International

Virtual

Observatory

Alliance

Simple Image Access Protocol

Version 2.0 (*draft Nov09)*

IVOA Working Draft 2009 November 4

This version:

Draft November 4, 2009

Latest version:

Draft November 4, 2009

Previous versions:

http://www.ivoa.net/Documents/cover/SIA-20090521.html (SIA V1.0)

Draft September 21, 2009

Editors:

D. Tody, F. Bonnarel

Authors:

F. Bonnarel, D. Durand, A. Micol, A. Richards, J. Salgado, D. Tody

Abstract

This is a draft-in-progress of version 2.0 of the Simple Image Access (SIA) protocol. SIA provides capabilities for both discovery and access to image data in the VO. Herein an *image* is defined as a regular array of sample values with associated metadata. Both two-dimensional images and multi-dimensional *image cubes* (*hypercubes*) containing some combination of spatial, spectral, time, and polarization axes are supported.

Status of This Document

This is an internal *Working Draft*. The document is incomplete at this point, and is intended only for discussion within the author group.

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/.

Acknowledgements

"Ack here, if any"

Contents

1 Introduction

To be added. Describe scope of SIAV2.

2 Interface Concepts (informative)

2.1 Images and Image Cubes

An ***image*** in SIA is a multidimensional array of sample values (*pixels* or *voxels*) with associated metadata, consistent with the "image" concept as used within astronomy (e.g., image as in the sense of a *FITS image*, although graphics images are possible as well). Sample values are typically physical measurements having an integer, floating, or complex flux value of some sort.

In principle images may be of any dimension. SIA deals primarily with two-dimensional (2D) images or with image "cubes" (images of at most three or four dimen-

sions, also sometimes referred to as *hypercubes*). In general when we refer to an "image" object we may mean either a 2D image or a multidimensional cube. Cubes typically have a spatial coordinate system on the first two image axes, and a spectral, time, or polarization coordinate system on the third or possibly fourth image axis. 2D images typically have two spatial axes although other combinations are possible, for example a longslit spectrum is a 2D image with one spatial and one spectral axis. In general any variation is possible which can be described with the associated *world coordinate system* (WCS). While the WCS described here are the most common ones for observational astronomy, more specialized WCS are possible, e.g., for solar and planetary data or for synthetic data from theoretical models.

Although a 1D image is also possible, usage of 1D images within astronomy is rare except for spectra, which have explicit support within VO via the SSA (simple spectral access) interface.

2.2 Virtual Data

A ***virtual dataset*** is a data object which can be described in a data query, but which may not physically exist until it is accessed, at which time it is created on the fly by the service. A typical example is a "cutout" (subset) of an image. Image access in the VO often deals with virtual data to offload computation to the service as well as to minimize the amount of data that has to be transferred over the network. In some cases this can make a tremendous difference, for example a data cube may be hundreds of gigabytes in size, but a client may require only a small region for analysis. Other services may generate images on the fly from more fundamental data, e.g., visibility or event data.

2.3 Region Of Interest (ROI)

Although SIA can be used to find and download entire images, often the client is only interested in a specific region of the space sampled by the individual image or data collection. We call this the ***region of interest*** (ROI). The ROI is defined by the spatial, spectral, and temporal boundaries of the region required by the client for analysis. In some cases the type of polarization desired is specified as well. While the ROI is commonly specified this does not have to be the case for data discovery queries: any element of the ROI may be omitted, or the entire ROI may be omitted so long as a valid query is posed (in particular the spatial region no longer needs to be specified as was the case in SIA version 1.0).

The specified ROI can be a simple search region (for a service which returns "archival" images), but in general it specifies the *ideal image* desired by the client. Often in data analysis scenarios the client application needs image data in a specified region of the sky. Ideally images will be available which exactly match the ROI. Delivering the "ideal image" may be possible when an image is computed on the fly by the service.

2.4 Data Derivation

Data can come from a variety of sources, and may go through various types of processing, including by the data access service itself, before being delivered to a client analysis application. It is important for analysis to understand the origin of the data and what processing it has undergone. To address this issue we introduce two new concepts, *data source* and *creation type*.

2.4.1 Data Source

The data source specifies where the data originally came from, that is, the data collection to which the service provides access. The following values are currently defined:

survey	A survey dataset, which typically covers some region of observational parameter space in a uniform fashion, with as complete as possible coverage in the region of parameter space observed.
pointed	A pointed observation of a particular astronomical object or field. Typically these are instrumental observations taken as part of some PI observing program. The data quality and characteristics may be variable, but the observations of a particular object or field may be more extensive than for a survey.
custom	Data which has been custom processed, e.g., as part of a specific research project.
theory	Theory data, or any data generated from a theoretical model, for example a synthetic image or spectrum.
artificial	Artificial or simulated data. This is similar to theory data but need not be based on a physical model, and is often used for testing purposes.

2.4.2 Creation Type

The creation type describes the process used to produce the dataset as returned by the service, from the data source. Typically this describes only the processing performed by the data service, but it could describe some additional earlier processing as well, e.g., if data is partially precomputed. The creation type is especially important for virtual data and for data which is derived from the parent data set by some complex form of processing. The following values are currently defined:

archival	The entire archival or project dataset is returned. Transformations such as metadata or data model mediation or format conversions may take place, but the content of the da-

	taset is not substantially modified (e.g., all the data is returned and the sample values are not modified).
cutout	The dataset is subsetted in some region of parameter space to produce a subset dataset. Sample values are not modified, e.g., cutouts could be recombined to reconstitute the original dataset.
filtered	The data is filtered in some fashion to exclude or alter portions of the dataset, e.g., passing only data in selected regions along a measurement axis, or processing the data in a way which recomputes the sample values, e.g., due to interpolation or flux transformation. Filtering is often combined with other forms of processing, e.g., projection.
mosaic	Data from multiple non- or partially-overlapping datasets are combined to produce a new dataset.
projection	Data is geometrically warped or dimensionally reduced by projecting through a multidimensional dataset.
imageExtraction	Extraction of an image from another dataset, e.g., on-the-fly generation of an image from more fundamental non-image data (as opposed to a cutout or projection of an existing image).
spectralExtraction	Extraction of a spectrum from another dataset, e.g., extraction of a spectrum from a spectral data cube through a simulated aperture (not relevant for SIA).
catalogExtraction	Extraction of a catalog of some form from another dataset, e.g., extraction of a source catalog from an image, or extraction of a line list catalog from a spectrum (not valid for a SIA service).

The full creation type may involve more than one of these operations, for example, both projection and filtered, or both image extraction and filtered.

This list is by no means complete in general astronomical data processing terms, but is intended to express only the types of operations which might take place during VO data access, where subsetting, filtering, projection, image extraction, etc., are all defined operations. Other values may be added in the future. The creation type is not intended to describe the processing done to produce the data collection itself, which the service is used to access.

2.5 Service Type

Not all SIA services are of the same type: services are further classified by their subtype, indicating how they generate the image returned by the service. The subtype of a SIA service is similar to the dataset creation type as described in section

2.4.2; usually the creation type and the SIA service subtype are the same, but this is not always the case. A simple service which returns only entire archival images is an "***archival***" SIA service. A service which can return a subregion of a larger image is a "***cutout***" service. A SIA service which can combine multiple input images is a "***mosaic***" service (a mosaic service could also do cutouts if presented with a sufficiently small ROI). A SIA service which dynamically generates an image from more fundamental data, e.g., visibility or event list, is an "***imageExtraction***" service.

[*An issue is whether to permit a given service to support more than one type of image generation. Can the same service support both archival and cutout access for example? These could easily be separate capabilities of the same service, but then we might need a way for the client to specify the type of output desired. Or, the service could describe all the data available and let the client decide what type of image to retrieve.*]

2.6 Data Cube Access

A special subtype of SIA service is one which deals *only* with 2D images. A SIA service which is capable of cube access **may** also support 2D image access (hence for example would be capable of logically associating a cube with any related 2D projections). Image cube support is an advanced capability and if provided is indicated in the service capabilities description.

3 Interface Overview (informative)

The basic form of a SIA service (or any other second generation DAL service) is specified in detail in section XX. In the current section we merely summarize the basic elements of a standard data service.

3.1 Request Format

In general a service may implement multiple ***operations***, such as queryData; altogether these define the ***interface*** to the service. Interfaces may change with time hence are versioned. It is possible for a given service implementation to simultaneously expose multiple interfaces or versions of interfaces.

The SIA interface described in this document is based on a distributed computing platform (DCP) comprising Internet hosts that support the Hypertext Transfer Protocol (HTTP). Thus, the online representation of each operation supported by a service is composed as a HTTP Uniform Resource Locator (URL).

A request URL is formed by concatenating a ***baseURL*** with zero or more operation-defined ***request parameters***. The baseURL defines the network address to which request messages are to be sent for a particular operation of a particular service instance on a particular server. Service operations generally share the same baseURL but this is not required.

3.1.1 Parameters

Parameters may appear in any order. If the same parameter appears multiple times in a request the operation is *undefined* (if alternate values for a parameter are desired the *range-list* syntax may be used instead). Parameter *names* are case-insensitive. Parameter *values* are case-sensitive unless defined otherwise in the description of an individual parameter.

All operations define the following standard parameters:

REQUEST	The request or operation name, e.g., "queryData" (mandatory).
VERSION	The version number of the interface (optional).

The values of both the REQUEST and VERSION parameters are case-insensitive.

A given service instance may support multiple versions of the SIA interface, which includes both the input parameters and the query response with all of its complex metadata. By default the service assumes the highest *standard* version which is implemented (access to any experimental versions supported by a service requires explicit specification of the version by the client). Explicit specification of the interface version assumed by the client is necessary to ensure against a runtime version mismatch, e.g., if the client caches the service endpoint but a newer version of the service is subsequently deployed. If desired the client can omit the VERSION parameter to disable runtime version checking, and default to the highest version standard interface implemented by the service.

All other request parameters are defined separately for each service operation.

3.1.2 Parameter Values

Integer numbers are represented as defined in the specification of integers in XML Schema Datatypes. Real numbers are represented as specified for double precision numbers in XML Schema Datatypes. Sexagesimal formatting is not permitted, either for parameter input or in output metadata, other than in ISO 8601 formatted time strings (sexagesimal format is fine for a user interface but inappropriate for a lower level machine interface, where it only complicates things).

SIA defines a special ***range-list*** format for specifying numerical ranges or lists of ranges as parameter values. For example, "`1E-7/3E-6;source`" could specify a spectral bandpass defined in the rest frame of the source. The syntax supports both open and closed ranges. Ranges or range lists are permitted only when explicitly indicated in the definition of an individual parameter. For a full description of range list syntax refer to section XX.

3.1.3 Error Response

In the case of a service-level error, service operations **must** return a VOTable containing an `INFO` element with name `QUERY_STATUS` and the value set to "`ERROR`". More fundamental service or protocol errors may result in an HTTP level error, hence a client program should be prepared to handle either response. A

null query, that is a queryData which does not find any data, is not considered an error. More information on error responses is given in section XX.

3.2 Synchronous and Asynchronous Requests

A service operation which executes *synchronously* normally does not return a response to the client until the operation completes (the exception is an operation which streams data back to the client). An *asynchronous* operation returns an immediate response to the client indicating whether or not the request was accepted, with the operation continuing to execute as a background job on the server. A service operation which executes asynchronously performs the same action as a synchronous request, but may take an arbitrarily long time to execute.

SIA uses the VO standard *Universal Worker Service* (UWS) pattern to manage asynchronous requests [XX]. This provides a standard mechanism for submitting and monitoring asynchronous requests regardless of the context (hence the mechanism is not specific to SIA).

In terms of the service interface, the primary difference between a synchronous and asynchronous request is the URL used to submit the request to the service. A synchronous request is indicated by appending "`/sync`" to the service *base-URL*; an asynchronous request is indicated by appending "`/async`". Synchronous requests may be submitted using either HTTP GET or POST. Asynchronous requests **must** be submitted using HTTP POST, as required by the HTTP protocol (this is required since an asynchronous request changes state on the server). In order to provide a consistent URL generation method, the request type suffix (/sync or /async) is required regardless of the service capabilities. The request parameters are the same in either case.

3.3 Interface Summary

SIA defines the following service operations:

queryData	Find and describe data matching the specified query constraints. An access reference, provided for each such data object, may be used to retrieve the described dataset, which may be computed on-demand by the service if virtual data is referenced. Support for synchronous execution is mandatory. The request may optionally execute asynchronously if supported by the service.
accessData	Directly access a specific dataset or data collection, computing and returning the specified virtual image. This may be used to repeatedly access portions of a large image or data collection. Used for precision data access. Support for synchronous execution is mandatory. The request may optionally execute asynchronously if supported by the service.

stageData	Request computation of one or more images, as described in a prior call to *queryData*. The request executes asynchronously. Images are *staged* to storage local to the service as they are computed, and may later be retrieved by the client application.
getCapabilities	Describe the capabilities of the individual service implementation. This information is cached in the Registry and may be used to find services with the required capabilities. Synchronous only.
getAvailability	Verify that the service is up and running normally, and ready to accept client requests. Synchronous only.

In a typical scenario the client would issue a *queryData* to find data of interest, followed by one or more synchronous HTTP GET operations to retrieve images referenced in the query response. If many images are to be requested or image generation is expected to be computationally intensive, a *stageData* request could be issued to cause the computation to run asynchronously; a sequence of synchronous GET operations could still be used to retrieve the generated images following completion of the data staging job. If more advanced image generation is required the *accessData* request would be used, for example to reproject an image or to perform repeated precision access to a large data cube.

In most cases where the same query is being posed to many services to broadly search for data for a particular analysis, *queryData* is preferable as it will find all data in a certain region of interest in parameter space regardless of the type of service or the detailed capabilities of each individual service. If a given service is capable of generating virtual data it may get closer to what is requested with a cutout or projection; otherwise archival images will be found which overlap the ROI or otherwise match the specified query constraints. For more precision analysis where the client wants to precisely specify how to generate an image *accessData* is preferred, although it may require more knowledge on behalf of the client of the characteristics and capabilities of the specific data collection, image, or service being accessed. An iterative approach is possible, using *queryData* to find data of interest, possibly followed by *accessData* for precision access to specific datasets. Combinations are also possible, e.g., the access reference URL returned by a *queryData* could itself be a (hidden, to the client) call to the service *accessData* operation. Any such access reference URL could be used as input to a subsequent *stageData* request if asynchronous image generation is desirable.

These service operations are described in more detail in section XX below.

3.4 Request Examples

Some examples of basic usage of SIA follow.

Invoke a synchronous *queryData* to find data in a given region on the sky:

$baseURL/sync?REQUEST=queryData&POS=180.0&SIZE=0.2

Invoke an asynchronous *accessData* to cutout a subcube of a spectral data cube (here PUBDID would identify the specific cube to be accessed):

```
curl -d "REQUEST=accessData&PUBDID=XX" \
  -d "POS=180.0&SIZE=0.2&BAND=10E-3/12E-3" $baseURL/async
```

In this example the commonly available *curl* application (*wget* or a browser could also be used) is used to issue a HTTP POST request to the remote UWS-based job manager integrated into the SIA service. The query may run for an arbitrarily long time; during execution the standard UWS facilities may be used to monitor the job status. When the job completes the generated image can be retrieved with a conventional synchronous HTTP GET.

4 Requirements for Compliance

The keywords "**must**", "**required**", "**should**", and "**may**" as used in this document are to be interpreted as described in the W3C specifications (IETF RFC 2119).

4.1 Mandatory Capabilities

In order to be ***minimally compliant*** a service is required to implement all elements of the SIA protocol identified as **must** in this document. In brief, the minimal service implementation must include the following:

1. The service **must** implement the *queryData* method, providing synchronous return of the query reponse encoded as a VOTable document. At least the POS, SIZE, BAND, TIME, and FORMAT query parameters **must** be supported by the service (regardless of whether these are valid for the data being accessed). The query response **must** include all metadata fields identified as mandatory in the protocol.
2. The service **must** support synchronous retrieval via the access reference URL of image data referenced in the query response returned by the *queryData* operation. An exception is where data is described which is not available for retrieval, in which case the access reference may have a null value (for example a dataset which has been observed but which is not available online).
3. The *getCapabilities* and *getAvailability* operations **must** be provided to allow a client (or the resource registry) to query the capabilities of the service, and to provide a means to monitor service availability and health.

4.2 Advanced Capabilities

A service **may** implement the *accessData* request. A service **may** implement the *stageData* request.

A service **may** provide multidimensional image (image cube) support. Many services will provide only access to 2D images.

Many services serve static archival image files or provide simple cutouts, in which case asynchronous capabilities are not required as requests are unlikely to time out. A service which is likely to time out while servicing a request (e.g., be-

cause generation of virtual data is computationally expensive) **should** provide asynchronous capabilities via the *stageData* operation. If a service provides any asychronous capabilities it **may** provide asychronous execution of all provided service operations with the exception of *getCapabilities* and *getAvailability.*

A service is said to be ***fully compliant*** if, in addition to the functionality required to be minimally compliant, the service implements all the **should** elements of the interface defined herein.

A more advanced service may in addition implement other advanced capabilities identified as **may** in this document.

5 Service Operations

5.1 QueryData

The purpose of the *queryData* operation is to find images (physical or virtual) which satisfy the specified query constraints. The result is returned encoded as a VOTable document wherein each row of the table describes one candidate dataset. Referenced datasets may be directly downloaded via the given access reference URL, or passed on to the *stageData* operation for computation as an asynchronous operation.

5.1.1 Use of Parameters as Query Constraints

A data discovery query uses parameters as query constraints to find only data of interest to the client.

Unless otherwise specified, if the service does not support a query parameter defined by the protocol it **must** permit the parameter to be present in the query without error, even if the parameter is not actually used as a query constraint by the service. Most parameters are used as query constraints, to narrow the selection of data by the service. If a given parameter is not specified in the query or is not supported by the service, or cannot be applied to the data as the necessary dataset metadata is not available (note this is different than the case of theory data described below), then a logical value of “all” should be assumed, meaning that the parameter is not used to constrain the query. This allows a query to succeed even if it includes parameters which the service does not support, so that the same query can be submitted to multiple service instances. Since queries can be imprecise due to missing metadata it is up to the client to analyze the query response to further refine the query.

If a service supports a parameter but the value given cannot be parsed or is otherwise illegal (as opposed to merely not matching any data) then an error response should be returned to the client. If a service does not support a parameter it is not required to parse the parameter value and report errors, i.e., it may ignore the unsupported parameter.

If a service is required to support certain input parameters, that means that the service **should** (if possible) use such a parameter to constrain a query. If this is

not done and the service merely ignores a mandatory input parameter which the service is required to support, then it may be easy for the client to pose a query which results in an overflow of the query response or some other error condition. For example, if a client queries for data based only on the spectral bandpass and the service does not support the BAND parameter, the query may overflow or be declared invalid even though valid data is available.

Specific parameters may or may not have meaningful values for a given data collection. For example, for theory data, anything having to do with time or position on the sky may be undefined. For solar or planetary data, time is defined but the spatial position on the celestial sphere may be undefined or not meaningful. In such a case, where a specific value is specified for an attribute which is undefined or has no meaning for a given data collection, the service should respond by finding no matching data (for example a query based on POS, if cast to a broad range of services, would probably not find any matching data if posed against a service providing access to theory data). For data collections where all physical measurement parameters are meaningful, for example spectra of galactic or extragalactic astronomical targets, all parameters should be supported and used to constrain the query, even if only imprecise values of the parameters are known for a given dataset.

5.1.2 Mandatory Query Parameters

The following parameters **must** be implemented by a compliant service:

Parameter	*Sample value*	*Physical unit*	*Datatype*
POS	52,-27.8	degrees; defaults to ICRS	string
SIZE	0.05	degrees	double
BAND	2.7E-7/0.13	meters	string
TIME	1998-05-21/1999	ISO 8601 UTC	string
POL	any	-	string
FORMAT	fits	-	string

All services **must** support queries containing at least these six parameters, representing coverage in the fundamental physical measurement axes, and the output data format or formats desired by the client. Although services must support these parameters, this does not necessarily mean that the quantity referred to is meaningful for the class of data being queried (XX). While a compliant service must implement these parameters and use them where possible to constrain queries, a valid query can be composed from any combination of parameters, and may include or omit any given parameter. If a parameter is not specified, it is not used to constrain the query. For example if POS is not specified, data from any spatial region, or data for which POS is undefined, will satisfy the query and other parameters must be used to constrain the query.

5.1.2.1 POS

The central position of the region of interest (ROI). The coordinate values are specified in list format (comma separated) with no embedded white space, as defined in section XX.

Example: `POS=52,-27.8`

POS defaults to right-ascension and declination in decimal degrees in the ICRS coordinate system. A coordinate system reference frame **may** optionally be specified to specify a coordinate system other than ICRS. When appropriate for the data being accessed the service **must** support ICRS and **should** support galactic and ecliptic coordinates as well (notable exceptions being services for theory data and solar and planetary data). The reference frame is specified as a list format modifier, with the acceptable values as defined in the respective table of the CoordSys object in the Spectrum data model (McDowell/Tody et al. 2007), which is in turned based upon the spatial coordinate frames defined by Table 3 (standard reference frames) in STC (Rots 2007).

Example: `POS=52,-27.8;GALACTIC`

Coordinates requiring more than two values are possible merely by having more than two comma-delimited values before the qualifier.

Whether or not a service supports a specific coordinate systems for POS is a service-defined optional capability. It is an error if a coordinate reference frame is specified which the service does not support.

[*POS,SIZE can also be used to specify the input for multi-position queries, similar to what is specifed for TAP PQL. This still needs to be added to the specification for SIAV2.]*

5.1.2.2 SIZE

[*This section has been revised from SIAV1 to attempt to more clearly describe the use of the CAR projection for large area searches. Of course the ROI if based upon CAR still blows up at the pole. But large area searches are much simpler using CAR and the distinction is negligible for small regions, and regions which extend over the pole are rare or nonexistent in practice. The alternative would be to specify SIZE in angular units. This is ok at a fixed declination and more natural for small regions but results in a complex N-S boundary for large regions - CAR seems simpler for large regions. Which is the simplest approach? A related issue is whether to support rotation of the ROI, but this is rarely required and is already provided by REGION and by WCS-based image generating using accessData. Whatever we do this needs to be clearly spelled out as this is one of the major points of confusion with SIAV1.]*

The width and height of the rectangular region of interest, defining either the *ideal image* footprint (for services which generate images on the fly, e.g., a cutout service) or the *search region* (for services which find static archival images intersecting the specified region). Values are specified in decimal degrees in the coordinate frame defined for POS, e.g., for ICRS the values would be the width in RA and height in DEC, both specifed in decimal degrees.

Example: `SIZE=0.05[,0.03]`

If only a single value is given it applies to both the width and height of the search region, otherwise the two values may be specified separately.

In the case of a service returning *archival* (whole) images, POS,SIZE specifies the search region to be used to search for images. In the case of a cutout or mosaic service which computes images on the fly, POS,SIZE specifes the footprint of the "ideal image" to be generated. Thus a single POS,SIZE can be used to search for data regardless of the type of service being queried. If only discovery of existing archival images overlapping some search region is required the REGION parameter provides an alternate and more general mechanism capable of dealing with arbitrary regions.

A special case is when SIZE=0 or is unspecified. For an archival image service which returns whole images this tests whether the given point is in any image. For a cutout or mosaic service this will cause a default sized image to be returned. The default image size is service-defined and may be a value considered appropriate for the service, for the given image or data collection being accessed, or for the object (if any) at the given position.

POS and SIZE define a nonrotated, rectangular region in the specified coordinate system. For example in the case of the ICRS coordinate frame, coordinates are specified in right ascension and declination, using the cartesian (CAR) projection with the region center (POS) as the reference point. The cartesian projection is used here as it is simple and can scale easily to large regions or to the whole sky, yet works about as well as anything else for small regions. If more precise image searching is desired the REGION parameter can be used to more rigorously specify a generalized search region. If more precise image generation is desired then *accessData* can be used. Note that the use of the CAR projection to define the ROI has nothing to do with what projection we choose for any actual generated images.

Techical Note:

Large query regions are well defined for the CAR projection used for the ROI. For example, a region with a DEC size of 20 and a RA size of 360, with POS anywhere on the equator, defines a region 20 degrees wide covering the entire celestial sphere at the equator. A region with a DEC size of 90 and a RA size of 180 at 45 degrees N would cover one half of the northern hemisphere. Regions which extend past the pole are defined: they extend past and over the pole in declination. The ROI is undefined if a region extends over the pole since the CAR projection blows up at the pole, but can be approximated by assuming a RA size of 360.

Note that since SIZE is specified in terms of the coordinate frame used for POS, the extent of the ROI can be computed by taking the central position coordinates

and simply adding or subtracting half the value of SIZE. In the case of celestial coordinates since computations are done using the CAR (cartesian) projection, angular measures will differ from coordinate measures by a cosine(dec) factor as one approaches the pole (e.g, the angular extent of the ROI in RA decreases by a cosine(dec) factor as the region approaches the pole).

It is suggested that if only a single value is given for SIZE that this be considered as applying to the ROI size in declination (hence it is an angular extent), and a cosine(dec) factor should be applied in computing the extent of the search region in right ascension. The ROI in this case is thus automatically corrected for declination or latitude as the region approaches the pole. If the client supplies two values for SIZE it is responsible for applying this correction if desired. Since query refinement can be performed on the client side it is permitted to use approximation techniques for the search so long as the returned data is accurately described and no data is missed. Computations for other coordinate frames should follow a similar approach.

5.1.2.3 REGION (optional)

In the case of services which return archival images (whole images) the REGION parameter may be used to specify the spatial region to be searched more precisely than can be done with POS,SIZE. The region is specified as a STC-S formatted string [XX]. When REGION is used the search region is no longer limited to nonrotated rectangles, e.g., circles, ellipses, rotated regions, and arbitrary polygons may be specified as well.

Example: `Circle ICRS 148.9 69.1 2.0`

A service **may** implement the REGION parameter. A service which implements REGION **should** support at least the ICRS and GALACTIC coordinate frames, if appropriate for the data being accessed. It is an error if the REGION parameter is specified but is not supported by the service.

5.1.2.4 INTERSECT (optional)

A parameter that indicates how matched images should intersect the region of interest. The allowed values are:

* COVERS -- The candidate image covers or includes the entire ROI.
* ENCLOSED -- The candidate image is entirely enclosed by the ROI.
* CENTER -- The candidate image overlaps the center of the ROI.
* OVERLAPS -- The candidate image overlaps some part of the ROI.

INTERSECT applies to regions specified by both POS,SIZE (in region search mode) as well as REGION.

If this parameter is not present, INTERSECT=OVERLAPS is assumed. Calculations need not be exact, e.g., a nonrotated bounding box approximation may be used to compute the type of intersection of a candidate image with the ROI. If the

client requires a more precise measure, the spatial intersection a target image with the ROI may be computed precisely using the WCS metadata returned in the output VOtable. For a cutout or mosaicing service this parameter refers to the portion of the generated image containing valid (non-blank) data.

[*Do we want to keep this parameter? I am not aware of any cases where anyone has actually implemented this for SIAV1*; *it is possible but probably rare*]

5.1.2.5 BAND

The spectral bandpass is specified in *range-list* format (XX) either numerically as a wavelength value or range, or textually as a spectral bandpass identifier, e.g., a filter or instrumental bandpass name. If a numerical value is specified as a single value it matches any spectrum for which the spectral coverage includes the specified value. If a two valued range is given, a dataset matches if any portion of it overlaps the given spectral region. The range list may contain multiple elements in which case a candidate dataset matches if it matches any element of the range list. See section XX for a more detailed discussion of range lists.

For a numerical bandpass the units are wavelength in vacuum in units of meters (Hanisch *et.al*, 2005) in the rest frame of the source.

If a bandpass is specified as a string identifier it is assumed to be a bandpass identifier such as a standard VO bandpass name (“uv”, “radio”, “optical”, etc.) as specified in the resource metadata [RSM, Hanisch *et.al*, 2005] for Coverage.Spectral.Bandpass.

5.1.2.6 TIME

The temporal coverage (epoch of observation) is specified as a single value or range in ISO8601 format. If the time system used is not specified UTC is assumed. The value specified may be a single value or an open or closed range. If a single value is specified it matches any spectrum for which the time coverage includes the specified value. If a two valued range is given, a dataset matches if any portion of it overlaps the given temporal region.

[*Is ISO time all we need here? MJD could be offered as well, using a qualifier to specify the time system used.*]

5.1.2.7 POL

Specifies whether or not data is desired which measures polarization, and if so the type of polarization desired. Possible values include “any”, “none”, “stokes”, “linear”, “circular”, etc.

[*This needs more careful thought to specify the possible values. For the most part here we are concerned with data discovery, e.g., whether or not polarization is measured at all, or whether specific types of polarization are measured such as linear or circular. Extracting specific polarization planes is more of an issue for accessData.*]

5.1.2.8 FORMAT

The FORMAT parameter specifies the allowable data formats for a retrieved image. The value is a comma-delimited list as defined in section XX, where each element can be any recognized MIME-type such as

```
application/fits, image/jpeg, text/html
```

and so forth.

In addition to the standard MIME-type format specifications defined above, the following special shorthand values are defined:

FORMAT	Meaning
all	All formats supported by the service
fits	Shorthand for `image/fits` or `application/fits`
graphic	Any of the graphics formats: JPEG, PNG, GIF
html	The image is rendered as a HTML page with annotations

If FORMAT is omitted, FORMAT=ALL should be assumed, and the service should describe all available formats. FORMAT values are case insensitive.

[*Additional predefined FORMATs can be considered as well. For example, "text/xml" (as for SSA) for a richer alternative to FITS for returning full image metadata.*]

The FORMAT parameter describes the desired format of returned image data. If no data is available in the specified format, a null query response should be returned indicating that no data satisfying the query is available. If data is dynamically generated the service may generate data in the format requested by the client on the fly. Note FORMAT applies only to the data; the query response itself is always returned as a VOTable.

5.1.3 Recommended and Optional Query Parameters

The following additional parameters **should** or **may** be implemented by a service; all the "recommended" parameters are required for a fully compliant service. In the table below and those following, mandatory parameters are indicated by MAN, recommended parameters by REC, and optional parameters by OPT.

Parameter	Sample value	Unit	Req	Datatype
SPECRES/RP	2000	λnone	REC	double
SPATRES	0.05	degrees	REC	double
TIMERES	31536000 (=1yr)	seconds	OPT	double
FLUXLIMIT	?	?	OPT	double
TARGETNAME	mars		OPT	string
TARGETCLASS	star		OPT	string
ASTCALIB	absolute		OPT	string
FLUXCALIB	relative		OPT	string
WAVECALIB	absolute		OPT	string
PUBDID	ADS/col#R5983		REC	string

CREATORDID	ivo://auth/col#R1234		REC	string
COLLECTION	SDSS-DR5		REC	string
TOP	20	none	REC	int
MAXREC	5000		REC	string
MTIME	2005-01-01/2006-01-01	ISO 8601	REC	string
COMPRESS	hcompress		REC	string
RUNID			REC	string

The spatial, spectral, and time resolution of the data may all be used as query constraints to find data of interest. The flux limit specifies the minimum sensitivity of data required for analysis. The target name and class may be used to search for data for a specific target, or for a specific type of astronomical object. The creator and publisher dataset identifiers and data collection name may be used to precisely specify the data to be accessed. TOP and MAXREC are used to manage the data to be returned to the client. All parameters are explained in more detail below.

[*We may also want a parameter to enable/disable optional metadata extension metadata, used to provide more detailed metadata such as individual image footprints, detailed provenance information, instrument-specific information, and the like.*]

5.1.3.1 SPECRES/RP

The minimum spectral resolution, specified as [*either*] the minimum spectral resolution specified as a wavelength in meters [*or as*] the spectral resolving power $\lambda\lambda/d\lambda$ in dimensionless units.

[*Which or both to support needs further discussion; one possibility would be to permit use of a ";type" qualifier to specify either resolution or resolving power.*].

5.1.3.2 SPATRES

The minimum spatial resolution specified in decimal degrees. Spatial resolution refers to the PSF of the observed signal and is independent of the pixel size of the image so long as the image is adequately sampled.

5.1.3.3 TIMERES

The minimum time resolution, specified in seconds. For a typical image the time resolution corresponds to the bounds of the time coverage of the exposure. For a time cube the time resolution would refer to the time resolution along the time axis of the image.

5.1.3.4 FLUXLIMIT

The minimum measureable flux in the image, specified in Jansky per unit area.

[*What shall we use for the unit? Does anyone know how to compute this for optical data where a magnitude limit would normally be used?.*]

5.1.3.5 TARGETNAME

The target name, suitable for input to a name resolver. In general it is preferable to perform target name resolution on the client side, using POS to drive the query performed by the service, so that any service can respond to the query. The main reason that TARGETNAME is included at the service level is to make it possible to find images of objects that do not have a known position, for example, images of solar system planets or asteroids. If both TARGETNAME and POS are specified, both must satisfy the query for a candidate object to be matched.

5.1.3.6 TARGETCLASS

A comma delimited list of strings denoting the types of astronomical objects to be searched for.

Examples: `star, galaxy, pulsar, PN, AGN, QSO, GRB`

[*We need to add a reference here to the astronomical object classification specified by the IVOA Semantics WG.*

5.1.3.7 ASTCALIB

Specifies the minimum level of astrometric (spatial) calibration for acceptable data. Possible values are "`absolute`", "`relative`", and "`any`" (the default). If "relative" is specified, data which have an absolute astrometric calibration will be found as well.

5.1.3.8 WAVECALIB

Specifies the minimum level of spectral coordinate calibration for acceptable data. Possible values are "`absolute`", "`relative`", and "`any`" (the default). If "relative" is specified, spectra which have an absolute spectral coordinate calibration will be found as well.

[*Note there is no TIMECALIB - it doesn't seem to be needed as pretty much all data is time calibrated; all we need is probably resolution and error.*]

5.1.3.9 FLUXCALIB

Specifies the minimum level of flux calibration for acceptable data. Possible values are "`absolute`", "`relative`", "`normalized`", and "`any`" (the default). If "relative" is specified, spectra which have an absolute flux calibration will be found as well. "Normalized" refers to spectra which have been normalized by dividing by a reference spectrum (including continuum normalization).

5.1.3.10 PUBDID

The IVOA publisher's dataset identifier, assigned by the publisher of a dataset. PUBDID will uniquely identify a dataset within the collection managed by the publisher, however the same dataset published in different places may have a different PUBDID assigned by each publisher. This differs from CREATORDID which is unique to the data, however there is no guarantee that data creators will assign

IVOA identifiers to created datasets. A data publisher can always assign a unique PUBDID when a dataset is published to the VO. ADS dataset identifiers are an example of a PUBDID, but in general any publisher may assign their own unique publisher dataset identifier. Publisher dataset identifiers may be determined by a prior query or some external means, such as another form of archive query.

Note:
A special case of a publisher's dataset identifier is the **ADS dataset identifier,** used to reference published IVOA datasets in journal articles.

5.1.3.11 CREATORDID

An IVOA dataset identifier, assigned at creation time by the creator of the parent data collection (survey project, observatory, etc.). Datasets may have a globally unique CreatorDID assigned prior to publication of the data to the VO, for example when the data is generated by a processing pipeline, or ingested into the master archive for the data collection. This is possible since the Creator entity for a data collection (e.g., an observatory or survey project) controls its own namespace, which can be registered as a globally unique Authority identifier. When a CreatorDID has been assigned this is the most universal way to refer to a dataset, as all replicated versions will share the same CreatorDID regardless of where they are published. Creator dataset identifiers may be determined by a prior query or by some other means, such as another form of archive query.

Example: `ivo://nrao.edu/vla#1998s2/4992a`

5.1.3.12 COLLECTION

Either the IVOA resource identifier or the “shortName” of a data collection as defined by the service, for example `SDSS-DR6`, or `ivo://nrao.edu/vla`. By data collection we refer to an organized, uniform collection of datasets from a single source, for example a single data release from a survey, or an instrumental data collection from an observatory. Unless an IVOA identifier is input, the service should treat the search term as a case insensitive, minimum match string. For instance, “dss” would match either `dss1` or `ESO-DSS2`. Allowable data collection references are specified in the service capabilities.

5.1.3.13 TOP

TOP limits the number of returned records in the query response table to the specified number of top ranked ones. Records are ranked according to a “score” heuristic (Dolensky 2006). The details of the actual heuristic used are up to the service, but the general idea is that the better a candidate dataset matches the query, the higher the score it receives. Metrics such as distance from the specified position, or the degree of overlap with a specified bandpass or time interval, determine the score. If two datasets would otherwise have the same score, the service may use other unspecified dataset characteristics, such as some intrinsic

data quality metric, to further rank candidate datasets. If the service implements a ranking heuristic the query response table should normally be returned sorted in order of decreasing score. TOP can also be used by the client to limit the size of the query response table in cases where the query might find a very large number of candidate objects.

5.1.3.14 MAXREC

The maximum number of records to be returned in the query response. This may be used by the client to increase or decrease the built-in default limit defined by the service, up to some maximum service-specified default. A service should typically have a modest default MAXREC, providing a reasonable query response time, and a large upper limit on MAXREC, provided to enable large queries. Very large values of MAXREC may allow streaming an arbitrary amount of data back to the client.

If a query response exceeds the value of MAXREC currently in effect, MAXREC rows of data should be returned to the client, setting the query status value to OVERFLOW to indicate that overflow occurred. It is not an error if query overflow occurs.

5.1.3.15 MTIME

Find only datasets modified, created, or deleted in the given range of dates, specified as a single element in range-list format, as an open or closed range, with the dates specified in ISO 8601 format. Note this is not the same thing as TIME, which refers to time of observation. MTIME may be used to periodically query services for new or updated data. Deleted datasets are indicated by a non-null deletion date in the Dataset.Deleted field of the query response. Services which support MTIME should also support Dataset.Deleted (see also XX).

5.1.3.16 COMPRESS

If this flag is present, datasets returned via the *getData* method **may** optionally be returned to the client in compressed form. Valid values are “gzip”, “hcompress”, and “rice”. By default compressed data is not permitted.

Dataset-level compression is distinguished from protocol-level compression, which is performed at the level of the HTTP protocol, on the entire data stream, and is transparent to the client.

[*Note: We may need an additional parameter to specify the level of compression. Lossy compression should also be supported.*]

5.1.3.17 RUNID

The RUNID is an opaque string used to associate multiple service invocations in service logs, e.g., to identify them as all belonging to the same job or application. RUNID is not used by SIA itself, except in cases where SIA may call another VO service, in which case the RUNID parameter **should** be passed on to the called

service. The purpose of RUNID is to allow the job run ID to be logged, and in particular, if a job involves multiple requests to multiple services, to allow all just requests to be associated by having a common RUNID. This applies to all service operations regardless of whether they execute synchronously or asynchronously. [*An exception might be getAvailability*).

5.1.4 Service-Defined Parameters

The service **may** support additional service-defined parameters. Parameter names must not match any of the reserved parameter names defined herein, independent of case.

Any service defined parameters should be defined in the metadata query response (). Appendix A presents an example of this, where service defined parameters are used to dynamically generate spectra based upon a theoretical model.

5.2 Query Response

[*This section has not yet been fully integrated; in particular the Mapping model has evolved considerably (see text file notes).*]

The output returned by a query is an XML document compliant with VOTable V1.2 or greater (VOTable 2009) and should be returned with a base MIME-type of `text/xml` to enable simple display of query results in browsers using direct rendering of the XML, or an optional style sheet. Parameterization may be used to further refine the MIME-type, for example "`text/xml;content=x-votable`" may be used to indicate that the content of the XML document returned is a VOTable.

Note:
The `FORMAT` parameter has no influence on the query response. `FORMAT` applies only to the returned datasets, not to the query response. The query response is always returned as a VOTable.

The VOTable **must** contain a `RESOURCE` element, identified with the tag `type = "results"`, containing a single `TABLE` element with the results of the query. Additional `RESOURCE` elements may be present, and the usage of any such elements is defined below (extensions).

The `RESOURCE` element **must** contain an `INFO` with `name="QUERY_STATUS"`. Its value attribute should be set to `"OK"` if the query executed successfully, regardless of whether any matching data were found. All other possible values for the value attribute are described below (section ???).

Examples:

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

Another `INFO` with `name="SERVICE_PROTOCOL"` should contain the protocol version number in its value attribute and the name of the service protocol as the fixed string `"SIAV2"` (see version negotiation…).

Example:

```
<INFO name="SERVICE_PROTOCOL" value="2.0">SIAV2</INFO>
```

Additional INFOs may be provided, e.g., to echo the input parameters back to the client in the query response (a useful feature for debugging or to self-document the query response), however this is not required.

In the query response table each row represents a different physical or virtual dataset which is potentially available to the client. The VOTable `GROUP` construct is used to associate related groups of fields. Table FIELDs describe the attributes of each dataset; if all datasets share the same value for an attribute it can be represented as a PARAM.

Hint:

Put constant values in `PARAM` elements instead of repeating them in each table row.

5.2.1 Query Response Metadata

Names of fields and parameters are left to the service provider. UTYPEs of standard fields are required for identification of interface elements and **must** be given and **must** comply with the SIAV2 protocol (this document) SIAV2 utypes are a subset of the Obs data model, considered as required for images and cubes. UCDs **should** also be given when specified by the protocol (not all interface or data model elements have assigned UCDs), but are not used to identify interface or data model elements. Values for the UCDs of standard interface and data model elements, where defined, are given in this specification.

Note:

UTYPE values **must** be provided to identify interface or data model elements.
UCD values for standard data model elements **should** be provided as well.

The SIAV2 query response consists of a number of fields, identified by UTYPE, grouped into component data models (or packages) of the form "*<component-name>*'/'*<field-name>*'.'*<field-name>*", as defined in the utype specification document (Louys et al, 2009). The components of the query response are described directly by the SIAV2 protocol (this document), but they are compliant with the Observation data model (xxx et al, 2009). In most cases the UTYPE values are

close to the one defined for the SSA query response (Tody et al , 2008). Hence most of the query response metadata consists of generic component data models. For example, if the Observation data model specifies obs:Target.Name this appears in the SIAV2 query response as sia2:Target.Name exactly like in SSA (apart from the namespace). Applications can refer to Target.Name regardless of whether the data to be accessed is an Image or some other data object such as a Spectrum or a Cube.

In the following, query response parameters which are mandatory, recommended, or optional are indicated as such in the tables or specified more precisely in the text. Additional attributes from the Observation data model not shown here may appear in the query response table.

When a generic data model is applied in a specific context, the requirements for what is required, what is optional, and flexibility in what is permitted will vary depending upon how the data model is being used. Hence when data model attributes are indicated as **mandatory or recommended** in this document, this overrides any similar requirements specified in the Observation data model document. The SIAV2 query response is also more restrictive than the underlying model; in particular the **allowable units** are more restrictive than what is permitted in the model, in order to be more consistent with other elements of SIAV2, and to provide more uniformity to make multiband data discovery by the client easier. Hence within SIAV2, characterization restricts the allowable units for spatial coordinates to decimal degrees.
It is difficult to specify every detail of every metadata element in this document without burdening the text with too much detail; furthermore, many optional metadata values are omitted from the summary tables shown here. Full details are given in the Observation data model document, and in a convenient summary form in a spreadsheet which lists all metadata elements with full details for each.

Query metadata may be mapped to VOTable fields in any order, so long as fields which are part of the GROUP construct (all the component data models are GROUP elements) appear in consecutive table fields.

5.2.2 Types of Metadata

Metadata in the query response is grouped into a number of component data models as summarized in the table below, and explained in more detail in the sections which follow.

Service Metadata	
Query	Describes the query itself
Association	Logical associations
Access	Dataset access-related metadata
Data Model Metadata	

Dataset	General dataset metadata
DataID	Dataset identification (creation)
Provenance	Instrumentalm or software Provenance
Curation	Publisher metadata
Target	Observed target, if any
CoordSys	Coordinate system frames
Char	Dataset characterization
Mapping	Dataset Axes Mapping or WCS
Characterization Metadata	
Char/FluxAxis	Observable, normally a flux measurement
Char/SpectralAxis	Spectral measurement axis, e.g., wavelength
Char/TimeAxis	Temporal measurement axis
Char/SpatialAxis	Spatial measurement axis
Char/Polarization	Polarization Axis
Char/*.Coverage	Coverage in any axis
Char/*.Resolution	Resolution on any axis
Char/*.SamplingPrecision	Sampling or Precision on any axis
Char/*.Accuracy	Accuracy and error in any axis
Mapping metadata	
Image matrix mapping	
WCS Mapping	

Service metadata is specific to the functioning of the service itself, for example to step through large queries or retrieve selected datasets. **Data model metadata** describes each dataset, and is common between the SIAV2 protocol and the Observation data model. **Characterization metadata** physically characterizes the dataset in terms of the spatial, spectral, and temporal measurement axes and the observable. Characterization is part of the data model but is broken out separately in the table above to show the major elements of the characterization model. Most of the metadata returned by SIAV2 is generic dataset metadata, which means it is not actually specific to images and cubes and may be used in other DAL interfaces to describe other types of dataset, for example a catalog. It is obvious that SIAV2 responses is very close to the SSA one. For data model metadata, please refer to the Observation data model for details unless specified otherwise in this document.

Each of these types of query response metadata is discussed in more detail in the sections which follow.

5.2.3 Query Metadata

Query metadata describes the query itself.

UTYPE	Description	Req
Query.Score	Degree of match to query params	REC

5.2.3.1 Query.Score

A record with a higher score more closely matches the query parameters. The score is expressed as a floating point number with an arbitrary scale (different queries may return results with different scale factors and cannot be inter-compared). If scoring is used, the query response table should be returned sorted in order of decreasing values of score, with the top-scoring items at the top of the list. The details of the heuristic used to compute the score are left to the service. See the discussion of the `TOP` parameter in section …..

5.2.4 Association Metadata

Association metadata is used to describe logical associations relating datasets described in the query response, as described in section … . Logical associations between query response records may refer to the data access operation itself, e.g., where the same data object is available in multiple output formats, or to logical associations relating the physical data, e.g., where multiple primary datasets are part of the same observation. The same dataset may belong to multiple associations.

UTYPE	Description	Req
Association.Type	Type of association	OPT
Association.ID	Unique ID identifying the association instance	OPT
Association.Key	Unique key different for each element of association	OPT

Each such association is described by a separate instance of the Association model, with a defined Association Type, ID, and Key. In many cases the Association Type and Key can be represented as fixed PARAMs, leaving only Association.ID to be represented as a FIELD in each table row.

In general, specification of the allowable Association types is beyond the scope of this specification. The semantic details of Associations are intended to be defined either at a lower level, for a specific data collection or service, or at a higher level, e.g., to describe complex data associations. An exception is the MultiFormat association described in the next section.

5.2.4.1 MultiFormat Association

A pre-defined case is the *MultiFormat* association, where several records refer to the same dataset which is available in several different output data formats. In this case Association.Type should be set to "`MultiFormat`", Association.ID can be anything, and Association.Key should be set to "`@Access.Format`" to indicate that the key which differentiates the elements of the association is the value

of the Access.Format field of the record. If several query response records are of this type the association **should** be specified to indicate the association. In all other cases (currently undefined by the protocol) the association **may** be specified.

5.2.4.2 Association.Type

A service-defined type used to indicate what type of association is being referred to. The value should be unique within the scope of the query response. There can be many types of logical associations. Associations provide a means of describing complex data aggregations relating multiple datasets (images in the case of SIAV2). Association is a type of extension mechanism, and the definition of associations is beyond the scope of SIAV2; SIAV2 like SSA merely provides the means to define and manipulate associations. Examples of possible associations might be a multi CCD detector observation consisting of as many images as CCD, each of which appears in the query response as an individual image, or a group of query response records which all refer to the same dataset but differ only in the output format.

Since the association type may be shared by many table records, it may be best specified as a PARAM in the output VOTable, using an ID-REF to link it to the association it refers to. An association type **should** be provided for each association in the table.

5.2.4.3 Association.ID

The association ID is a string, unique within the scope of a given VOTable, identifying one instance of a given association. All members of the association instance share the same Association.ID. The association ID **must** be provided for any association. The content of the string is up to the service. Multiple association IDs may be provided for a single record if a record belongs to more than one association. Note that Association.ID is unrelated to the VOTable ID, which is used to uniquely identify the elements of a VOTable.

Extension metadata may optionally be provided to describe an association in more detail. Extension metadata appears in the output VOTable as optional additional RESOURCE elements (see section ….). The ID-REF mechanism may be used to link such an extension record to the association in the main table. The contents of an association metadata extension record are externally defined and beyond the scope of SIAV2.

5.2.4.4 Association.Key

The association key **should** be provided to identify what is “different” for each member of an association. The value is a string and may be either an arbitrary value defined by the association, or a reference to one or more table fields which

form the association key. If a table field is referenced the '@' character should be prefixed to the VOTable ID of the referenced FIELD to indicate the indirection (e.g., "`@Format`"), otherwise the literal string is used as the key. A key may contain multiple elements delimited by commas.

5.2.5 Access Metadata

Access metadata is required to tell a client how to access the datasets described in the SSA query response.

UTYPE	Description	Req
Access.Reference	URI (URL) or template used to access the dataset	MAN
Access.Format	MIME type of dataset	MAN
Access.Size	Estimated (not actual) dataset size	REC
Access.Parameters.*		OPT

5.2.5.1 Access Reference

The simplets case of access reference is a URI (typically a URL) which can be used to synchronously retrieve the specific dataset described in a row of the query table response. If the dataset pointed to by the access reference does not exist at query time, it will be computed on the fly when accessed.
Access Reference can also be an URI template if some of the PARAMETERS can be filled interactively by the client during the AccessData phase. Access.Parameters will help to do this.
SIAV2 supports data staging and asynchronous data access. Support for these functionalities is described below and is usefull to support generation of simulated or synthetic data, as well as very large images retrieval.

When the access reference is a URL, it is convenient to be able to input the access reference directly in a Web browser or other standard Web tool to access the referenced dataset. For this reason the access reference string should be URL-encoded if it contains any reserved URL metacharacters (the "#" character used in dataset identifiers is particularly nasty). See also section… . The CDATA construct used in earlier data access interfaces (SIAP V1.0) does not serve the same purpose and should not be used; use URL encoding instead.

5.2.5.2 Output Format

The file format of a candidate dataset is specified by its MIME type. Both uncompressed and compressed data can be indicated in this fashion.

The file format says nothing about the data model used by whatever data object is stored in the file; this is specified by the Dataset.DataModel attribute discussed in section ……

A single data object may be available in multiple file formats. In such a case an association **should** be defined to indicate that the entries all refer to the same data object.

5.2.5.3 Dataset Size Estimate

The approximate estimated size of the dataset, specified in kilobytes, **should** be given to help the client estimate download times and storage requirements when generating execution plans. Only an approximate, order of magnitude value is required (a value rounded up to the nearest hundred KB would be sufficient). In the VO dataset sizes can vary by many orders of magnitude hence it is important to know this information to optimize execution plans before attempting to download data or request computation. It is preferable to return an order of magnitude estimate of the dataset size, than no value at all. A precise value is *not* required.

5.2.5.4 Access Parameters:

[*This should probably be handled by getCapabilties instead but is left in here for the present to record the type of functionality which we need to describe.]*

These parameters allow to describe detailed and specific access modes to the data:

- Access.param.Interactive gives the list of interactive parameters providing optionnaly possible ranges of values (eg POS, SIZE, COMPRESS, etc…)
- In order to give information to the client of where to find appropriate information in the retrieved file a couple of UTypes have been defined. Such specification is made necessary because sometimes the actual science data is only a subpart of the retrieved file :
- Acces.param.extnum and Access.param.extname give the Extension number and Extension name in FITS (or VOTABLE)
- Access.param.Cutout gives the cutout limit (à la “IRAF”) in Fits Array,
- Access.param.Field and acces.param.row give the name/number of the Field/row in FITS table or VOTABLE….)

5.2.6 Data Model Metadata

The following metadata components are in common with the Spectrum data model.

5.2.6.1 General Dataset Metadata

General dataset metadata describes the overall dataset.

UTYPE	*Description*	*Req*	*Default*
Dataset.DataModel	Datamodel name and version	MAN	Obs-1.0
Dataset.Type	Type of dataset	MAN	Image, cube

Dataset.Length	Number of pixels in image/cube	MAN	
Dataset.Deleted	Set to deletion time, if dataset is deleted	OPT	

Dataset.DataModel is a string identifying the data model type and version used in the described dataset. For SIAV2-compliant data this should be a value such as "Obs-1.0", as specified in the Obs data model document for the version of the data model being used. For pass-through of native project data some other value should be used which identifies the specific project data model used, e.g., "HST-STIS-1.0".

For the current SIAV2 interface, Dataset.Type is either "Image", or “Cube”. Dataset.Length is mandatory and specifies the dimensionless "length" of the image, i.e., the total number of pixels or samples in the full image. Dataset.Deleted is used with the MTIME query parameter to inform the client that a previously existing dataset has been deleted; if a service supports MTIME it should also support Dataset.Deleted. The value is the ISO 8601 date (as in MTIME) at which the dataset was deleted, or null for a normal non-deleted dataset. Dataset.Deleted should be returned in a query only if MTIME is used in the query, and the deletion date matches the interval of time specified by MTIME. Otherwise deleted datasets should never be visible in a query. A service may permanently delete dataset deletion history after a period of time (currently unspecified) long enough to permit clients to discover deleted datasets.

5.2.6.2 Dataset Identification and Provenance Metadata

Dataset identification metadata is used to describe the fundamental identify of a dataset, including where it came from and how it was created.

UTYPE	*Description*	*Req*	*Default*
DataID.Title	Dataset title	MAN	
DataID.Creator	Creator name (string)	REC	
DataID.Collection	IVOA Identifier of collection	REC	
DataID.DatasetID	IVOA Dataset ID	OPT	
DataID.CreatorDID	Creator assigned dataset identifier	REC	
DataID.Date	Data processing/creation date	OPT	
DataID.Version	Version of creator-produced dataset	OPT	
DataID.CreationType	Dataset creation type	REC	archival
Provenance.ObsConfig.Instrument	Instrument name	OPT	
DataID.ObsConfig.Bandpass	Bandpass name, e.g., filter	OPT	
DataID.ObsConfig.DataSource	Original source of data	REC	survey

Dataset.Title is a short, human-readable description of a dataset, and should be less than one line of text. Information such as the instrument or survey name, filter, target name, etc., is typically included in a condensed form. The exact contents of Dataset.Title are up to the data provider. Dataset.Creator identifies the entity which created the dataset, and should be a short string consistent with the RSM specification, e.g., "SDSS". Dataset.Collection is the registered IVOA identi-

fer of the data collection to which the dataset belongs, e.g., "`ivo://sdss/dr5/spec`".

The CreatorDID is the IVOA dataset identifier (if any) assigned by the entity which created the dataset *content*, typically (but not always) an observatory or survey project. If the dataset referred to is virtual data, CreatorDID refers to the parent dataset from which the virtual data will be created (see …. for further details). If a CreatorDID has been assigned to a dataset it **should** be provided, otherwise it should be omitted. DataID.Date, specified in ISO time format, specifies the date when the dataset was created or last modified by the DataID.Creator entity. If a dataset is modified or replaced without changing its CreatorDID, DataID.Date and DataID.Version should be updated accordingly. DataID.CreationType describes how the dataset returned by the service was or will be created, as defined in section … .

Provenance metadata are used to provide information on the scientific origin of the DataSet either on the observing or on the processing point of view.
Provenance.ObsConfig.Instrument is a short string identifying the instrument used to create the data (instrument may be an actual telescope instrument or something else, e.g., a program in the case of theory data). Provenance.ObsConfig.Bandpass is a short string specifying the bandpass name if any, e.g., a filter name or an instrumental bandpass such as I, J, K, Q, HI, and so forth. Values specified with Provenance.ObsConfig.Bandpass may be used as input to the BAND parameter (…..) to refine a query (if this feature is supported by the service).

Provenance.DataSource describes the original source of the data.

5.2.6.3 Curation Metadata

Curation metadata describes who curates the dataset and how it is published to the VO.

UTYPE	*Description*	*Req*	*Default*
Curation.Publisher	Publisher	MAN	
Curation.Reference	URL or Bibcode for documentation	REC	
Curation.PublisherDID	Publisher's ID for the dataset	REC	
Curation.Date	Date curated dataset last modified	OPT	
Curation.Version	Version of curated dataset	OPT	
Curation.Rights	Restrictions: public, proprietary, etc	OPT	public

Curation.Publisher is a short string identifying the publisher of the data, e.g., a data archive or data center, or an indexing service such as the ADS. Curation.PublisherDID is the IVOA dataset identifier (URI) assigned by the publisher to identify the dataset within its holdings. Curation.Reference is a forward link to publications which reference the dataset; multiple instances are permitted. Curation.Date and Curation.Version refer to the dataset *as curated by the publisher*,

hence can differ from the same values given in DataID, which refer to the *content* of the dataset as generated by the dataset Creator. Curation.Rights specifies whether the dataset is "public" or "proprietary". Proprietary data requires authentication and authorization by the data provider to access, and once downloaded should be protected from subsequent access on the client side.

Note:
If the same dataset is replicated at several locations with multiple publishers, it is possible to set up an association group to indicate this fact.

5.2.6.4 Astronomical Target Metadata

Target metadata describes the astronomical target observed, if any.

UTYPE	*Description*	*Req*	*Default*
Target.Name	Target name	OPT	
Target.Class	Target or object class	OPT	
Target.Redshift	Target redshift	OPT	
Target.VarAmpl	Target variability amplitude, typical	OPT	

Target.Name is a short string identifying the observed astronomical object, suitable for input to a name resolver. Target.Class is the object class if known, e.g., Star, Galaxy, AGN, QSO, and so forth (see section). Target.Redshift, Target.VarAmpl, are as defined in the data model. Either standard target values, or derived quantities, may be used in the query response.

5.2.6.5 Coordinate System Metadata

Coordinate system metadata describes the coordinate system reference frames used in the SIAV2 query response.

UTYPE	*Description*	*Req*	*Default*
CoordSys.SpaceFrame.Name	Spatial coordinate frame	REC	ICRS
CoordSys.SpaceFrame.Equinox	Equinox	OPT	2000.0
CoordSys.TimeFrame.Name	Timescale	OPT	TT
CoordSys.TimeFrame.Zero	Zero point of timescale in MJD	OPT	0.0

These reference frames apply to all spatial (sky), spectral, and time coordinates used in the SIAV2 query response (including Characterization) unless otherwise specified. Note that spatial coordinates are not limited to the celestial sphere; any spatial coordinate frame specified in the data model may be specified, including solar and planetary coordinate systems, although the default is ICRS.

5.2.6.6 Dataset Characterization Axis Metadata

The Characterization axis metadata specifies the type of physical quantity on each physical measurement axis as well as the observable.

UTYPE	*Description*	*Req*	*De-fault*
Char/FluxAxis.Ucd	ucd for flux	REC	
Char/SpectralAxis.Ucd	ucd for spectral coord	REC	
Char/TimeAxis.Ucd	ucd for time coord	REC	
Char/SpatialAxis.Ucd	ucd for time coord	REC	
Char/PolarizationAxis.Ucd	Ucd for pol axis	REC	

Values are specified as UCDs, as defined in the data model. For example, to specify that the flux axis is flux density per unit wavelength, the value "`phot.fluDens;em.wl`" would be given.

5.2.6.7 Characterization Coverage Metadata

The Coverage component of the Characterization data model (Char) describes the coverage of the dataset in each of the four primary measurement axes.

UTYPE	*Description*	
Char/SpatialAxis.Coverage.Location.coord	Observed position, e.g., RA DEC	MAN
Char/SpatialAxis.Coverage.Bounds.Extent	angular area, sq deg	MAN
Char/SpatialAxis.Coverage.Bounds.limits.LoLimit2Vec		
Char/SpatialAxis.Coverage.Bounds.limits.HiLimit2Vec		
Char/SpatialAxis.Coverage.Support.AreaType		
Char/SpatialAxis.Coverage.Support.Area	Accurate Field of View	OPT
Char/TimeAxis.Coverage.Location.coord	Midpoint of exposure (MJD)	MAN
Char/TimeAxis.Coverage.Bounds.Extent	Total elapsed exposure time	REC
Char/TimeAxis.Coverage.Bounds.limits.LoLimit	Start time	OPT
Char/TimeAxis.Coverage.Bounds.limits.HiLimit	Stop time	OPT
Char/TimeAxis.Coverage.Support.Extent	Effective exposure time	OPT
Char/SpectralAxis.Coverage.Location.coord	Midpoint of Spectral coord range	MAN
Char/SpectralAxis.Coverage.Bounds.Extent	Width of spectrum in meters	MAN
Char/SpectralAxis.Coverage.Bounds.limits.LoLimit	Start in spectral coordinate	REC
Char/SpectralAxis.Coverage.Bounds.limits.HiLimit	Stop in spectral coordinate	REC
Char/PolarizationAxis.enumeration		

Within Char, Coverage specifies the *location* (central or characteristic value), *bounds* (measurement limits), *support* (region covered within the bounds), for each measurement axis. The coordinate system reference frames specified in Coordsys apply here. Spatial coordinates are specified in units of decimal degrees, spectral coordinates in units of meters, and time coordinates in units of days. The Polarization axis is peculiar in this that it gives the list of available polarization parameters for the polarization system given by the UCD (eg Q, U, V parameters for Stokes system….)

5.2.6.8 Characterization Resolution and Sampling Metadata

The `Resolution` component of Characterization specifies the sampling and resolution estimates for the dataset.

UTYPE	*Description*	*Req*	*Default*
Char/SpectralAxis.Resolution	Spectral res. FWHM	REC	*BinSize*
Char/TimeAxis.Resolution	Temporal res. FWHM	OPT	*BinSize*
Char/SpatialAxis.Resolution	Spatial resolution of data	REC	
Char/SpectralAxis.SamplingPrecision.RefVal	Wavelength bin size	OPT	
Char/TimeAxis.SamplingPrecision.RefVal	Time bin size	OPT	
Char/SpectralAxis.SamplingPrecision.FillFactor	Sampling filling factor	OPT	1.0
Char/SpatialAxis.SamplingPrecision.FillFactor	Sampling filling factor	OPT	1.0
Char/TimeAxis.SamplingPrecision.FillFactor	Sampling filling factor	OPT	1.0

The spatial and spectral resolution **should** be specified. Note that, for consistency within Char, the spectral resolution is specified here in spectral coordinate units (FWHM in meters), unlike the SPECRP query parameter, which is specified as $\lambda/d\lambda$.

5.2.6.9 Characterization Accuracy and Error Metadata

The `Accuracy` component of Characterization specifies the sampling, resolution, and error estimates for the dataset.

UTYPE	*Description*	*Req*	*Default*
Char/FluxAxis.Accuracy.StatError	Statistical error	OPT	
Char/FluxAxis.Accuracy.SysError	Systematic error	OPT	
Char/FluxAxis. CalibrationStatus	Type of flux calibration	REC	calibrated
Char/SpectralAxis.Accuracy.StatError	Spectral coord meas. error	OPT	
Char/SpectralAxis.Accuracy.SysError	Spectral coord meas. error	OPT	
Char/SpectralAxis. CalibrationStatus	Type of coord calibration	REC	calibrated
Char/TimeAxis.Accuracy.StatError	Time coord statistical error	OPT	
Char/TimeAxis.Accuracy.SysError	Time coord systematic error	OPT	
Char/TimeAxis. CalibrationStatus	Type of coord calibration	OPT	calibrated
Char/SpatialAxis.Accuracy.StatError	Astrometric statistical error	REC	
Char.SpatialAxis.Accuracy.SysError	Systematic error	OPT	
Char.SpatialAxis. CalibrationStatus	Type of coord calibration	REC	calibrated

Both overall statistical and systematic error estimates may be specified. The calibration status of all three primary measurement axes as well as the observable **should** be given, otherwise "calibrated" is assumed.

5.2.7 Mapping Metadata

The mapping model specifies the image matrix and the transformation from image pixel coordinates to the specified world coordinate system (WCS). Image axes with any combination of spatial, spectral, time, or polarization coordinates are supported.

UTYPE	*Description*	*Req*	*Default*
Image Matrix Transform			
Mapping.NAxes	Number of image axes		
Mapping.NAxis[]	Length of each axis in pixels		
Mapping.CoordRefPixel[]	Reference pixel		
Mapping.CoordRefValue[]	WCS value at reference pixel		
Mapping.CDMatrix[]	Coord definition matrix		
Mapping.PCMatrix[]	Coord definition matrix		
Mapping.CDelt[]	World coord delta per pixel		
Mapping.AxisMap[]	Image-to-WCS axis mapping		
Mapping.WCSAxes	Number of WCS axes		
World Coord Transform			
Mapping.SpatialAxis.CoordType	Coordinate type as in FITS		
Mapping.SpatialAxis.Projection	Celestial projection		
Mapping.SpatialAxis.CoordFrame	Spatial coordinate frame		
Mapping.SpatialAxis.CoordEquinox	Coordinate equinox (if used)		
Mapping.SpatialAxis.CoordUnit	Unit for coordinate value		
Mapping.SpatialAxis.CoordName	Axis name (optional)		
Mapping.SpectralAxis.CoordType	Coordinate type as in FITS		
Mapping.SpectralAxis.Algorithm	Algorithm type as in FITS		
Mapping.SpectralAxis.RestFreq	Rest frequency of spectral line		
Mapping.SpectralAxis.RestWave	Rest wavelength of spectral line		
Mapping.SpectralAxis.CoordUnit	Unit for spectral coordinate value		
Mapping.SpectralAxis.CoordName	Axis name (optional)		
Mapping.SpectralAxis.CoordValue[]	Spectral value/band at pixel index		
Mapping.TimeAxis.CoordType	Time scale (UTC, TT, TAI, ...)		
Mapping.TimeAxis.CoordUnit	Time unit		
Mapping.TimeAxis.CoordName	Time axis name (optional)		
Mapping.TimeAxis.CoordValue[]	Time value at pixel index		
Mapping.TimeAxis.RefPosition	TOPOCENT, BARYCENT, ...		
Mapping.PolAxis.CoordType	Polarization system (Stokes etc.)		
Mapping.PolAxis.CoordName	Polarization axis name (optional)		
Mapping.PolAxis.CoordValue[]	Polarization type at pixel index		

In the above table, UTYPEs which have “[]” appended are vector-valued (the value is a string consisting of a sequence of numbers or string tokens delimited by spaces). The “[]” is not part of the actual UTYPE.

The Mapping model used in SIA is essentially the same as FITS WCS although it is represented slightly differently and has been somewhat simplified. The biggest deviation is in the representation of polarization which is represented as a simple lookup table assigning a polarization type to each pixel index on the polarization axis (e.g, “I”, “Q”, “U”, “V” for full Stokes).

A detailed description of the FITS WCS model is beyond the scope of the current document but can be found in FITS WCS papers 1-5. In summary the WCS transformation consists of a general linear transformation of the input image pixel coordinates (with the transform represented either as the CD matrix or as the PC matrix plus CDELT), followed optionally by a nonlinear transformation to produce the final world coordinates. To apply the linear transformation one first subtracts the coordinates of the reference pixel, then applies the transformation matrix, and finally adds the world coordinates at the reference pixel to establish the zero point. The result is a linear transformation of the input pixel coordinates to “intermediate” (linear) world coordinates.

In our representation the *AxisMap* is then used to map the axes of the intermediate world coordinates to the axes of the final world coordinate system. The spatial, spectral, time, or polarization transforms may then be applied independently to the associated intermediate world coordinte values. A nonlinear coordinate system may be represented either as a continuous function consisting of a well-known projection or algorithm of some sort (e.g., TAN, F2V, etc.), or as a lookup table wherein each pixel index on the axis is directly assigned a world coordinate.

[*More needs to be added here to fully specify this metadata, in particular the vector representation, allowable units, and allowable coordinate types and algorithms, following the FITS model, but this should suffice to demonstrate the approach.*]

5.2.8 Additional Service-Defined Metadata

A given service **may** return additional query response metadata not defined by the SIAV2 protocol. This additional metadata may take the form of additional table columns, or additional RESOURCE elements in the query response VOTable.

Service-defined output metadata **should** use service-defined UTYPEs and UCDs as long as they do not clash - and can be easily distinguished - from mandatory and reserved SIAV2 output columns.

5.2.9 Metadata Extension Mechanism

The metadata extension mechanism allows a data provider to add additional custom metadata to the query response to describe collection-specific details of the data. Extensions are provided in the query response only by explicit demand by the Extension query parameter (default) is none. Extension parameter value is an enumeration of

Extension metadata appears in the query response as additional `RESOURCE` elements in the VOTable. The format and contents of these `RESOURCE` elements is

up to the data provider. The ID-ref mechanism of VOTable is used to link extension elements to associated fields of the main query response VOTable.

The extension RESOURCE elements can contain PARAMs, TABLEs, or nested RESOURCEs. The ID-ref mechanism simply allows associating FIELDs from the main table to RESOURCEs, TABLEs and GROUPs in the extension RESOURCEs. It is actually the core of an indexing mechanism where the values of the referring FIELD are used as a key to associate a specific query response field to some additional information.

- In case the referred element is a TABLE in the extension RESOURCE, this TABLE must contain a FIELD identical to the referring FIELD and the indexing mechanism will provide a classical “a la RDBMS" jointure.
- In case the referred element is a RESOURCE of the extension, the key value is assumed to be the ID of a nested element (PARAM, RESOURCE, or TABLE) which is associated to the FIELD of the main table.
- It is also possible to provide backward linking by referring to FIELDs in the main section from elements in the extension.

As for any VOTable, the client software is informed about the role and guided for the usage of these extensions by the UTYPEs of the main query response and extension elements. UTYPE identifies the exact meaning of the element in a specific data model. In the context of the DAL metadata extension mechanism, UTYPE:

- gives the meaning of the association mechanism described above,
- givdes the meaning of all the extension elements.
- Current available IVOA data model and UTYPEs are defined for resource metadata, VOEvent, STC, Observation, (including Provenance and Characterization) and the Spectrum data model.
- Typical use of the Extension mechanism is for inclusion of Observation Field of Views (characterization support), Provenance details such as Filter transmission curve or specific data access mechanisms

5.3 Image Retrieval

Retrieval of individual images following a *queryData* is done via the *access reference* (“acref”) URL given in the query response record describing the image. Images are retrieved individually (but multiple transfers can normally proceed simultaneously). Transfers are synchronous but can stream, allowing arbitrarily large images to be transferred. The acref URL normally uses the HTTP protocol, although this is not required. In the case of a HTTP transport, image retrieval is done with a HTTP GET request.

Note:

For maximum flexibility image retrieval is URL-based via the access reference. The acref can point to a static image file or to a Web resource used to generate the image upon demand. A common way to do this is in SIA implementations is by having the

```
access reference URL resolve to an accessData request, which can
either copy out a static file or invoke server-side code to gen-
erate the requested image.
```

[*An additional possibility is for the acref URL to point to a VOSpace containing the image, once VOSpace is integrated into SIAV2.*]

5.3.1 Successful Output

The output of *a getImage* request should be a single image or other document returned with a MIME-type identifying the format of the data actually returned. If an image is returned it must conform to the simple image data model as outlined in Image Service Types above [*we need to add something to spell this out more clearly.*]. When the input acref points to a physical image the primary type of the MIME code should be "image/". Other MIME types are allowed, depending on the capabilities of the image service; for example, a MIME-type of "text/html" may be used when the acref URL points to an HTML description and/or preview of the image.

If a FITS image is returned the image **should** contain a valid FITS WCS. Any areas of the image which do not contain valid data, e.g., because the requested region extends beyond the bounds of the source image, should be flagged with a blank value, using the FITS keyword BLANK to identify the blank fill value used.

5.3.2 Error Response

If an error occurs and the requested image cannot be returned for any reason an error response **should** be returned as outlined in section XX. This consists of a VOTable with a single RESOURCE element containing an INFO with name "query_status" indicating that an error has occurred. The DESCRIPTION text in the INFO **should** describe the error which occurred.

While service errors shdould result in a protocol error as described above, not all errors are catchable in the serivce code and the client should be prepared for HTTP level errors as well. The response from a *getImage* request should always be inspected to verify that what was returned was what was expected.

5.4 AccessData

The *accessData* operation provides advanced capabilities for precise, client directed access to a specific image or image collection. Unlike *queryData*, *accessData* is not a query but rather a command to the service to generate a single image, and the output is not a table of candidate datasets but the actual requested image (or an error response if an error occurs). Use of *accessData* will generally require a prior call to *queryData* to get metadata describing the image or image collection to be accessed in order to plan subsequent access requests. *AccessData* is ideal for cases where an image with a specific orientation and scale is required, or for cases where the same image or image collection is to be repeatedly accessed, for example to generate multiple small image cutouts from an image, or to interactively view subsets of a large image cube.

5.4.1 Logical Access Model (informative)

The *accessData* operation is used to generate an image upon demand as directed by the client application. Upon successful execution the output is an image the parameters of which are what was specified by the client. The input may be an archive image, some other form of archive dataset (e.g., radio visibility or event data from which an image is to be generated), or a uniform data collection consisting of multiple data products from which the service automatically selects data to generate the output image.

In producing an output image from the input dataset *accessData* defines a number of transformations which it can perform. All are optional; in the simplest case the input dataset is an archival image which is merely delivered unchanged as the output image with no transformations having been performed. Another common case is to apply only a single transformation such as an image section or a general WCS-based projection. In the most complex case more than one transformation may be applied in sequence.

Starting from the input dataset of whatever type, the following transformations are available to generate the output image:

1. **Per-axis input filter**. The spatial, spectral, temporal or polarization axis (if any) can be filtered to select only the data of interest. Filters are defined as a range-list of acceptable ranges of values using the BAND, TIME, and POL parameters as specified later in this section, for the spectral, temporal, and polarization axes respectively. POS and SIZE are specified as for *queryData* except that the default coordinate frame matches that of the data being accessed (more on this below). Often the 1D BAND, TIME, and POL axes consist of a discrete set of samples in which case the filter merely selects the samples to be output, and the axis in question gets shorter (for example selecting a single band of a multiband image or a single polarization from a polarization cube). In the case of axis reduction where an axis is “scrunched”, possibly collapsing the entire axis to a single pixel, the filter can also be used to exclude data from the computation. Data which is excluded by a filter is not used for any subsequent computations as the output image is computed.
2. **WCS-based projection**. This step defines as output a pixellated image with the given image geometry (number of axes and length of each axis) and world coordinate system (WCS). Since the input dataset has a well-defined sampling and world coordinate system the operation is fully defined. If the input dataset is a pixellated image the image is reprojected as defined by the new WCS. If the input dataset is something more fundamental such as radio visibility or event data then the input data is sampled or imaged to produce the output image. Distortion, scale changes, rotation, cutting out, axis reduction, and dimensional reduction are all possible by correctly defining the output image geometry and WCS.

3. **Image section.** The *image section* provides a way to select a subset of a pixellated image by the simple expedient of specifying the *pixel* coordinates in the input image of the subset of data to be extracted (in our case here pixel coordinates would be specified relative to the image resulting from the application of steps 1 and 2 above). Axis flipping, dimensional reduction, and *axis reduction* (scrunching of an axis, combining a block of pixels into one pixel) can also be specified using an image section. *Dimensional reduction*, reducing the dimensionality of the image, occurs if an axis is reduced to a single value. The image section can provide a convenient technique for cutting out sections of images for applications that find it more natural to work in pixel than world coordinates. For example the section "`[*,*,3]`" applied to a cube would produce a 2D X-Y image as output, extracting the image plane at Z=3. Dimensional reduction affects only the dimensionality of the image pixel matrix; the WCS retains its original dimensionality.
4. **Function**. More complex transformations can be performed by applying an optional transformation function to an axis (typically the Z axis of a cube). For example the spectral index could be computed from a spectral data cube by computing the slope of the spectral distribution along the Z axis at each point [x,y,z] in the output image.

These processing stages define a *logical* set of transformations which can optionally be applied, in the order specifed, to the input dataset to compute the output image. Defining a logical order for application of the transformations is necessary in order for the overall operation to be well defined, as the output of each stage of the transformation defines the input to the following stage.

In terms of implementation the service is free to perform the computation in any way it wants so long as the result agrees with what is defined by the logical sequence of transformations. It is possible for example, for each pixel in the final output image, to trace backwards through the sequence of logical transformations to determine the signal from the input dataset contributing to that pixel. Any actual computation which reproduces the overall transformation is permitted.

In practice it may be possible to apply all the transformations at once in a single computation, or the actual computation may include additional finer-grained processing steps specific to the particular type of data being accessed and the sofware available for processing. The *AccessData* model specifies the final output image to be generated, but it is up to the service to determine the best way to produce this image given the data being accessed and the software available. The actual processing performed may vary greatly depending upon what type of data is accessed. [*We need to add some use cases to illustrate in concrete terms how this works.*].

Since *accessData* tells the service what to do rather than asking it what it can do, it is easy for the client to pose an invalid request which cannot be evaluated. In the event of an error the service should simply return an error status to the client indicating the nature of the error which occurred.

5.4.2 Input Parameters

5.4.2.1 Introduction

[*The remainder of this section is only partially complete; in particular the individual input parameters are only briefly described and not yet fully specified. The current text is intended more as a design dialog than as a formal specification and will be formalized as this section is developed.*]

Part of the motivation for *accessData* is to provide for precision image access without over-complicating *queryData*. Otherwise queryData would have more parameters and the semantics would need to differ for data discovery and access. If we move most of the advanced, less frequently used functionality into *accessData* both operations are likely to be more tightly defined, plus queryData will remain largely comparable in function to what we had in SIAV1 (except for being refined and updated to more modern standards). It becomes feasible to add advanced functionality via *accessData* without compromising the basic functionality of SIA.

The image generation parameters are considered advanced functionality hence these have been moved from queryData to accessData; however few SIAV1 services or clients have used these so the imact should not be great. Image generation of this sort requires more detailed knowledge of the service capabilities hence more logically belongs in accessData, plus WCS-based image generation is a fundamental part of the more sophisticated logical operation model defined for accessData.

The input parameters for *accessData* describe the image to be generated, and for the most part specify how to generate it. Hence this operation differs from *queryData*, which is more of a query to the service to see how close it can come to the ideal image required by the client application for analysis. *AccessData* is simpler in some ways because it just tells the service what to do. If the request is invalid then the service merely returns an error. Otherwise it generates the requested image (usually some sort of subset or transform) and returns the data to the client.

The input parameters to *accessData* are thus either the specifications for the image to be generated, or parameters specifying how to generate the image. The technique used to specify the image to be generated is simply to describe the image we want back. It is then up to the service to decide how best to generate it from the available data. This focuses the client on what it wants, and gives the service maximum scope for deciding how to service the request.

The most general way to specify an image to be generated is to specify the geometry and WCS of the desired output image. This specifies the dimensionality and size of the output image as well as the spatial, spectral, time and polarization coordinates of each output pixel. The input may be an archival image, a data collection or other set of images, or more fundamental data such as radio visibility or event data.

While specifying the geometry and WCS of the output image is the most general technique for image generation, a simple alternative approach is to specify a “cut-out” or “image section” of the original input image. This provides a simple techni-

que for cases where a single pixelated image is used as the input, allowing image-specific pixel coordinates to be used.

The input dataset may also be filtered (or subsetted) independently in each axis before any further processing is done. All transformations are optional. The full set of possible transformations is outlined in section XX above.

5.4.2.2 Input Dataset

To define an *accessData* operation first we must specify the input dataset, datasets, or data collection to be used to generate the new image.

Parameter	Sample value	Unit	Req	Datatype
PUBDID	ADS/col#R5983		REC	string
CREATORDID	ivo://auth/col#R1234		REC	string
COLLECTION	SDSS-DR7		REC	string

The input may be a single image (including cube data), a list of images (for example for construction of a mosaic image), a data collection, or a non-image primary dataset of some sort, such as radio visibility data or high energy event data. The input data may be anything so long as the service is capable of processing such data to make an image. In the case of an uniform data collection (for example a survey with full coverage of some region of the sky) the form of the input data is transparent to the client. [*We need to have a way for the queryData to define appropriate input data for accessData, including non-image datasets, so long as image data can be generated.*]

5.4.2.3 Filter Parameters

Specifying a filter to be applied to the input dataset is done using the POS,SIZE, BAND, TIME, and POL input parameters.

Parameter	*Sample value*	*Physical unit*	*Datatype*
POS	52,-27.8	degrees; defaults to ICRS	string
SIZE	0.05	degrees	double
BAND	2.7E-7/0.13	meters	string
TIME	1998-05-21/1999	ISO 8601 UTC	string
POL	U,V	-	string

The filter mechanism provides a simple way to “cutout” data with the region to be extracted being specified in world coordinates (if pixel coordinates are more convenient the image section mechanism may be used instead). For example, given a multiband image cube with bands U,B,V, the filter BAND=V would extract the 2D V-band image from the cube. Any combination of filters may be used. If a filter is combined with other transformations only the data which passes the filter will be used for subsequent processing.

The value of a BAND, TIME, or POL filter may be any valid range list (or simple list) containing either numerical or string values - whichever is used in the dataset for the axis in question. Ranges are useful for simple cutouts to limit the data to be returned or used for further processing to only what is required. An important

case of this occurs when a subsequent axis reduction or function operation is performed. For example if the spectral or time axis of a cube is reduced to a single value, a range list filter may be used to filter out night sky lines, or time intervals where the data is bad (RFI, spacecraft instrument issues, etc.). [*In the most general case this could require upload of a large filter.]*

Unlike *queryData* where a fixed set of standard units are used for discovery, *accessData* normally requires that the coordinate frames used with the filter parameters match the data being accessed. A prior call to *queryData* will provide the frame and unit information if it is not already known. Hence the "BAND=V" in our multiband image example above. With a full-Stoke radio polarization cube, "POL=I" would return the total intensity plane. If the spectral axis of a spectral data cube is in velocity units these are what would be used for the query, e.g., "BAND=-2.3/1.1".

An exception to the rule that only native frames and units should be used is made for the spatial filter specified using POS,SIZE. In this case the *default* spatial frame would be the native frame specified by the dataset or data collecting being accessed. Otherwise any supported spatial coordinate frame can be used as in the *accessData* version of POS,SIZE. The units are always decimal degrees.

5.4.2.4 Image Geometry and WCS

Specifying the geometry and WCS of the output image is the most general technique available for image generation, potentially allowing any output image to be generated which can be described by a standard WCS. Reprojection, scaling, rotation, cutting out, generation of a 2D slice through a cube at an arbitrary position and orientation, dimensional reduction, axis reduction, OTF imaging from more fundamental data, etc., can all be provided via this technique.

[*This section defines an updated, multidimensional version of the "image generation parameters" used in SIAV1. We still need to specify the image generation parameters; basically what is required is the Mapping, consisting of the image geometry NAxes and NAxis[] plus the output image WCS parameters. What is required is essentially a version of what is presented in section XX for the Mapping metadata expressed as a set of input parameters. Unfortunately, to support general access to cube data where we slice and dice arbitrary cubes we probably need a fairly general WCS - the simplifications adopted in SIAV1 for basic 2D reprojection are not enough. It is not so bad as it sounds though, as these various transformations are all variations on the same general model, which is essentially the same as the already widely implemented FITS WCS*].

5.4.2.5 Image Section

An alternative to specifying the output image in world coordinates is to specify a *section* (subset) of the input image in pixel coordinates. This is simple and well defined, and implicitly specifies the geometry and WCS of the output image. This limits the operation to a single input image, requiring knowledge of the image boundaries and extent, however it is simple and permits precision access to an image down to the pixel level. [T*his technique adopts the image section concept*

already used in cfitsio and IRAF image sections]. For example, "`[*,*,2]`" would return the plane at Z=2 of the referenced image cube. To cutout a portion of a 2D image we might use something like "`[100/200,350/450]`". To reduce the third axis to a single value we might have "`[*,*,//*]`". If we specify the pixels to be returned from an existing image this automatically specifies the WCS and geometry of the output image, plus we are guaranteed to specify an exact cutout with no interpolation required.

In the fully general approach where we specify the output image geometry it becomes possible to reduce the image dimensionality, e.g., produce a 2D plane from a 3D cube. This is known as *dimensional reduction*. It is nontrivial as the result may be an image of lesser dimension than the associated WCS. Furthermore when we reduce a axis of the image, we need to specify the *reduction algorithm* to be used, for example, sum, mean, or higher moments of each successive input sample (block of pixels to be reduced).

5.4.2.6 Functions

The final stage of processing defined by *accessData* provides for application of a general function to an axis. Unlike axis reduction, a function may change the type of data on the image axis.

[*This is an issue as the WCS model may not be general enough to represent such data. However this is potentially a very powerful feature for data analysis, particular of cube data; we need to determine a basic set of functions worth supporting at this level. A typical example of such a function for a spectral data cube is the spectral index, or slope of the spectrum at a given point along the spectral axis. In the most general case, general transformations should probably be moved out into a separate tasking mechanism*.]

5.4.2.7 Other Parameters

Finally since we are generating a single output image we can directly specify other aspects of the image to be returned, such as the image format and whether or not any compression is used. General processing parameters not specific to any one transformation are also included here.

Parameter	`Sample value`	Unit	`Req`	Datatype
REDUCE	sum			
FORMAT	`fits`	-	`MAN`	string
COMPRESS	`hcompress`		`REC`	string
RUNID			`REC`	string

REDUCE specifies the method to be used for axis reduction. The default is to sum the pixel values (signal) [*other possible values such as mean, median, etc. are TBD*; *we distinguish reduction, which does not change the axis type, from functions such as spectral index which can potentially generate any quantity*.]. FORMAT specifies the format of the image to be returned; it is an error if a format is requested which is not supported by the service. COMPRESS is used to enable compression, and specify the type of compression to be used, e.g., gzip, HCOMPRESS, RICE, and so forth. [*detailing the available compression algorithns, and compression levels for the lossy algorithhms, is TBD.*].

5.4.3 Request Response

If the request is successful an image is returned, otherwise an error response VOTable is returned.

5.5 StageData (optional)

This is used to initiate an asynchronous (batch) job to generate a single image or a number of images. Each image is specified by its acref as given in a prior call to *queryData*. Async execution is required for computationally expensive image generation tasks, e.g., on the fly imaging of radio data or large scale mosaic generation, or for scaling up, e.g., to pose a job to compute thousands of image cut-outs.

5.5.1 Input Parameters

TBD - list of acrefs essentially, plus any job options, passed via POST to the /async endpoint for the service.

5.5.2 Request Response

TBD - basically it is the URL of the UWS job to be used to monitor/control the job as it executes.

5.6 GetCapabilities (mandatory)

TBD - XML description of service capabilities.

5.7 GetAvailability (mandatory)

TBD - A simple request used to poll a service periodically to monitor it and verify that it is still up and ready to receive requests.

6 Basic Service Elements

Adapt from the comparable section in SSA or TAP or DAL2.

7 Service Registration

Say something about registering a service (we may also want to mention service verification).

8 Service Metadata

Logical definition of the service metadata describing the service capabilities. This is the information content for what is returned by *getCapabilities.*

Appendix A: “Appendix Title”

Insert appendix here

References

[1] R. Hanisch, *Resource Metadata for the Virtual Observatory* , http://www.ivoa.net/Documents/latest/RM.html

[2] R. Hanisch, M. Dolensky, M. Leoni, *Document Standards Management: Guidelines and Procedure* , http://www.ivoa.net/Documents/latest/DocStdProc.html