useful for describing parameters of a request related to a specific target location (for example, a SODA cutout of specific archival data).

For example, the following would describe the maximum shape for an input `POS` parameter for a large (IRIS) data file (accessible via the CADC SODA service):

```
<GROUP name="inputParams">
  <PARAM name="ID" datatype="char" arraysize="23"
        ucd="meta.id;meta.dataset" value="cadc:IRIS/I212B2H0.fits" />
  ...
  <PARAM name="POS" datatytpe="char" arraysize="*" xtype="shape"
        ucd="obs.field" value="">
    <VALUES>
      <MAX value="polygon␣134.38␣-6.37␣134.42␣6.01
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣146.87␣5.97␣146.84␣-6.41" />
    </VALUES>
  </PARAM>
  ...
</GROUP>
```

In the specific context of a SODA service, the maximum shape is generally going to be the bounds of the data. The type label used in the maximum shape only tells the client how to interpret the value; it does not limit the caller to only using that type of shape.

There is no general purpose definition of a minimum shape value for parameters or a definition of a minimum or maximum shape to describe field values (in a column of a table), but specific services may define something that is applicable in a more limited context.

3.13 URI

URI values (Berners-Lee and Fielding et al., 2005) serialised in VOTable or service parameters should have the following metadata in the *`FIELD`* element: `datatype="char"`, `arraysize="*"`, `xtype="uri"` (where arraysize may also be fixed length or variable length with limit).

3.14 UUID

Universal Unique Identifier (UUID) values serialised in VOTable or service parameters should have the following metadata in the *`FIELD`* element: `datatype="char"`, `arraysize="36"`, `xtype="uuid"` (where arraysize may also be fixed length or variable length with limit).

UUID values (Leach and Mealling et al., 2005) are serialised using the canonical ascii (hex) representation, for example: e0b895ca-2ee4-4f0f-b595-cbd83be40b04.

3.15 Unsupported Types

Support for any specific *xtype* in implementations (client or service) is specified in the service standard document. However, support for a specific *xtype* as input (params and uploaded content) should generally be considered optional. Implementations should be able to read and write the underlying data type without knowing the semantics added by the *xtype*. In cases where understanding the meaning of an *xtype* is required (for example, the POS param in SODA) and a service does not support the serialized value, the service should issue an error message that starts with the following text with the most specific *xtype* noted:

```
unsupported-xtype: {xtype} [optional detail here]
```

and may include additional detail where noted. For example, the value of the SODA POS parameter is a `xtype="shape"`, but if the implementation does not support the "range" construct, it would respond (minimally) with:

```
unsupported-xtype: range
```

This behaviour will allow for new *xtype*s to be introduced and for `xtype="shape"` to be extended to include additional subtypes in the future.

4 Parameters

A DAL job is defined by a set of parameter-value pairs. Some of these parameters have a standard meaning and are defined here, but most are defined by the service specification or another related standard.

4.1 Case Sensitivity

Parameter names are not case sensitive; a DAL service must treat upper-, lower-, and mixed-case parameter names as equal. Parameter values are case sensitive unless a concrete DAL service specification explicitly states that the values of a specific parameter are to be treated as case-insensitive. For example, the following are equivalent:

```
FOO=bar
Foo=bar
foo=bar
```

Unless explicitly stated by the service specification, these are not equivalent:

```
FOO=bar
FOO=Bar
FOO=BAR
```

In this document, parameter names are typically shown in upper-case for typographical clarity, not as a requirement.

4.2 Multiple Values

Parameters may be assigned multiple values with multiple parameter=value pairs using the same parameter name. Whether or not multiple values are permitted and the meaning of multiple values is specified for each parameter by the specification that defines the parameter. For example, the UPLOAD parameter (section 4.3.5) permits multiple occurrences of the specified pair (table,uri), e.g.:

```
UPLOAD=foo,http://example.com/foo
UPLOAD=bar,http://example.com/bar
```

Services must respond with an error if the request includes multiple values for parameters defined to be single-valued.

4.3 Standard Parameters

4.3.1 REQUEST

The REQUEST parameter is intended for service capabilities that have multiple modes or operations, including non-standard (site-specific) optional features. Most standard service capabilities will not define values for this parameter.

If defined for a specific service capability, the REQUEST parameter is always single-valued.

4.3.2 VERSION

The VERSION parameter has been removed because the different meaning of request parameters it is intended to disambiguate are not allowed within minor revisions of a standard; there are no useful scenarios where VERSION would work.

4.3.3 RESPONSEFORMAT

The RESPONSEFORMAT parameter is used so the client can specify the format of the response (e.g. the output of the job). For DALI-sync requests, this is the content-type of the response. For DALI-async requests, this is the content-type of the result resource(s) the client can retrieve from the UWS result list resource; if a DALI-async job creates multiple results, the RESPONSEFORMAT should control the primary result type, but details can be specific to individual service specifications. While the list of supported values are specific to a concrete service specification, the general usage is to support values that are MIME media types (Freed and Borenstein, 1996) for known formats as well as shortcut symbolic values.

table type	media type	short form
VOTable	application/x-votable+xml	votable
VOTable	text/xml	votable
comma-separated values	text/csv	csv
tab separated values	text/tab-separated-values	tsv
FITS file	application/fits	fits
pretty-printed text	text/plain	text
pretty-printed Web page	text/html	html

In some cases, the specification for a specific format may be parameterised (e.g., the media type may include optional semi-colon and additional key-value parameters). A DAL service must accept a RESPONSEFORMAT parameter indicating a format that the service supports and should fail (Section 5.2) where the RESPONSEFORMAT parameter specifies a format not supported by the service implementation.

A concrete DAL service specification will specify any mandatory or optional formats as well as new formats not listed above; it may also place limitations on the structure for formats that are flexible. For example, a resource that responds with tabular output may impose a limitation that

FITS files only contain FITS tables, possibly only of specific types (ascii or binary).

If a client requests a format by specifying the media type (as opposed to one of the short forms), the response that delivers that content must set that media type in the Content-Type header. This is only an issue when a format has multiple acceptable media types (e.g., VOTable). This allows the client to control the Content-Type so that it can reliably cause specific applications to handle the response (e.g., a browser rendering a VOTable generally requires the text/xml media type). If the client requests a plain media type (e.g., not parameterised) and the media type does allow optional parameters, the service may respond with a parameterised media type to more clearly describe the output. For example, the text/csv media type allows two optional parameters: charset and header. If the request includes RESPONSEFORMAT=text/csv the response could have Content-Type text/csv or text/csv;header=absent at the discretion of the service. If the request specifies specific values for parameters, the response must be equivalent.

Individual DAL services (not just specifications) are free to support custom formats by accepting non-standard values for the RESPONSEFORMAT parameter.

The RESPONSEFORMAT parameter should not be confused with the FORMAT parameter used in many DAL services. The latter is generally used as a query parameter to search for data in the specified format; FORMAT and RESPONSEFORMAT have the same sense in TAP-1.0, but this is not generally the case.

The RESPONSEFORMAT parameter is always single-valued.

4.3.4 MAXREC

For resources performing discovery (querying for an arbitrary number of records), the resource must accept a MAXREC parameter specifying the maximum number of records to be returned. If MAXREC is not specified in a request, the service may apply a default value or may set no limit. The service may also enforce a limit on the value of MAXREC that is smaller than the value in the request. If the size of the result exceeds the resulting limit, the service must only return the requested number of rows. If the result set is truncated in this fashion, it must include an overflow indicator where possible as specified in Section 5.4.1.

The service must support the special value of MAXREC=0. This value indicates that, in the event of an otherwise valid request, a valid response

be returned containing metadata, no results, and an overflow indicator. The service is not required to execute the request and the overflow indicator does not necessarily mean that there is at least one record satisfying the query. The service may perform validation and may try to execute the request, in which case a MAXREC=0 request can fail.

The MAXREC parameter is always single-valued.

4.3.5 UPLOAD

The UPLOAD parameter is used to reference read-only external resources (typically files) via their URI, to be uploaded for use as input resources to the query. The value of the UPLOAD parameter is a resource name-URI pair. For example:

```
UPLOAD=table1,http://example.com/t1
```

would define an input named table1 at the given URI. Resource names must be simple strings made up of alphabetic, numeric, and the underscore characters only and must start with an alphabetic character.

Services that implement UPLOAD must support http or https as a URI scheme. A VOSpace URI (vos:<something>) is a more generic example of a URI that requires more service-side functionality; support for the vos scheme is optional.

To upload a resource inline, the caller specifies the UPLOAD parameter (as above) using a special URI scheme *param*. This scheme indicates that the value after the colon will be the name of the inline content. The content type used is multipart/form-data, using a *file* type input element. The *name* attribute must match that used in the UPLOAD parameter.

For example, in the POST data we would have this parameter:

```
UPLOAD=table3,param:t3
```

and this content:

```
Content-Type: multipart/form-data; boundary=AaB03
[...]
--AaB03x
Content-disposition: form-data; name="t3"; filename="t3.xml"
Content-type: application/x-votable+xml
[...]
```

```
--AaB03x
[...]
```

If inline upload is used by a client, the client must POST both the UPLOAD parameter and the associated inline content in the same request. Services that implement upload of resources must support the param scheme for inline uploads.

In principle, any number of resources can be uploaded using the UPLOAD parameter and any combination of URI schemes supported by the service as long as they are assigned unique names in the request. For example:

```
UPLOAD=table1,http://example.com/t1.xml
UPLOAD=image1,vos://example.authority!tempSpace/foo.fits
UPLOAD=table3,param:t3
```

Services may limit the size and number of uploaded resources; if the service refuses to accept the upload, it must respond with an error as described in Section 5.2. Concrete service specifications will typically provide a mechanism for referring to uploaded resources (e.g. in other request parameters) where necessary.

4.3.6 RUNID

The service should implement the RUNID parameter, used to tag service requests with the identifier of a larger job of which the request may be part. The RUNID value is a string with a maximum length of 64 characters.

For example, if a cross match portal issues multiple requests to remote services to carry out a cross-match operation, all would receive the same RUNID, and the service logs could later be analysed to reconstruct the service operations initiated in response to the job. The service should ensure that RUNID is preserved in any service logs and should pass on the RUNID value in calls to other services made while processing the request.

The RUNID parameter is always single-valued.

5 Responses

All DAL service requests eventually (after zero or more HTTP redirects) result in one of three kinds of responses: successful HTTP status code (200) and a service- and resource-specific representation of the results, or an HTTP

status code and a standard error document (see below), or an HTTP status code and a service- and resource-specific error document.

5.1 Successful Requests

Successfully executed requests must eventually (after zero or more redirects) result in a response with HTTP status code 200 (OK) and a response in the format requested by the client (Section 4.3.3) or in the default format for the service. The service should set HTTP headers (Fielding and Gettys et al., 1999) that are useful to the correct values where possible. Recommended headers to set when possible:

Content-Type
Content-Encoding
Content-Length
Last-Modified

For jobs executed using a DALI-async resource, the result(s) must be made available as child resources of the result list and directly accessible there. For jobs that inherently create a fixed result, service specifications may specify the name of the result explicitly. For example, TAP-1.0 has a single result and it must be named result, e.g.:

```
GET http://example.com/base/joblist/123/results/result
```

For concrete DAL service specifications where multiple result files may be produced, the specification may dictate the names or it may leave it up to implementations to choose suitable names.

5.2 Errors

If the service detects an exceptional condition, it must return an error document with an appropriate HTTP-status code. DAL services distinguish three classes of errors:

- Errors in communicating with the DAL service
- Errors in the use of the specific DAL protocol, including an invalid request
- Errors caused by a failure of the service to complete a valid request

Error documents for communication errors, including those caused by accessing non-existent resources, authentication or authorization failures, services being off-line or broken are not specified here since responses to these errors may be generated by other off-the-shelf software and cannot be controlled by service implementations. There are several cases where a DAL service could return such an error. First, a DALI-async resource must return a 404 (not found) error if the client accesses a job within the UWS job list that does not exist, or accesses a child resource of the job that does not exist (e.g., the error resource of a job that has not run and failed, or a specific result resource in the result list that does not exist). Second, access to a resource could result in an HTTP 401 (not authorized) response if authentication is required or an HTTP 403 (forbidden) error if the client is not allowed to access the requested resource. Although UWS is currently specified for HTTP transport only, if it were to be extended for use via other transport protocols, the normal mechanisms of those protocols should be used.

An error document describing errors in use of the DAL service protocol may be a VOTable document (Ochsenbein and Taylor et al., 2019) or a plain text document. The content of VOTable error documents is described in Section 5.4 below. Service specifications will enumerate specific text to be included. For plain text error documents the required text would be included at the start of the document; for VOTable error documents, the required (and optional) text would be included as content of the INFO element described in Section 5.4.2. In either case, DAL services will allow service implementers to add additional explanatory text after the required text (on the same line or on subsequent lines). In all cases, these are errors that occur when the job is executed and do not override any error behaviour for a UWS resource which specifies the behaviour and errors associated with interacting with the job itself.

If the invalid job is being executed using a DALI-async resource, the error document must be accessible from the <DALI-async>/<jobid>/error resource (specified by UWS) and when accessed via that resource it must be returned with an HTTP status code 200, e.g.:

```
GET http://example.com/base/joblist/123/error
```

For DALI-async errors, services should recommend and may mandate that required text be included in the error summary field of the UWS job in addition to the error document; this permits generic UWS clients to consume the standard part of the error description.

If the error document is being returned directly after a DALI-sync request, the service should use a suitable error code to describe the failure and include the error document in the body of the response. The Content-Type header will tell the client the format of the error document that is included in the body of the response. In general, HTTP status codes from 400-499 signify a problem with the client request and status codes greater than or equal to 500 signify that the request is (probably) valid but the server has failed to function. For transport protocols other than HTTP, the normal error reporting mechanisms of those protocols should be used.

5.3 Redirection

A concrete DAL service specification may require that HTTP redirects (302 or 303) be used to communicate the location of an alternate resource which should be accessed by the client via the HTTP GET method. For example, the UWS pattern used for DALI-async (2.1) requires this behaviour. Even when not required, concrete DAL service specifications must allow implementers to use redirects and clients must be prepared to follow these redirects using normal HTTP semantics (Fielding and Gettys et al., 1999).

5.4 Use of VOTable

VOTable is a general format. In DAL services we require that it be used in a particular way. The result VOTable document must comply with VOTable v1.2 or later versions (Ochsenbein and Taylor et al., 2019).

The VOTable format permits table creators to add additional metadata to describe the values in the table. Once a standard for including such metadata is available, service implementers should use such mechanisms to augment the results with additional metadata. Concrete DAL service specifications may require additional metadata of this form.

The VOTable must contain one ***RESOURCE*** element identified with the attribute `type="results"`; this resource contains the primary result (e.g., the only result for simple DAL services). Concrete DAL service specifications define what goes into the primary result. The primary ***RESOURCE*** element must contain, before the ***TABLE*** element, an ***INFO*** element with attribute *name* valued *QUERY_STATUS*. The value attribute must contain one of the following values:

QUERY_STATUS	**Interpretation**
OK	the job executed successfully and the result is included in the resource
ERROR	an error was detected at the level of the protocol, the job failed to execute, or an error occurred while writing the table data
OVERFLOW	the job executed successfully, the result is included in the resource, and the result was truncated

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 *RESOURCE* elements may be present, but the usage of any such elements is not defined here. Concrete DAL service specifications may define additional resources (and the type attribute to describe them) and service implementers are also free to add their own.

5.4.1 Overflow

If an overflow occurs (for example, result exceeds MAXREC) and the output format is VOTable, the service must include an *INFO* element in the *RESOURCE* with `name="QUERY_STATUS"` and the `value="OVERFLOW"`. If the initial *INFO* element (above) specified the overflow, no further elements are needed, e.g.:

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

If the initial *INFO* element specified a status of OK then the service must append an *INFO* element for the overflow after the table, e.g.:

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

There is no defined mechanism to indicate overflow (truncation) of output for formats other than VOTable. Specifically, simple text formats like `text/csv` and `text/tab-separated-values` do not support indicating overflow.

In general, services may truncate the output results when reaching a limit. A default or user-specified value of the MAXREC parameter defined in Section 4.3.4 is a common mechanism that causes truncation of results, but service providers may also impose limits in services that do not use MAXREC and indicate that the limit was reached with the overflow indicator.

5.4.2 Errors

If an error occurs, the service must include an *INFO* element with `name="QUERY_STATUS"` and the `value="ERROR"`. If the initial info element (above) specified the error, no further elements are needed, e.g.:

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

If the initial *INFO* element specified a status of OK then the service must append an *INFO* element for the error after the table, e.g.:

```
<RESOURCE type="results">
<INFO name="QUERY_STATUS" value="OK"/>
...
<TABLE>...</TABLE>
<INFO name="QUERY_STATUS" value="ERROR">
unexpected IO error while converting something
</INFO>
</RESOURCE>
```

The use of trailing *INFO* element allows a service to stream output and still report overflows or errors to the client. The content of these trailing INFO elements is optional and intended for users; client software should not depend on it.

5.4.3 Additional Information

Additional *INFO* elements should 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. For example:

```
<RESOURCE type="results">
 ...
<INFO name="standardID" value="ivo://ivoa.net/std/SIA"/>
 ...
</RESOURCE>
```

The following names for *INFO* elements should be used if applicable, but this list is not definitive.

Info Name	Value Interpretation
standardID	IVOA standardID for the service specification
citation	Reference to a publication that can/should be referenced if the result is used

The standardID of a service specification is the IVOA resource identifier for the StandardsRegExt record not including capability-specific fragments. For example, a VOTable produced by the `ivo://ivoa.net/std/SIA#query-2.0` capability would use the base `ivo://ivoa.net/SIA` (as in the above example) to say that the VOTable was produced by a SIA service.

For citations, the *INFO* element should also include a *ucd* attribute with the value *meta.bib* (if the value is a free-text reference) or *meta.bib.bibcode* (if the value is a bibcode). If other *meta.bib* UCDs are added to the vocabulary in future, they may also be used to describe the value.

A Changes

A.1 PR-DALI-1.2

- Clarified that truncation indicated by OVERFLOW can occur independent of MAXREC
- added new xtypes: hms, dms, moc, multipolygon, range, shape, uri, uuid
- changed VOSI-availability to optional
- changed VOSI-capability so it is only required for registered services

A.2 PR-DALI-1.1-20170412

- Changed vocabulary URI from ivo://ivoa.net/std/DALI#examples to http://www.ivoa.net/rdf/examples#. While this, in theory, is an incompatible change, no client has so far actually used the old vocabulary URI, and the old URI has very unintended consequences (e.g., an examples query term in DALI's registry record). Hence, we consider the change benign for a point release.

A.3 PR-DALI-1.1-20161101

- added explicit allowance for the use of non-standard xtypes with an arbitrary prefix

A.4 WD-DALI-1.1-20160920

- modified timestamp serialisation to allow the Z timezone indicator for civil time values
- specified counter-clockwise winding direction for polygons

A.5 WD-DALI-1.1-20160415

- Removed introductory language on including capability-propertied elements in examples.
- Expanded section on intervals to allow use of all integer and floating point datatypes supported by VOTable; only floating point intervals support open-ended intervals.
- Expanded section on geometry to allow use of `datatype="float"` in addition to double.
- Removed restrictions on the resource name and location for VOSI-availability resource.
- Removed restrictions on resource name for VOSI-tables resources.
- Fixed the timestamp format specification to correctly specify optional parts.
- Added reference to RFC2616 and minimised discussion of HTTP headers.

A.6 WD-DALI-1.1-20151027

Removed the requirement that the REQUEST must be a standard parameter. It is now recommended if a service capability supports more than one mode or operation. Removed VERSION parameter following experiences with TAP-1.1 prototypes.

Re-organised the section on literal values and clarified that these rules are intended to make input (parameters and other input docs) and output (response documents like VOTable) of services consistent. Added specification of interval, point, circle, and polygon literal values and specified VOTable xtype values for serialising such values in VOTable. Added VOTable serialisation and xtype for timestamp values. (Needed by SIA-2.0 and TAP-1.1)

Added bibtex cross-references.

A.7 PR-DALI-1.0-20130919

The following changes are in response to additional RFC commands and during the TCG review.

New architecture diagram and minor editorial changes to improve document.

Clarified RESPONSEFORMAT text to allow services to append mime-type parameters if the client did not specify them.

Relaxed the VERSION parameter so services should default to latest (instead of must) and to not differentiate between REC and pre-REC status.

Clarified the requirement for a VOTable RESOURCE with `type="results"` attribute so it is clear that this is the primary result and other RESOURCES may be present.

Clarified that HTTP-specific rules apply to RESTful web services and that although we describe such services here we do not preclude future use of other transport protocols.

A.8 PR-DALI-1.0-20130521

The following changes are in response to comments from the RFC period.

Made editorial changes from the DALI wiki page that were missed during WG review.

Changed all cross-references to be readable text.

Replaced example curl output from a POST with explanatory text.

POST of job parameters directly to job: restricted to creation and /parameters resource

Changed number of DALI-async and DALI-sync resources to zero or more.

Clarified that job parameters are the same if the type of job is the same, but services can have different types of jobs (and hence different parameters) on different job-list resources.

Fixed text forbidding any other vocab attributes in DALI-examples document.

Replace http-action and base-url with something or add sync vs async: replaced with capability property

Preventing loops with continuation in examples: removed.

Clarified that VOSI-capabilities does not require a registry extension schema and use of xsi:type.

Explicitly require that if VOSI-tables is not implemented, the service responds with a 404.

Clarified the purpose of requiring the service to use client-specified RESPONSEFORMAT as the Content-Type of the response.

Attempted to clarify the acceptable use of status codes for errors.

Removed single-table restriction from votable usage.

Clarified interpretation of dates and times as UTC timescale by default but permitting specific metadata to be specified.

Removed formatting of example links so they are not real hyperlinks in output documents.

Clarified that services can enforce a smaller limit than a requested MAXREC.

Removed text refering to IVOA notes on STC and Photometric metadata; added more general text that services should include additional metadata once standards for such are in place.

Explain the table at start of section 2.

Clarify requests that effect UWS job phase in DALI-async.

Removed malformed http post example from DALI-async section.

Remove reference to SGML specifically, but mention HTML5 as a poor choice for DALI-examples.

Add reference to RFC2616 in the RESPONSEFORMAT section since it talks about mimetypes.

Clarified text about setting job parameters and banned posting parameters directly to the job URL.

Replaced the base-url and http-action properties with a single capability property in DAL-examples. Changed the vocab identifier to be the IVOID for DALI with fragment indicating the DALI-examples section of the document.

A.9 WD-DALI-1.0-20130212

Simplified DALI-examples to conform to RDFa-1.1 Lite in usage of attributes.

References

Berners-Lee, T., Fielding, R. and Masinter, L. (2005), 'Uniform Resource Identifier (URI): Generic syntax', RFC 3986.
http://www.ietf.org/rfc/rfc3986.txt

Biron, P. and Malhotra, A. (2004), 'XML schema part 2: Datatypes second edition', W3C Recommendation.
http://www.w3.org/TR/xmlschema-2/

Bonnarel, F., Dowler, P., Demleitner, M., Tody, D. and Dempsey, J. (2017), 'IVOA Server-side Operations for Data Access Version 1.0', IVOA Recommendation 17 May 2017, arXiv:1710.08791.
http://doi.org/10.5479/ADS/bib/2017ivoa.spec.0517B

Demleitner, M., Gray, N. and Taylor, M. (2021), 'Vocabularies in the VO Version 2.0', IVOA Recommendation 25 May 2021.
http://doi.org/10.5479/ADS/bib/2021ivoa.spec.0525D

Demleitner, M., Plante, R., Stébé, A., Benson, K., Dowler, P., Graham, M., Greene, G., Harrison, P., Lemson, G., Linde, T. and Rixon, G. (2021), 'VODataService: A VOResource Schema Extension for Describing Collections, Services Version 1.2', IVOA Recommendation 02 November 2021.
http://doi.org/10.5479/ADS/bib/2021ivoa.spec.1102D

Dowler, P., Bonnarel, F. and Tody, D. (2015), 'IVOA Simple Image Access Version 2.0', IVOA Recommendation 23 December 2015.
http://doi.org/10.5479/ADS/bib/2015ivoa.spec.1223D

Dowler, P., Evans, J., Arviset, C., Gaudet, S. and Technical Coordination Group (2021), 'IVOA Architecture Version 2.0', IVOA Endorsed Note 01 November 2021.
http://doi.org/10.5479/ADS/bib/2021ivoa.spec.1101D

Dowler, P., Rixon, G. and Tody, D. (2010), 'Table Access Protocol Version 1.0', IVOA Recommendation 27 March 2010, arXiv:1110.0497.
http://doi.org/10.5479/ADS/bib/2010ivoa.spec.0327D

Fernique, P., Nebot, A., Durand, D., Baumann, M., Boch, T., Greco, G., Donaldson, T., Pineau, F.-X., Taylor, M., O'Mullane, W., Reinecke, M. and Derrière, S. (2022), 'MOC: Multi-Order Coverage map Version 2.0', IVOA Recommendation 27 July 2022.
https://ui.adsabs.harvard.edu/abs/2022ivoa.spec.0727F

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. (1999), 'Hypertext transfer protocol – HTTP/1.1', rfc2616.
http://www.w3.org/Protocols/rfc2616/rfc2616.html

Fielding, R. T. (2000), Architectural Styles and the Design of Network-based Software Architectures, PhD thesis, University of California, Irvine.
http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

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.
http://doi.org/10.5479/ADS/bib/2017ivoa.spec.0524G

Hanisch, R. J., Farris, A., Greisen, E. W., Pence, W. D., Schlesinger, B. M., Teuben, P. J., Thompson, R. W. and Warnock, III, A. (2001), 'Definition of the Flexible Image Transport System (FITS)'.
http://doi.org/10.1051/0004-6361:20010923

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

Leach, P., Mealling, M. and Salz, R. (2005), 'A Universally Unique IDentifier (UUID) URN namespace', RFC 4122.
https://www.ietf.org/rfc/rfc4122.txt

Ochsenbein, F., Taylor, M., Donaldson, T., Williams, R., Davenhall, C., Demleitner, M., Durand, D., Fernique, P., Giaretta, D., Hanisch, R., McGlynn, T., Szalay, A. and Wicenec, A. (2019), 'VOTable Format Definition Version 1.4', IVOA Recommendation 21 October 2019.
http://doi.org/10.5479/ADS/bib/2019ivoa.spec.1021O

Rots, A. H. (2007), 'Space-Time Coordinate Metadata for the Virtual Observatory Version 1.33', IVOA Recommendation 30 October 2007, arXiv:1110.0504.
http://doi.org/10.5479/ADS/bib/2007ivoa.spec.1030R

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