International
Virtual
Observatory
Alliance

IVOA Server-side Operations for Data Access

Version 1.0

IVOA Working Draft 2015-12-12

Working group
DAL

This version
http://www.ivoa.net/documents/SODA/20151212

Latest version
http://www.ivoa.net/documents/SODA

Previous versions
WD-AccessData-1.0-20151021
WD-AccessData-1.0-20140730
WD-AccessData-1.0-20140312

Author(s)
François Bonnarel, Markus Demleitner, Patrick Dowler, Douglas Tody

Editor(s)
François Bonnarel

Abstract

This document describes the SODA web service capability. SODA is a low-level data access capability or server side data processing that can act upon the data files, performing various kinds of operations: filtering/subsection, transformations, pixel operations, and applying functions to the data.

Status of This Document

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

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

Contents

Acknowledgments

The authors would like to thank all the participants in DAL- WG discussions for their ideas, critical reviews, and contributions to this document.

1 Introduction

The SODA web service interface defines a RESTful web service for performing server-side operations and processing on data before transfer.

1.1 The Role in the IVOA Architecture

TODO: new diagram from TCG

SODA services conform to the Data Access layer Interface (Dowler et al., 2013) specification, including the Virtual Observatory Support Interfaces (Grid and Web Services Working Group, 2011) resources.

1.2 Motivating Use Cases

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

1.2.1 Retrieve Subsection of a Datacube

Cutout a subsection using coordinate axis values. The input to the cutout operation will include one or more of the following:

- a region on the sky
- an energy value or range
- a time value or range
- one or more polarization states

The region on the sky should be something simple: a circle, a range of coordinate values, or maybe a polygon.

1.2.2 Retrieve subsection of a 2D Image

This is a special case of 1.2.1, where the cutout is only in the spatial axes.

1.2.3 Retrieve subsection of a Spectrum

This is a special case of 1.2.1, where the cutout is only in the spectral axis.

1.3 Provide the data in different formats

Examples are images in PNG, or JPEG instead of FITS and spectra in csv, FITS or VOTable.

1.3.1 Flatten a Datacube into a 2D Image

This use case will be developed and supported in the SODA-1.1 (or later) specification.

1.3.2 Flatten a Datacube into a 1D Spectrum

This use case will be developed and supported in the SODA-1.1 (or later) specification.

1.3.3 Rebin Data by a Fixed Factor

This use case will be developed and supported in the SODA-1.1 (or later) specification.

1.3.4 Reproject Data onto a Specified Grid

This use case will be developed and supported in the SODA-1.1 (or later) specification.

1.3.5 Compute Aggregate Functions over the Data

This use case will be developed and supported in the SODA-1.1 (or later) specification.

1.3.6 Apply Standard Function to Data Values

It could be "denoising" with standard methods or "on the fly" recalibration. This use case will be developed and supported in the SODA-1.1 (or later) specification.

1.3.7 Apply Arbitrary User-Specified Function to Data Values

This use case will be developed and supported in the SODA-1.1 (or later) specification.

1.3.8 Run Arbitrary User-Supplied Code on the Data

This use case will be developed and supported in the SODA-1.1 (or later) specification.

1.4 SODA Operation

In contrast to other IVOA protocols, SODA services are not usually discovered through Registry queries. Instead, clients encounter them in Datalink (Dowler et al., 2015) declarations, which can either be standalone or embedded within other services' responses.

Since this pattern can appear somewhat confusing at first, this introductory (non-normative) chapter discusses the usage scenarios for SODA services. In parallel, we provide advice on the server-side implications of these scenarios.

In all cases, the first step is a data discovery service; when used below, this term could refer to, for instance, SIA, SSA, or ObsTAP, but also to some sort of resolution engine for persistent identifiers.

1.4.1 Pure Datalink discovery

In the baseline scenario, the data discovery step has yielded a result with the media type

`application/x-votable+xml;content=datalink`.

To the client, this indicates that what is given in the access reference (e.g., the `access_url` column in ObsTAP or SIA version 2 or the column with the UCD `VOX:Image_AccessReference` in SIA version 1) is a datalink document. Within that document, there is a SODA service descriptor written as specified by Datalink. The whole document would looks somewhat like this:

of course, this needs descriptions and ranges; if this text is accepted for the main standard, MD will fill this in.

```
<RESOURCE type="results">
  [datalink links, one of them being]
  <TR>[id=ivo://example.com/data?ds1 service-def=soda; semantics=#proc]</TR>
</RESOURCE>

<RESOURCE type="meta" utype="adhoc:service" ID="soda">

  <PARAM name="standardID" datatype="char" arraysize="*"
       value="ivo://ivoa.net/std/SODA#sync-1.0" />

  <PARAM name="accessURL" datatype="char" arraysize="*"
       value="http://example.com/my-svcs/soda/sync?ID=ivo://example.com/data?ds1" />

  <GROUP name="inputParams">
     <PARAM name="POS" ucd="phys.angArea;obs" datatype="char"
          arraysize="*" />
     <PARAM name="BAND" ucd="em.wl" unit="m" datatype="double"
          arraysize="*"/>
     <PARAM name="TIME" ucd="time.interval;obs.exposure"
          unit="d" datatype="double"
          arraysize="*" xtype="interval" />
     <PARAM name="POL" ucd="meta.code;phys.polarization" datatype="char"
          arraysize="*" />
  </GROUP>
</RESOURCE>
```

Of course, the service is free to choose the VOTable ID of the resource with the utype `adhoc:service`; the service will only declare the parameters it (and the underlying data) actually supports.

From the Datalink row with `#proc` semantics, the client sees that there is a service for the dataset in question (identified here through its publisher DID, `ivo://example.com/data?ds1`), and from the service descriptor's standardID *`PARAM`* it learns that the service's parameters follow the rules laid down here, in particular as regards the three-factor semantics. For instance, the client is guaranteed that BAND, with UCD `em.wl` and unit meters actually denotes the parameter controlling where a cutout on the spectral axis will happen.

SODA's role here is exactly this guarantee of a specific semantics, as opposed to a non-standard service that could use BAND in an entirely different way.

An attractive implementation strategy for small-to-medium sized installations is to pre-generate the datalink files. In that way, no extra endpoint is required besides the discovery service and the SODA service.

If people think this is a good idea, I'll do SVGs of these

Here is a sketch of the query pattern in this case:

```
Client ---- discovery query ----> DAL service
                                        |
     +----- Results with ------------+
     |      Datalink-valued accrefs
     v
 DAL client --- retrieves accref ---> e.g., plain HTTP
                                                service
                                                  |
    +-------- Datalink document with -----------+
    |         SODA descriptor
    v
 SODA client -----> SODA instructions ----> SODA service
                                                  |
Data viewer <------ sliced-and-diced data -----+
```

1.4.2 Datalink Discovery with Backward Compatiblity

The problem with the scheme discussed in sect. 1.4.1 is that legacy clients, i.e., those that do not understand Datalink, will not be able to interpret the results of the discovery step. While this is probably desirable when services hand out large data cubes that legacy clients probably will not properly handle anyway, in many other situations services should deliver conventional (e.g., FITS) data products to such legacy clients. To still enable SODA and other Datalink functionality, DAL services can add a serivce descriptor in the DAL response that indicates the availability of a Datalink *service* accompanying the DAL service, looking more or less like this:

```
<RESOURCE type="results">
  [a result from services like TAP, SIA, SSA]
  <TABLE>
    [in particular, we have one field like]
    <FIELD ID="primaryID" name="pubDID" datatype="char" arraysize="*">
      <DESCRIPTION>The publisher DID for the dataset</DESCRIPTION>
    </FIELD>
    ...
  </TABLE>
</RESOURCE>
<RESOURCE type="meta" utype="adhoc:service">
```

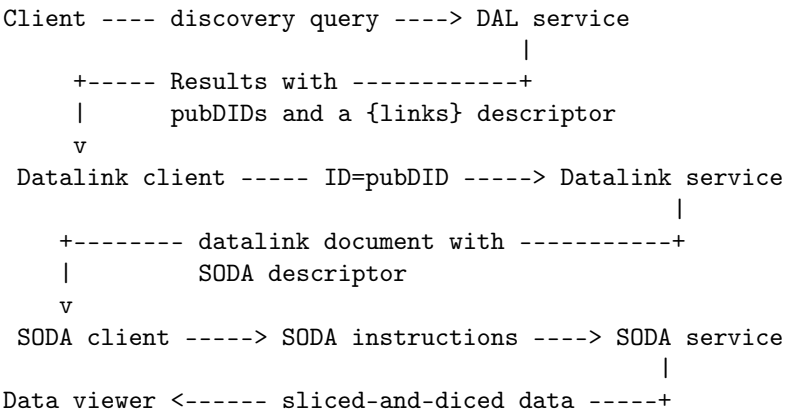
```
  <PARAM name="standardID" datatype="char" arraysize="*"
    value="ivo://ivoa.net/std/DataLink#links-1.0" />
  <PARAM name="accessURL" datatype="char" arraysize="*"
    value="http://example.com/mylinks/get" />
  <GROUP name="inputParams">
    <PARAM name="ID" datatype="char" arraysize="*"
      value="" ref="primaryID"/>
  </GROUP>
</RESOURCE>
```

Note that while this looks very similar to the SODA descriptor above, this fragment is in the DAL response rather than within a Datalink document itself, and it also describes a Datalink rather than a SODA service.

It references one (or more) field(s) from the DAL response.[1] This is explained in more detail in section 4.2 of the Datalink recommendation 1.0. The net result is that datalink-enabled clients can find ancillary data and use SODA services for data access by virtue of being able to retrieve Datalink documents, whereas legacy clients still retain basic functionality.

On the service side, this incurs the additional cost of having to provide a datalink {links} resource, on the client side, some extra dereferencing becomes necessary. Hence, this pattern should be preferred over the simpler pattern from sect. 1.4.1 only if there is a significant advantage in serving data to legacy clients.

The query pattern in this case looks like this:



[1] That pattern can be used within the Datalink document as in sect. 1.4.1, too, to refer to Datalink's ID column, which lets services use a constant access URL in the SODA descriptor.

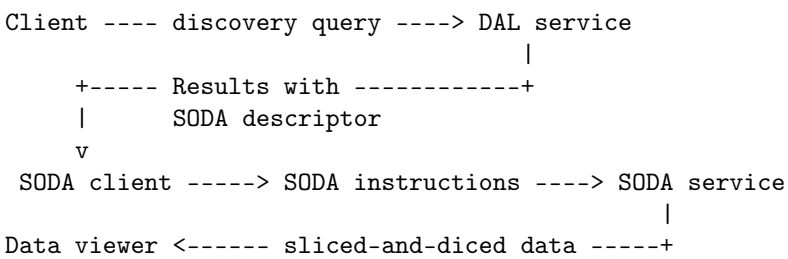
1.4.3 Sidestepping Datalink

In some situations, the extra request to retrieve the datalink document for each dataset is inconvenient, while the client may have sufficient information to operate the SODA service based on common metadata. A classic example would be a service containing relatively homogeneous results of a single instrument, perhaps a spectrograph where all spectra essentially have the same spectral coverage and a client may want to only retrieve, say, the vicinity of a spectral line.

In such cases a service may provide a shortcut by including a SODA descriptor directly in the DAL response. In essence the resulting descriptor looks like a union of the one given in sect. 1.4.1 and the one given in sect. 1.4.2: It includes the SODA parameters, the ID parameter with the reference to the column to take the publisher DID from, but it has a SODA standardID from sect. 1.4.1 rather than the Datalink one from sect. 1.4.2.

While sidestepping the extra datalink request might appear attractive in principle, the difficulty of determining the useful parameter ranges make this pattern only interesting in relatively few special cases. Clients must not rely on the presence of full SODA descriptors in DAL responses. Normal SODA operation follows the pattern given in sects. 1.4.1 and 1.4.2.

The query pattern here is:



2 Resources

SODA services are implemented as HTTP REST (Richardson and Ruby, 2007) web services with a {sync} resource that conforms to the DALI- sync resource description.

A stand-alone SODA service may have one or both of the {sync} and {async} resources. For either type, it could have multiple resources (e.g. to support alternate authentication schemes). The SODA service may also include other custom or supporting resources.

Either the {sync} or {async} SODA capability may be included as part of other web services. For example, a single web service could contain the SIA-2.0 {query} capability, the DataLink-1.0 {links} capability, and the SODA {sync} capability. Such a service must also have the VOSI-availability and

resource type	resource name	required
{sync}	service specific	
{async}	service specific	
DALI-examples	/examples	no
VOSI-availability	/availability	yes
VOSI-capabilities	/capabilities	yes

Table 1: Endpoints for AccessData services

VOSI-capabilities resources to report on and describe all the implemented capabilities.

2.1 {sync} resource

The {sync} resource is a synchronous web service resource that conforms to the DALI-sync description. Implementors are free to name this resource however they like, except that the name must consist of one URI segment only (i.e., contain no slash). This is to allow clients, given the access URL, can reliably find out the URL of the capabilities endpoint. Clients, in turn, can find the resource path using the VOSI-capabilities resource, but will in general be provided the access URLs through a previous data discovery query or through direct user input.

The {sync} resource performs the data access as specified by the input parameters and returns the data directly in the output stream. Synchronous data access is suitable when the operations can be quickly performed and the data stream can be setup and written to (by the service) in a short period of time (e.g. before any timeouts).

2.2 {async} resource

The {async} resource is an asynchronous web service resource that conforms to the DALI-async description. The considerations on naming the resource given in sect. 2.1 apply for it.

The {async} resource performs the data access as specified by the input parameters and either (i) stores the results for later transfer or (ii) pushes the results to a specified destination (e.g. to a VOSpace location). Asynchronous data access usually introduces resource constraints on the service (which may be limited) and usually imposes a higher latency before any results can be seen because the location of results does not have to be valid until the data access job is complete. Asynchronous data access is intended for (but not limited to) use when the operations take considerable time and results must be staged (e.g. some multi-pass algorithms or operations that result in multiple outputs).

2.3 Examples: DALI-examples

SODA services should provide a DALI-examples resource with one example invocation that shows the variety operations the service can perform. Example operations using the {sync} resource and that output a small data stream are preferred, as the examples may be used by automatic validators doing relatively frequent (of order daily) queries.

Parameters to be passed to the service must be given using the DALI `generic-parameter` term.

2.4 Availability: VOSI-availability

A SODA web service must have a VOSI-availability resource (Grid and Web Services Working Group, 2011) as described in DALI (Dowler et al., 2013).

2.5 Capabilities: VOSI-capabilities

A web service that includes SODA capabilities must have a VOSI-capabilities resource (Grid and Web Services Working Group, 2011) as described in DALI (Dowler et al., 2013). The standardID for the {sync} resource is

`ivo://ivoa.net/std/SODA#sync-1.0`.

The standardID for the {async} resource is

`ivo://ivoa.net/std/SODA#async-1.0`.

All DAL services must implement the `/capabilities` resource. The following capabilities document shows the minimal metadata for a standalone SODA service and does not require a registry extension schema:

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

Note

As SODA builds upon several concepts of DataLink (Dowler et al., 2015), that document should be read before trying to understand the following material.

```
    </interface>
  </capability>
  <capability standardid="ivo://ivoa.net/std/SODA#sync-1.0">
    <interface xsi:type="vod:ParamHTTP" role="std" version="1.0">
      <accessurl use="full">
        http://example.com/data/sync
      </accessurl>
    </interface>
    <!-- service details from extension schema could go here -->
  </capability>
  <capability standardid="ivo://ivoa.net/std/SODA#async-1.0">
    <interface xsi:type="vod:ParamHTTP" role="std" version="1.0">
      <accessurl use="full">
        http://example.com/data/async
      </accessurl>
    </interface>
    <!-- service details from extension schema could go here -->
  </capability>
</capabilities>
```

Note that the {sync} and {async} resources do not have to be named as shown in the accessURL(s) above. Multiple capability elements for the {sync} and the {async} resources may be included; this is typically used if the differ in protocol (http vs. https) and/or authentication requirements.

2.6 Parameter Description and Three-Factor Semantics

2.6.1 Three-factor Semantics

Parameters in SODA are defined by triples of name, UCD, and unit (the "SODA triple"). Data services are free to support as many such parameters as is appropriate for their datasets, in addition to supporting standard parameters. With the three factors, it is unlikely that two operators will by accident use the same three factors for parameters of differing semantics. In this way, when a new standard parameter is adopted, no existing service should require changes to remain compliant.

With standard parameters as defined in this document, clients can rely on certain semantics and exploit that knowledge in the provision of special UIs or APIs. Standard parameters defined so far are given in table 2. Instructions

for how to propose additional standard parameters are given on the landing page of the IVOA DAL working group[2].

Name	UCD	Unit	Semantics
ID	meta.ref.url;meta.curation		cf. sect. 3.1.1
POS	phys.angArea;obs		cf. sect. 3.2.1
BAND	em.wl	m	cf. sect. 3.2.2
TIME	time.interval;obs.exposure	d	cf. sect. 3.2.3
POL	meta.code;phys.polarization		cf. sect. 3.2.4

Table 2: SODA triples for the standard parameters defined here.

Both standard and non-standard parameters should follow DALI conventions if at all possible. Roughly, float-valued parameters should be mapped to interval-valued parameters (i.e., do not split up minimum and maximum into separate parameters). Depending on their semantics, integer parameters should either be intervals or enumerated parameters (which typically can be repeated). String-valued parameters should always be enumerated.

2.6.2 Discovery of Supported Parameter. Implementation strategies

There are two principal ways a client will be led to a SODA service: From a service descriptor that is part of a DAL discovery response, or by direct access mediated or not via a DataLink service. In the latter case an empty query to the service iwill answer by giving the service descriptor. . In both cases, clients must obtain the set of supported parameters from the applicable service descriptor.

In contrast to previous IVOA DAL protocols, SODA is not a data discovery protocol but instead operates on concrete datasets. In a typical case, some combinations of parameters (e.g., a positional or spectral cutout) may yield no output at all, as the coverage of an individual dataset is very limited. To provide meaningful user interfaces, clients would therefore need detailed information on the service parameters, in particular as regards their domains (i.e., set of values that are likely to yield non-empty results). This, in regard, requires efficient and rich service APIs, not unlikely involving service-specific parameters.

But the double ways towards SODA service execution makes the providing of this information difficult to describe and is not done in this version.

As a consequence of this, when the SODA service is driven from a data discovery phase, parameters domains can be inferred by the end-user from the discovery reponse. In the other case, it may happen that these parameters have to be furnished blindly by the end-user.

[2]At the time of writing, this is http://wiki.ivoa.net/twiki/bin/view/IVOA/IvoaResReg.

2.6.3 SODA Service Descriptor

The DataLink (Dowler et al., 2015) specification describes a mechanism for describing a service within a VOTable resource and recommends that services can describe themselves with a special resource with `name="this"`. SODA responses for empty sync queries should include a descriptor describing both standard and custom query parameters (if applicable). The descriptor for a service with standard parameters (see sect. 3) would be:

```
<RESOURCE type="meta" utype="adhoc:service" name="this">

  <PARAM name="standardID" datatype="char" arraysize="*"
       value="ivo://ivoa.net/std/SODA#sync-1.0" />

  <PARAM name="accessURL" datatype="char" arraysize="*"
       value="http://example.com/SODA/sync" />

  <GROUP name="inputParams">
     <PARAM name="ID" ucd="meta.ref.url;meta.curation" datatype="char"
      arraysize="*" />
     <PARAM name="POS" ucd="phys.angArea;obs" datatype="char"
      arraysize="*" />
     <PARAM name="BAND" ucd="em.wl" unit="m" datatype="double"
      arraysize="*"/>
     <PARAM name="TIME" ucd="time.interval;obs.exposure"
      unit="d" datatype="double"
      arraysize="*" xtype="interval" />
     <PARAM name="POL" ucd="meta.code;phys.polarization" datatype="char"
      arraysize="*" />
  </GROUP>
</RESOURCE>
```

This VOTable resource should be output for empty sync queries; Thus all inputs and outputs would be fully described.

2.6.4 Client Handling of Discovered Parameters

To keep SODA clients useful even when advanced data products are being accessed, the following procedure should be followed by clients:

1. Obtain the parameter triples domains as described in sect. 2.6.2.
2. Identify standard parameters understood by the client and provide suitable input widgets (e.g., a rubberband on top of a sky rendering, or a `spatial_cutout` method possibly allowing library-specific coverage objects) for them.
3. For the remaining parameters, provide generic UI or API elements (e.g., sliders or text boxes with domains annotated for intervals, pop-up menus for enumerated values).

3 Parameters for {sync} and {async}

The {sync} and {async} resources accept the same set of parameters.

3.1 Common Parameters

3.1.1 ID

The ID parameter is used to specify the dataset or file to be accessed. The values for the ID parameter are generally discovered from data discovery or DataLink requests. The values must be treated as opaque identifiers that are used as-is. The DataLink specification (Dowler et al., 2015) describes mechanisms for conveying opaque parameters and values in service descriptor resources that can be used by clients to set the ID parameter.

The ID parameter is single-valued in {sync} requests, so {sync} soda requests access a single dataset or file. Multiple ID parameters may be submitted in {async} requests on order to bundle access to multiple datasets or files in a single job.

The ID ucd is "meta.ref.url;meta.curation", and its unit is blank. In addition its datatype "char", with arraysize "*".

3.2 Filtering Parameters

Filtering parameters are used to extract subsets of larger datasets or data files. In general, filtering parameters are single-valued in {sync} requests and multi-valued in {async} requests (exceptions noted below). When multiple values of filtering parameters are used in an {async} job, each combination of values produces zero or one result. For example, if an {async} job included two POS and two BAND values, there could be as many as four results (or fewer if some combinations do not produce a result because the filter does not intersect the bounds of the data).

3.2.1 POS

The POS parameter defines the positional region(s) to be extracted from the data. The value is made up of a shape keyword followed by coordinate values. The allowed shapes are:

Shape	Coordinate values
`CIRCLE`	`<longitude> <latitude> <radius>`
`RANGE`	`<longitude1> <longitude2> <latitude1> <latitude2>`
`POLYGON`	`<longitude1> <latitude1> ... (at least 3 pairs)`

Table 3: POS Values in Spherical Coordinates

As in DALI, open intervals use -Inf or +Inf as one limit.
Examples for POS values:

- A circle at (12,34) with radius 0.5:

  POS=CIRCLE 12 34 0.5

- A range of [12,14] in longitude and [34,36] in latitude:

  POS=RANGE 12 14 34 36

- A polygon from (12,34) to (14,34) to (14,36) to (12,36) and (implicitly) back to (12,34):

  POS=POLYGON 12 34 14 34 14 36 12 36

  The inside is always assumed to be the smaller of the region to the left and the region to the right so only polygons smaller than half the sphere can be specified.

- A band around the equator:

  POS=RANGE 0 360 −2 2

- The north pole:

  POS=RANGE 0 360 89 +Inf

All longitude and latitude values (plus the radius of the CIRCLE) are expressed in degrees in the ICRS. A future version of this specification may allow the use of other reference systems (specifically the native system of the data).

The POS parameter is single-valued for {sync} requests and multi-valued for {async} jobs.

Since it is string-valued, POS is unitless (although the values contained in the the string are all in decimal degrees). The ucd is “phys.angArea;obs”. However the datatype of the POS parameter is “char”.

3.2.2 BAND

The BAND parameter defines the energy interval(s) to be extracted from the data. The value is an open or closed numeric interval of values in the native spectral axis coordinate system and units of the data. The intervals always include the bounding values. As in DALI, open intervals use -Inf or +Inf as one limit.

If there is one single value the interval is assumed to be infinitely small (a scalar value).

- The closed interval [500,550]:

  BAND=500 550

- The open interval (-inf,300]:

  BAND=−Inf 300

- The open interval [750,inf):

  BAND=750 +Inf

- The scalar value 550, equivalent to [550,550]:

  BAND=550

Extracting using a scalar value should normally extract a single pixel along the energy axis of the data; extracting using an interval should extract one or more pixels.

All energy values are expressed as barycentric wavelength in meters. A future version of this specification may allow the use of other reference systems (specifically the native system of ther data).

The BAND parameter is single-valued for {sync} requests and multi-valued for {async} jobs.

The ucd of the BAND parameter is "em.wl", the unit is "m". Its datatype is double with an arraysize of 2, and the xtype is "interval" as defined in DALI.

3.2.3 TIME

The TIME parameter defines the time interval(s) to be extracted from the data. The value is an open or closed interval with either numeric values (interpreted as Modified Julian Dates). As in DALI, open intervals use -Inf or +Inf as one limit.

If there is one single value the numeric interval is assumed to be infinitely small (a scalar value).

- An open interval from the MJD 55100.0 and all later times:

  TIME= 55100.0 +Inf

- A range of MJD values:

  TIME=55123.456 55123.466

- An instant in time using Modified Julian Date:

TIME=55678.123456

Time values are always UTC. The TIME parameter is single-valued for {sync} requests and multi-valued for {async} jobs.

The ucd of the TIME parameter is "time.interval;obs.exposure" and the unit is "d". The datatype is "double" with an arraysize of 2, and the xtype is, again, "interval" as defined in DALI

3.2.4 POL

The POL parameter defines the polarization state(s) (Stokes) to be extracted from the data.

- Extract the unpolarized intensity:

  POL=I

- Extract the standard circular polarization:

  POL=V

- The POL parameter is multi-valued; multiple values can be included in a single request and all will be extracted. Extract only the IQU components:

  POL=I
  POL=Q
  POL=U

The POL is multi-valued for both {sync} and {async} requests. Unlike general filtering parameters, all values of POL are combined into a single filter; for example, if the request includes the three values above, the job would generate one result with some or all of these polarization states (per combination of ID and other filtering parameters).

The ucd of the POL PARAMETER is "meta.code;phys.polarization" and the unit is blank. The datatype is "char" with arraysize "*".

4 {sync} Responses

All responses from the {sync} resource follow the rules for DALI-sync resources, except that the {sync} response allows for error messages for individual input identifier values.

4.1 Successful Requests

Successfully executed requests should result in a response with HTTP status code 200 (OK) and a response in the format requested by the client or in the default format for the service.

If the values specified for cutout parameters do not include any pixels from the target dataset/file, the service must respond with HTTP status code 204 (No Content) and no response body.

The service should set the following HTTP headers to the correct values where possible.

Content-Type	mime-type of the response
Content-Encoding	encoding/compression of the response (if applicable)

Since the response is usually dynamically generated, the Content-Length and Last-Modified headers cannot usually be set.

4.2 Errors

The error handling specified for DALI-sync resources applies to service failure. Error documents should be text using the text/plain content-type and the text must begin with one of the following strings:

Error	General error (not covered below)
AuthenticationError	Not authenticated
AuthorizationError	Not authorized to access the resource
ServiceUnavailable	Transient error (could succeed with retry)
UsageError	Permanent error (retry pointless)

Table 4: error messages with their meaning

5 {async} Responses

The {async} resource conforms to the DALI-async resource description, which means the job is a UWS job with all the job control features available. All result files are to be listed as children of the UWS results resource. The service provider is free to name each result.

A Changes from Previous Versions

A.1 Changes from WD-SODA-1.0-20151212

- POS is now unitless
- Aligned parameter UCDs with what is in Obscore

- Removed gratuitous xtypes.

A.2 Changes from WD-SODA-1.0-20151120

Change the name of the protocol. Suppression of SELECT and COORD. xtype description are in DALI. Reference to this has been added.

A.3 Changes from WD-AccessData-1.0-20151021

Added general introduction on PARAMETER description to section 3. Modified SELECT and COORD sections in order to detach them from SimDal. Added Appendix on xtype description with BNF syntax.

A.4 Changes from WD-AccessData-1.0-20140730

- Removed REQUEST parameter since the DAL-WG decision to not include it when there is only one value.
- Clarified that ID and filierting parameters are single valued for {sync} and multi-valued for {async}, wth POL being multi-valued but still being treated as a single filter.

A.5 Changes from WD-AccessData-1.0-20140312

This is the initial document version.

References

Dowler, P., Bonnarel, F., Michel, L. and Demleitner, M. (2015), 'IVOA Recommendation: IVOA DataLink', IVOA Working Draft, arXiv:1509.06152.
URL: *http://www.ivoa.net/documents/DataLink/*

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

Grid and Web Services Working Group (2011), 'IVOA support interfaces version 1.0'.
URL: *http://www.ivoa.net/documents/VOSI/index.html*

Richardson, L. and Ruby, S. (2007), *RESTful Web Services*, O'Reilly.