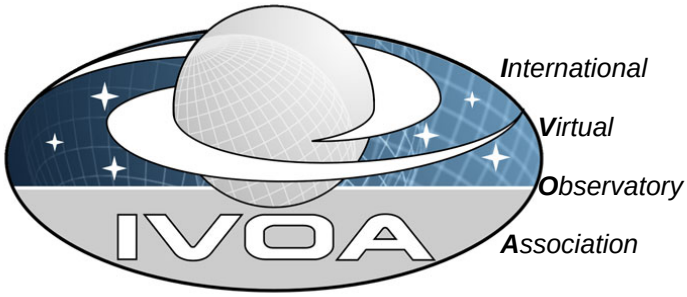

IVOA Registry Relational Schema Version 1.0

IVOA Working Draft 5 March 2013

Working Group:
Registry WG
This version:
http://www.ivoa.net/Documents/RegTAP-20130305
Latest version:
http://www.ivoa.net/Documents/RegTAP/
Previous versions:
None (The schema discussed here started its existence as part of the Registry Interfaces Version 2 working draft)
Authors:
Markus Demleitner
Paul Harrison
Marco Molinaro
Gretchen Greene
Theresa Dower
.

Abstract

Registries provide a mechanism with which VO applications can discover and select resources—first and foremost data and services—that are relevant for a particular scientific problem. This specification defines an interface for searching this resource metadata based on the IVOA's TAP protocol. It specifies a set of tables that comprise a useful subset of the information contained in the registry records, as well as the table's data content in terms of the XML VOResource data model. The general design of the system is geared towards allowing easy authoring of queries.

Status of this Document

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

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

Acknowledgments

This document has been developed in part with support from the German Astronomical Virtual Observatory (BMBF Bewilligungsnummer 05A08VHA).

Conformance-related definitions

The words "MUST ", "SHOULD", "MAY", "RECOMMENDED", and "OPTIONAL" (in upper or lower case) used in this document are to be interpreted as described in the IETF standard RFC 2119 *[std:RFC2119]*.

Contents

1. Introduction

In the Virtual Observatory (VO), registries provide a means for discovering useful resources, i.e., data and services. Individual publishers offer the descriptions for their resources ("resource records") in publishing registries. At the time of writing, there are roughly 14000 such resource records active within the VO, originating from 20 publishing registries.

The protocol spoken by these publishing registries, OAI-PMH, is not suitable for data discovery, and even if it were, data discovery would at least be fairly time consuming if each client had to query dozens or, potentially, hundreds of publishing registries.

To enable efficient data discovery nevertheless, there are services harvesting the resource records from the publishing registries and offering interfaces more suitable for querying by their users. The IVOA Registry Interfaces specification *[std:RI1]* defined, among other aspects of the VO registry system, such an interface using SOAP and an early draft of an XML-based query language.

This document provides an update to the query ("full registry") part of this specifiation, aimed towards usage with current VO standards, in particular TAP *[std:TAP]* and ADQL *[std:ADQL]*. It follows the model of ObsCore *[std:OBSCORE]* of defining a representation of a data model within a relational data base. In this case, the data model is a simplification of the VO's resource metadata interchange representation, the VOResource XML format *[std:VOR]*. The simplification yields 13 tables. This specification gives the table metadata together with rules for how to fill these tables from VOResource-serialized metadata records.

This architecture allows client applications to perform "canned" queries on behalf of their user as well as complex queries formulated directly by advanced users, using tools they already know.

It is a design goal of this specification that different registries operating on the same set of registry records will return identical responses for most queries.

1.1. The Relational Registry within the VO Architecture

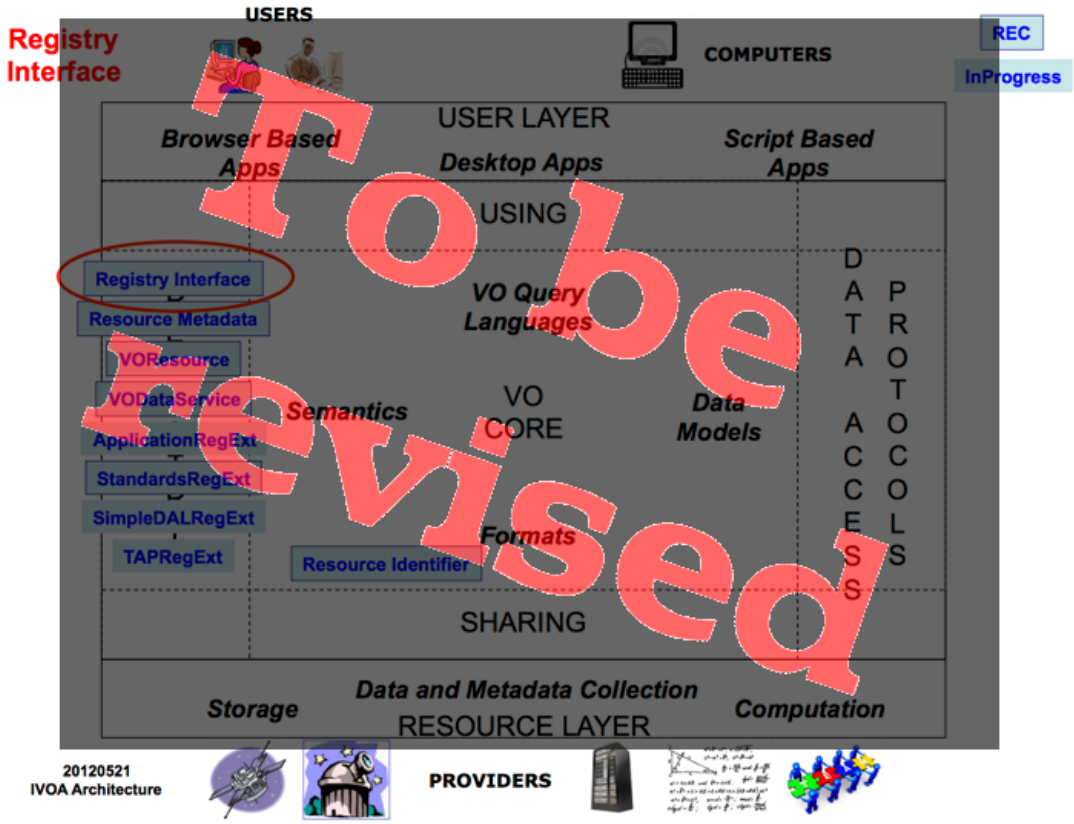

Figure 1: IVOA Architecture diagram with the Relational Registry specification (RegTAP) and the related standards marked up.

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

VOResource, v1.03 *[std:VOR]*
: VOResource sets the foundation for a formal definition of the data model for resource records via its schema definition. This document refers to concepts laid down there via the utypes given here.

VODataService, v1.1 *[std:VODS11]*
: VODataService extends the VOResource data model by important concepts and resource types (tablesets, data services, data collections). These concepts and types are reflected in the database schema. Again utypes on the tables and columns link this specification and VODataService.

Other Registry Extensions
: Registry extensions are VO standards defining how particular resources (e.g., Standards) or capabilities (e.g., IVOA defined interfaces) are described. Most aspects introduced by them are reflected in the `res_detail` table using utypes algorithmically generated from the XML schema documents given by these standards. This document should not in general need updates for registry extension updates. Still, in particular with a view to the caveat on the limits of the utypes generation algorithm given in section 5, we note the versions current as of this specification: SimpleDALRegExt 1.0 *[std:DALREGEXT]*, StandardsRegExt 1.0 *[std:STDREGEXT]*, TAPRegExt 1.0 *[std:TAPREGEXT]*, Registry Interfaces 1.0 *[std:RI1]*.

TAP, v1.0 *[std:TAP]*
: The queries against the schema defined here, and the results of these queries, will usually be transported using the Table Access Protocol TAP. It also allows discovering local additions to the registry relations via TAP's metadata publishing mechanisms.

IVOA Identifiers, v1.12 *[std:VOID]*
: IVOA identifiers are something like the primary keys within the VO registry. Also, the notion of an authority as laid down in IVOA Identifiers plays an important role as publishing registries can be viewed as a realization of a set of authorities.

This standard also relates to other IVOA standards:

ADQL, v2.0 *[std:ADQL]*
: The rules for ingestion are designed to allow easy queries given the constraints of ADQL 2.0. Also, we give three functions that extend ADQL using the language's built-in facility for user-defined functions.

utypes
: To link columns and tables in the relational resource model to entities defined in VOResource and ancillary specifications, we employ utypes; as of this writing, no IVOA recommendation conclusively defines utypes. It is expected that a Recommendation on utypes will tie building them to a formal data model definition in a language much simpler than XML schema. Since our data model comes in XML schema, we thus need custom rule. We do, however, strive to keep with the spirit of what is expected to become a recommendation.

RegTAP-STC
: This specification is complemented by a schema of four tables, also under the rr schema, that gives coverages of resources on the spatial, temporal, spectral and redshift axes. These will be defined in a separate document.

2. Design Considerations

In the design of the tables, the goal has been to preserve as much of VOResource and its extensions, including the element names, as possible.

An overriding consideration has been, however, to make natural joins between the tables behave usefully, i.e., to actually combine rows relevant to the same entity (i.e., resource, table, capability, etc.). To disambiguate column names that name the same concept on different entities (name, description, etc.) and would therefore interfere with the natural join, a shortened tag for the source object is prepended to the name. Thus, `description` from a resource becomes `res_description`, whereas the same element from a capability becomes `cap_description`.

Furthermore, camel-case identifiers have been converted to underscore-separated ones (thus, `standardId` becomes `standard_id`) to have all-lowercase column names; this saves potential headache if users choose to reference the columns using SQL delimited identifiers. Dashes in VOResource attribute names are converted to underscores, too, with the exception of `ivo-id`, which is just rendered `ivoid`.

3. Note on case normalization

ADQL has no operators for case-insensitive matching of strings. To allow for robust and straightforward queries nevertheless, most columns containing values not usually intended for display are required to be converted to lower case; in the table descriptions below, there are explicit requirements on case normalization near the end of each section. This is particularly important when the entities to be compared are defined to be case-insensitive (e.g., ucds, IVORNs, utypes). Client software that can inspect user-provided arguments (e.g., when filling template queries) should also convert the respective fields to lower-case.

This conversion MUST cover all ASCII letters, i.e., A through Z. The conversion SHOULD take place according to algorithm R2 in section 3.13, "Default Case Algorithms" of *[std:UNICODE]*. In practice, non-ASCII characters are not expected to occur in columns for which lowercasing is required.

Analogously, case-insensitive comparisons as required by some of the user-defined functions for the relational registry MUST compare the ASCII letters without regard for case. They SHOULD compare according to D144 in *[std:UNICODE]*.

4. QNames in VOResource attributes

VOResource and its extensions make use of XML QNames in attribute values, most prominently in `xsi:type`. The standard representation of these QNames in XML instance documents makes use of an abbreviated notation using prefixes declared using the xmlns mechanism as discussed in *[std:XMLNS]*. Within an ADQL database, no standard mechanism exists that could provide a similar mapping of URLs and abbreviations. The correct way to handle this problem would thus be to have full QNames in the database (e.g., `{http://www.ivoa.net/xml/ConeSearch/v1.0}ConeSearch` for the canonical `cs:ConeSearch`). This, of course, would make for excessively tedious and error-prone querying.

For various reasons, VOResource authors have always been encouraged to use a set of "standard" prefixes. This allows an easy and, to users, unsurprising exit from the problem of the missing xmlns declarations: For the representation of QNames within the database, these recommended prefixes are now mandatory. Future VOResource extensions define their mandatory prefixes themselves.

Following the existing practice, minor version changes are not in general reflected in the recommended prefixes—e.g., both VODataService 1.0 and VODataService 1.1 use `vs:`. For reference, here is a table of XML namespaces and prefixes for namespaces relevant to this specification:

oai	http://www.openarchives.org/OAI/2.0/
ri	http://www.ivoa.net/xml/RegistryInterface/v1.0
vg	http://www.ivoa.net/xml/VORegistry/v1.0
vr	http://www.ivoa.net/xml/VOResource/v1.0

dc	http://purl.org/dc/elements/1.1/
vs	http://www.ivoa.net/xml/VODataService/v1.0
vs	http://www.ivoa.net/xml/VODataService/v1.1
cs	http://www.ivoa.net/xml/ConeSearch/v1.0
sia	http://www.ivoa.net/xml/SIA/v1.0
ssap	http://www.ivoa.net/xml/SSA/v1.0
tr	http://www.ivoa.net/xml/TAPRegExt/v1.0
vstd	http://www.ivoa.net/xml/StandardsRegExt/v1.0

5. VOResource Utypes

This specification piggybacks on top of the well-established VOResource standard. It explicitely does not define a full data model, but rather something like a reasonably query-friendly view of a partial representation of one. The link between the actual data model, VOResource and extensions as defined by the XML Schema documents, and the fields within this database schema, is provided by utypes.

These utypes are generated from the XML schema files for VOResource and its extensions by means of an XSL stylesheet. They are intended to be compatible with the utypes generated according to a future IVOA specification. It is expected to be based on a modelling language called VO-DML; since VOResource is not modelled in VO-DML — and probably never will, since interoperability with other OAI-PMH applications makes XML Schema a compelling choice as the modelling language —, we have to provide some custom method to define utypes and their relation to modelling elements

The basic premise here is that 1:1 relationships are mapped to utypes by concatenating type and attribute names until the leaf element is reached; attributes and elements (in XSD terms) are not distinguished. 1:n relationships yield one utype for the attribute (the "collection utype", which is used on some tables), while utypes for the elements of the collection are rooted in the collection element's types. For example, the capabilities of a service have the collection utype `vr:Service.capability`, but the standard identifier of a capability has the utype `Capability's standardID`.

XML namespaces are ignored in utype generation. In particular, the `xsi:type` attributes that provide polymorphism for VOResource's `Resource` and `Capability` elements yield utypes ending in type, as in `vr:Resource.type` and `vr:Capability.type`.

At the utype level, polymorphism is reflected by using the names of the type definitions in which an element or attribute is defined as a utype fragment. For example, in the description of a single data collection there can be values for `vr:Resource.created` (as the definition of the `created` attribute is on `vr:Resource` itself) as well as `vs:DataCollection.instrument` (since `instrument` comes from the definition of the XML Schema `DataCollection` type in this case).

The data model name of the utypes, i.e., the part in front of the colon, is the canonical target name space prefix taken from the XML schema. The remarks on versioning those from section 4 apply here as well.

An XSLT stylesheet producing utypes together with short documentation strings from the XML schema files is given in Appendix A. Note that due to limitations of XSLT version 1, the complexity of XML schema, as well as considerations of implementation simplicity, this stylesheet is not completely general (see also comments in the file). Future VOResource extensions could break it if, e.g., they were to inherit from types derived from `vr:Resource` in a different file. As given here, the stylesheet produces the desired results for the versions of VOResource and its extensions given in section 1.1.

Note that for readability utypes are given in camel case throughout the normative text of this document, following the XML Schema names. However, utypes are generally considered to be case insensitive. Hence,

in the database tables, all utypes are stored all-lowercase; this is mandated by the case conversion rules in the table descriptions.

6. Discovering Relational Registries

The relational registry can be part of any TAP service. The presence of the tables discussed here is indicated by declaring support for the data model `Registry 1.0` with the IVORN `ivo://ivoa.net/std/RegTAP/vor` in the service's capabilities (see *[std:TAPREGEXT]*). Technically, this entails adding

```
<dataModel ivo-id="ivo://ivoa.net/std/RegTAP/vor">Registry 1.0</dataModel>
```

as a child of the capability element with the type `{http://www.ivoa.net/xml/TAPRegExt/v1.0}TableAccess`.

A client that knows the access URL of one relational registry can thus discover all other services exposing one. The "Find all TAP endpoints offering the relational registry" example in section 9 shows a query that does this.

7. VOResource Tables

In the following table descriptions, names and utypes of tables and columns are normative and MUST be used as given, and all-lowercase. Descriptions are not normative (as given, they usually are taken from the schema files of VOResource and its extensions with slight redaction). Registry operators MAY provide additional columns in their tables, but they MUST provide all columns given in this specification.

All table descriptions start out with brief remarks on the relationship of the table to the VOResource XML data model. Then, the columns are described in a selection of TAP_SCHEMA metadata. For each table, recommendations on explicit primary and foreign keys as well as indexed columns are given, where it is understood that primary and foreign keys are already indexed in order to allow efficient joins; these parts are not normative, but operators should ensure decent performance for queries assuming the presence of the given indices and relationships. Finally, lowercasing requirements (normative) are given.

The following tables make up the data model `ivo://ivoa.net/std/RegTAP/vor`:

Table Utype	Description
rr.capability vr:Service.capability	Pieces of behaviour of a resource.
rr.interface vr:Interface	Information on access modes of a capability.
rr.intf_param vs:InputParam	Input parameters for services.
rr.relationship vr:Resource.content.relationship	Relationships between resources, e.g., mirroring, derivation, but also providing access to data within a resource.
rr.res_date vr:Resource.curation.date	A date associated with an event in the life cycle of the resource. This could be creation or update. The role column can be used to clarify.
rr.res_detail N/A	Utype-value pairs for members of resource or capability and their derivations that are less used and/or from VOResource extensions. The pairs refer to a resource if cap_index is NULL, to the referenced capability otherwise.

Table Utype	Description
rr.res_role N/A	Entities, i.e., persons or organizations, operating on resources: creators, contacts, publishers.
rr.res_schema vs:TableSchema	Sets of tables related to resources.
rr.res_table vs:Table	(Relational) tables that are part of schemata or resources.
rr.resource vr:Resource	The resources, i.e., services, data collections, organizations, etc., present in this registry.
rr.subject vr:Resource.content.subject	Topics, object types, or other descriptive keywords about the resource.
rr.table_column vs:Table.column	Metadata on columns of tables pertaining to resources.
rr.validation N/A	Validation levels for resources and capabilities.

7.1. The resource Table

The `rr.resource` table contains most atomic members of `vr:Resource` that have a 1:1 relationship to the resource itself. Members of derived types are, in general, handled through the `res_detail` table even if 1:1 (see 7.13). The content_level, waveband, and rights members are 1:n but still appear here. If there are multiple values, concatenate them with hash characters (#). Use the `ivo_hashlist_has` ADQL extension function to check for the presence of a single value. This convention saves on tables while not complicating common queries significantly.

A local addition is the `creator_seq` column. It contains all content of the `name` elements below a resource element `curation` child's `creator` children, concatenated with a sequence of semicolon and blank characters (`"; "`). The individual parts must be concatenated preserving the sequence of the XML elements. The resulting string is primarily intended for display purposes ("author list") and is hence not case-normalized. It was added since the equivalent of an author list is expected to be a metadatum that is displayed fairly frequently, but also since the sequence of author names is generally considered significant. The `res_role` table, on the other hand, does not allow recovering the input sequence of the rows belonging to one resource.

The `res_type` column reflects the lower-cased value of the `ri:Resource` element's `xsi:type` attribute, where the canonical prefixes are used. While custom or experimental VOResource extensions may yield more or less arbitrary strings in that column, here is an enumeration of `res_types` from the VOResource and its IVOA-recommended extensions at the time of writing, together with explanations taken from the schema files and comments.

vg:authority
: A naming authority (these records allow resolving who is responsible for IVORNs with a certain authority; cf. *[std:VOID]*)

vg:registry
: A registry is considered a publishing registry if it contains a capability element with `xsi:type="vg:Harvest"`. Old, RegistryInterface 1-compliant registries also use this type with a capability of type `vg:Search`. The relational registry as specified here, while superceding these old `vg:Search` capabilities, does *not* use this type any more. See section 6 on how to locate services supporting it.

vr:organisation
: The main purpose of an organisation as a registered resource is to serve as a publisher of other resources.

vr:resource
: Any entity or component of a VO application that is describable and identifiable by a IVOA Identifier; while it is technically possible to publish such records, the authors of such records should probably be asked to use a more specific type.

vr:service
: A resource that can be invoked by a client to perform some action on its behalf

vs:catalogservice
: A service that interacts with one or more specified tables having some coverage of the sky, time, and/or frequency.

vs:dataservice
: A service for accessing astronomical data; publishers choosing this over `vs:catalogservice` probably intend to communicate that there are no actual sky positions in the tables exposed.

vs:datacollection
: A schema as a logical grouping of data which, in general, is composed of one or more accessible datasets. Use the `rr.relationship` table to find out services that allow access to the data (the `served_by` relation), and/or look for values for `vs:datacollection.accessurl` in `rr.res_detail`.

vstd:standard
: A description of a standard specification.

The `status` attribute of `vr:Resource` is considered an implementation detail of the XML serialization and is not kept here. Neither `inactive` nor `deleted` records may be kept in the `resource` table. Since all other tables in the relational registry should keep a foreign key on the `ivoid` column, this implies that only `active` records can be found in the relational registry; in other words, users can expect a resource they find using the relational registry to exist and work.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	Unambiguous reference to the resource conforming to the IVOA standard for identifiers
res_type vr:Resource.type	adql:VARCHAR(*)	Resource type (something like vs:datacollection, vs:catalogservice, etc)
created vr:Resource.created	adql:TIMESTAMP	The UTC date and time this resource metadata description was created. This timestamp must not be in the future. This time is not required to be accurate; it should be at least accurate to the day. Any insignificant time fields should be set to zero.
short_name vr:Resource.shortName	adql:VARCHAR(*)	A short name or abbreviation given to something. This name will be used where brief annotations for the resource name are required. Applications may use to refer to this resource in a compact display. One word or a few letters is recommended. No more than sixteen characters are allowed.
res_title	adql:VARCHAR(*)	The full name given to the resource

Name Utype	Type	Description
vr:Resource.title		
updated vr:Resource.updated	adql:TIMESTAMP	The UTC date this resource metadata description was last updated. This timestamp must not be in the future. This time is not required to be accurate; it should be at least accurate to the day. Any insignificant time fields should be set to zero.
content_level vr:ContentLevel	adql:VARCHAR(*)	Description of the content level or intended audience
res_description vr:Resource.content.description	adql:VARCHAR(*)	An account of the nature of the resource.
reference_url vr:Resource.content.referenceURL	adql:VARCHAR(*)	URL pointing to a human-readable document describing this resource.
creator_seq vr:Creator.name	adql:VARCHAR(*)	The creator(s) of the resource in the order given by the resource record author.
content_type vr:Resource.content.type	adql:VARCHAR(*)	Nature or genre of the content of the resource
source_format vr:Resource.content.source.format	adql:VARCHAR(*)	The format of source_value. Recognized values include "bibcode", referring to a standard astronomical bibcode (http://cdsweb.u-strasbg.fr/simbad/refcode.html).
source_value vr:Resource.content.source	adql:VARCHAR(*)	A bibliographic reference from which the present resource is derived or extracted.
version vr:Resource.curation.version	adql:VARCHAR(*)	Label associated with creation or availablilty of a version of a resource.
region_of_regard vr:Resource.coverage.regionOfRegard	adql:REAL	A single numeric value representing the angle, given in decimal degrees, by which a positional query against this resource should be "blurred" in order to get an appropriate match.
waveband vs:Waveband	adql:VARCHAR(*)	A hash-separated list of regions of the electro-magnetic spectrum that the resource's spectral coverage overlaps with.
rights vr:Rights	adql:VARCHAR(*)	Information about rights held in and over the resource.

This table should have the `ivoid` column explicitely set as its primary key.

The following columns MUST be lowercased during ingestion: `ivoid`, `res_type`, `content_level`, `content_type`, `source_format`, `waveband`, `footprint_ivoid`. Clients are advised to query the

`res_description` and `res_title` columns using the the `ivo_hasword` function, and to use `ivo_hashlist_has` on `content_level`, `waveband`, and `rights`.

7.2. The res_role Table

This table subsumes the contact, publisher, contributor, and creator members of the VOResource data model. They have been combined into a single table to reduce the total number of tables, and also in anticipation of a unified data model for such entities in future versions of VOResource.

The actual role is given in the `base_role` column, which can be one of `contact`, `publisher`, or `creator`. Depending on this value, here are the utypes for the table fields (we have abbreviated `vr:Resource.curation.publisher` as Rcp, `vr:Contact` as Co, `vr:Creator` as Cr, and `vr:ResourceName` as R):

base_role value	contact	publisher	creator	contributor
role_name	Co.name	Rcp	Cr.name	R
role_ivoid	Co.name.ivoid	Rcp.ivoid	Cr.name.ivoid	R.ivoid
address	Co.address	N/A	N/A	N/A
email	Co.email	N/A	N/A	N/A
telephone	Co.telephone	N/A	N/A	N/A
logo	Co.logo	N/A	Cr.logo	N/A

Not all columns are available for each role type in VOResource. For example, contacts have no logo, and creators no telephone members. Unavailable metadata MUST be represented with NULL values in the corresponding columns.

Note that, due to current practice in the VO, it is not easy to predict what `role_name` will contain; it could be a single name, where again the actual format is unpredictable (full first name, initials in front or behind, or even a project name), but it could as well be a full author list. Thus, when matching against `role_names`, you will have to use rather lenient regular expressions. Changing this, admittedly regrettable, situation would probably require a change in the VOResource schema.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
role_name N/A	adql:VARCHAR(*)	The real-world name or title of a person or organization
role_ivoid N/A	adql:VARCHAR(*)	An IVOA identifier of a person or organization
address N/A	adql:VARCHAR(*)	A mailing address for a person or organization
email N/A	adql:VARCHAR(*)	An email address the entity can be reached at
telephone N/A	adql:VARCHAR(*)	A telephone number the entity can be reached at
logo N/A	adql:VARCHAR(*)	URL pointing to a graphical logo, which may be used to help identify the entity

Name Utype	Type	Description
base_role N/A	adql:VARCHAR(*)	The role played by this entity; this is one of contact, publisher, and creator

The `ivoid` column should be an explicit foreign key into the `resource` table. It is recommended to maintain indexes on at least the `role_name` column, ideally in a way that supports regular expressions.

The following columns MUST be lowercased during ingestion: `ivoid`, `role_ivoid`, `base_utype`. Clients are advised to query the remaining columns, in particular `role_name`, case-insensitively, e.g., using `ivo_nocasematch`.

7.3. The subject Table

Since subject queries are expected to be frequent and perform relatively complex checks (e.g., resulting from thesaurus queries in the clients), the subjects are kept in a separate table rather than being hash-joined like other string-like 1:n members of resource.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
subject xs:token	adql:VARCHAR(*)	Topics, object types, or other descriptive keywords about the resource.

The `ivoid` column should be an explicit foreign key into `resource`. It is recommended to index the `subject` column, preferably in a way that allows to process case-insensitive and pattern queries using the index.

The `ivoid` column MUST be lowercased during ingestion. Clients are advised query the `subject` column case-insensitively, e.g., using `ivo_nocasematch`.

7.4. The capability Table

The capability table describes the capabilities of a resource; it only contains the members of the base type `vr:Capability`. Members of derived types are kept in the `res_detail` table (see 7.13).

The table has an integer-typed column `cap_index` to disambiguate multiple capabilities on a single resource. Operators are free to choose the actual values as convenient, although it is recommended to just enumerate the capabilities in their physical sequence per-resource.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
cap_index N/A	adql:SMALLINT	Running number of this capability within the resource.
cap_type vr:Capability.type	adql:VARCHAR(*)	The type of capability covered here.

Name Utype	Type	Description
cap_description vr:Capability.description	adql:VARCHAR(*)	A human-readable description of what this capability provides as part of the over-all service
standard_id vr:Capability.standardID	adql:VARCHAR(*)	A URI for a standard this capability conforms to.

This table should have an explicit primary key made up of `ivoid` and `cap_index`. The `ivoid` column should be an explicit foreign key into `resource`. It is recommended to maintain indexes on at least the `cap_type` and `standard_id` columns.

The following columns MUST be lowercased during ingestion: `ivoid`, `cap_type`, `standard_id`. Clients are advised to query the `cap_description` column using the `ivo_hasword` function.

7.5. The res_schema Table

The `res_schema` table corresponds to VODataService's `schema` element. It has been renamed to avoid clashes with the SQL reserved word `SCHEMA`.

The table has an integer-typed column `schema_index` to disambiguate multiple schema elements on a single resource. Operators are free to choose the actual values as convenient, although it is recommended to just enumerate the schema elements in their physical sequence per-resource.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
schema_index N/A	adql:SMALLINT	A running number for the res_schema rows belonging to a resource.
schema_description vs:TableSchema.description	adql:VARCHAR(*)	A free text description of the table-set explaining in general how all of the tables are related.
schema_name vs:TableSchema.name	adql:VARCHAR(*)	A name for the set of tables.
schema_title vs:TableSchema.title	adql:VARCHAR(*)	A descriptive, human-interpretable name for the table set.
schema_utype vs:TableSchema.utype	adql:VARCHAR(*)	An identifier for a concept in a data model that the data in this schema as a whole represent.

This table should have an explicit primary key made up of `ivoid` and `schema_index`. The `ivoid` column should be an explicit foreign key into `resource`.

The following columns MUST be lowercased during ingestion: `ivoid`, `schema_name`, `schema_utype`. Clients are advised to query the `schema_description` and `schema_title` columns using the the `ivo_hasword` function.

7.6. The res_table Table

The `res_table` table models VODataService's `table` element. It has been renamed to avoid name clashes with the SQL reserved word TABLE.

The table contains an integer-typed column `table_index` to disambiguate multiple tables on a single resource. Operators are free to choose the actual values as convenient, although it is recommended to just enumerate the tables in their physical sequence per-resource (but not per schema—`table_index` MUST be unique within a resource).

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
schema_index N/A	adql:SMALLINT	Index of the schema this table belongs to, if it belongs to a schema (otherwise NULL).
table_description vs:Table.description	adql:VARCHAR(*)	A free-text description of the table's contents
table_name vs:Table.name	adql:VARCHAR(*)	The fully qualified name of the table. This name should include all catalog or schema prefixes needed to distinguish it in a query.
table_index N/A	adql:INTEGER	An artificial counter for the tables belonging to a resource
table_title vs:Table.title	adql:VARCHAR(*)	A descriptive, human-interpretable name for the table
table_type vs:Table.type	adql:VARCHAR(*)	A name for the role this table plays. Recognized values include "output", indicating this table is output from a query; "base_table", indicating a table whose records represent the main subjects of its schema; and "view", indicating that the table represents a useful combination or subset of other tables. Other values are allowed.
table_utype vs:Table.utype	adql:VARCHAR(*)	An identifier for a concept in a data model that the data in this table as a whole represent.

This table should have an explicit primary key made up of `ivoid` and `table_index`. The pair `ivoid`, `schema_index` should be an explicit foreign key into `res_schema`. It is recommended to maintain an index on at least the `table_description` column, ideally one suited for queries with `ivo_hasword`.

The following columns MUST be lowercased during ingestion: `ivoid`, `table_name`, `table_type`, `table_utype`. Clients are advised to query the `table_description` and `table_title` columns using the the `ivo_hasword` function.

7.7. The table_column Table

The `table_column` table models the content of VOResource's `column` element. The table has been renamed to avoid a name clash with the SQL reserved word `COLUMN`.

Since it is expected that queries for column properties will be fairly common in advanced queries, it is the column table that has the unprefixed versions of common member names (name, description, ucd, utype, etc).

The `flag` column contains a concatenation of all values of a `column` element's `flag` children, separated by hash characters. Use the `ivo_hashlist_has` function in queries against `flag`.

The `table_column` table also includes information from VODataService's data type concept. VODataService 1.1 includes several type systems (VOTable, ADQL, Simple). The `typesystem` column contains the value of the column's `datatype` child, with the VODataService XML prefix fixed to vs; hence, this column will contain one of NULL, `vs:TAPType`, `vs:SimpleDataType`, and `vs:VOTableType`.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
table_index N/A	adql:INTEGER	An artificial counter for the tables belonging to a resource
description vs:BaseParam.description	adql:VARCHAR(*)	A free-text description of the column's contents
name vs:BaseParam.name	adql:VARCHAR(*)	The name of the column
ucd vs:BaseParam.ucd	adql:VARCHAR(*)	A unified content descriptor that describes the scientific content of the parameter.
unit vs:BaseParam.unit	adql:VARCHAR(*)	The unit associated with all values in the column.
utype vs:BaseParam.utype	adql:VARCHAR(*)	An identifier for a role in a data model that the data in this column represents.
std vs:TableParam.std	adql:SMALLINT	If 1, the meaning and use of this parameter is reserved and defined by a standard model. If 0, it represents a database-specific parameter that effectively extends beyond the standard.
datatype vs:DataType	adql:VARCHAR(*)	The type of the data contained in the column.
extended_schema vs:DataType.extendedSchema	adql:VARCHAR(*)	An identifier for the schema that the value given by the extended attribute is drawn from.
extended_type vs:DataType.extendedType	adql:VARCHAR(*)	The data value represented by this type can be interpreted as of a custom type identified by the value of this attribute.
arraysize vs:DataType.arraysize	adql:VARCHAR(*)	The shape of the array that constitutes the value.

Name Utype	Type	Description
delim vs:DataType.delim	adql:VARCHAR(*)	The string that is used to delimit elements of an array value when arraysize is not '1'.
type_system vs:DataType.type	adql:VARCHAR(*)	The type system used, as a QName with a canonical prefix; this will ususally be one of vs:SimpleDataType, vs:VOTableType, and vs:TAPType.
flag vs:TableParam.flag	adql:VARCHAR(*)	hash-separated keywords representing traits of the column. Recognized values include "indexed", "primary", and "nullable".

The pair `ivoid`, `table_index` should be an explicit foreign key into `res_table`. It is recommended to maintain indexes on at least the `description`, `name`, `ucd`, and `utype` columns, where the index on `description` should ideally be able to handle queries using `ivo_hasword`.

The following columns MUST be lowercased during ingestion: `ivoid`, `name`, `ucd`, `utype`, `datatype`, `type_system`. Clients are advised to query the `description` column using the `ivo_hasword` function, and to query the `flag` column using the `ivo_hashlist_has` function.

7.8. The interface Table

The `interface` table subsumes both the `vr:Interface` and `vr:accessURL` types from VOResource. The integration of `accessURL` into the `interface` table means that an interface in the relational registry can only have one access URL, where in VOResource it can have many. In practice, this particular VOResource capability has not been used by registry record authors. Since access URLs are probably the item most queried for, it seems warranted to save one indirection when querying for them.

Should interfaces with multiple access URLs become necessary in the future, we propose a slight denormalization by creating mulitple interfaces with one access URL each within the registry's ingestion logic.

The table contains an integer-typed column `intf_index` to disambiguate multiple interfaces on a single capability. Operators are free to choose the actual values as convenient, although it is recommended to just enumerate the interfaces in their physical sequence per capability.

Analogous to `resource.res_type`, the `intf_type` column contains type names; VOResource extensions can define new types here, but at the time of writing, the following types are mentioned in IVOA-recommended schemata:

vs:paramhttp
: A service invoked via an HTTP query with either form-urlencoded or multipart form-data parameters.

vr:webbrowser
: A (form-based) interface intended to be accessed interactively by a user via a web browser.

vg:oaihttp
: A standard OAI PMH interface using HTTP queries with form-urlencoded parameters.

vg:oaisoap
: A standard OAI PMH interface using a SOAP Web Service interface.

vr:webservice
: A Web Service that is describable by a WSDL document.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
cap_index N/A	adql:SMALLINT	The index of the parent capability
intf_index N/A	adql:SMALLINT	A running number for the interfaces of a capability.
intf_type vr:Interface.type	adql:VARCHAR(*)	The type of the interface (WebBrowser, ParamHTTP, etc).
intf_role vr:Interface.role	adql:VARCHAR(*)	A tag name the identifies the role the interface plays in the particular capability. If the value is equal to "std" or begins with "std:", then the interface refers to a standard interface defined by the standard referred to by the capability's standardID attribute.
std_version vr:Interface.version	adql:VARCHAR(*)	The version of a standard interface specification that this interface complies with. When the interface is provided in the context of a Capability element, then the standard being refered to is the one identified by the Capability's standardID element.
query_type vs:ParamHTTP.queryType	adql:VARCHAR(*)	The type of HTTP request, either GET or POST.
result_type vs:ParamHTTP.resultType	adql:VARCHAR(*)	The MIME type of a document returned in the HTTP response.
wsdl_url vr:WebService.wsdlURL	adql:VARCHAR(*)	The location of the WSDL that describes this Web Service. If not provided, the location is assumed to be the accessURL with '?wsdl' appended.
url_use vr:AccessURL.use	adql:VARCHAR(*)	A flag indicating whether this should be interpreted as a base URL, a full URL, or a URL to a directory that will produce a listing of files.
access_url vr:AccessURL	adql:VARCHAR(*)	The URL at which the interface is found.

This table should have the triple `ivoid`, `cap_index`, and `intf_index` as an explicit primary key. The pair `ivoid`, `cap_index` should be an explicit foreign key into `capability`. It is recommended to maintain an index on at least the `intf_type` column.

The following columns MUST be lowercased during ingestion: `ivoid`, `intf_type`, `intf_role`, `std_version`, `query_type`, `result_type`, `url_use`.

7.9. The intf_param Table

The `intf_param` table keeps information on the parameters avalailable on interfaces. It is therefore closely related to `table_column`, but the differences between the two are significant enough to warrant a separation between the two tables. Since the names of common column attributes are used where applicable in both tables (e.g., name, ucd, etc), the two tables cannot be (naturally) joined.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
cap_index N/A	adql:SMALLINT	The index of the parent capability
intf_index N/A	adql:SMALLINT	A running number for the interfaces of a capability.
description vs:BaseParam.description	adql:VARCHAR(*)	A free-text description of the column's contents
name vs:BaseParam.name	adql:VARCHAR(*)	The name of the column
ucd vs:BaseParam.ucd	adql:VARCHAR(*)	A unified content descriptor that describes the scientific content of the parameter.
unit vs:BaseParam.unit	adql:VARCHAR(*)	The unit associated with all values in the column.
utype vs:BaseParam.utype	adql:VARCHAR(*)	An identifier for a role in a data model that the data in this column represents.
std vs:InputParam.std	adql:SMALLINT	If 1, the meaning and use of this parameter is reserved and defined by a standard model. If 0, it represents a database-specific parameter that effectively extends beyond the standard.
datatype vs:DataType	adql:VARCHAR(*)	The type of the data contained in the column.
extended_schema vs:DataType.extendedSchema	adql:VARCHAR(*)	An identifier for the schema that the value given by the extended attribute is drawn from.
extended_type vs:DataType.extendedType	adql:VARCHAR(*)	The data value represented by this type can be interpreted as of a custom type identified by the value of this attribute.
use_param vs:InputParam.use	adql:VARCHAR(*)	An indication of whether this parameter is required to be provided for the application or service to work properly.

The triple `ivoid`, `cap_index`, and `intf_index` should be an explicit foreign key into `interface`. It is recommended to maintain indexes on at least the `description`, `name`, `ucd`, and `utype` columns.

See section 7.7 for requirements on lowercasing of column attributes.

7.10. The relationship Table

The relationship element is a slight denormalization of the `vr:Relationship` type: Whereas in VOResource, a single relationship element can take several IVORNs, in the relational model, the pairs are stored directly. It is straightforward to translate between the two representations in the database ingestor.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
relationship_type vr:Relationship.relationshipType	adql:VARCHAR(*)	The named type of relationship; this can be mirror-of, service-for, served-by, derived-from, related-to, or something user-defined.
related_id vr:Relationship.relatedResource.ivoId	adql:VARCHAR(*)	The URI form of the IVOA identifier for the resource refered to.
related_name vr:Relationship.relatedResource	adql:VARCHAR(*)	The name of resource that this resource is related to.

The `ivoid` column should be an explicit foreign key into the `resoure` table. You should index at least the `related_id` column.

The following columns MUST be lowercased during ingestion: `ivoid`, `relationship_type`, `related_id`.

7.11. The validation Table

The `validation` subsumes the `vr:validationLevel` members of both `vr:Resource` and `vr:Capability`.

If the `cap_index` column in `NULL` the validation comprises the entire resource. Otherwise, only the referenced capability has been validated.

While it is recommended that harvesters only accept resource records from their originating registries, rows in the validation table could very well originate from third-party registries. Hence, rows in `rr.validation` for the same resource might originate from completely different registries. This can trigger potentially problematic behaviour when the original registry updates its resource record in that naive implementations will lose all third-party validation rows; also, the semantics of validation results over updates of resources and/or resource records is not fully mapped to VOResource. Implementations are free to handle or ignore validation results as they see fit, and they may add validation results of their own.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
validated_by vr:Validation.validatedBy	adql:VARCHAR(*)	The IVOA ID of the registry or organisation that assigned the validation level.
level vr:Validation.validationLevel	adql:SMALLINT	A numeric grade describing the quality of the resource description, when applicable, to be used to in-

Name Utype	Type	Description
		dicate the confidence an end-user can put in the resource as part of a VO application or research study.
cap_index N/A	adql:SMALLINT	If non-NULL, the validation only refers to the the capability referenced here.

The `ivoid` should be an explicit foreign key into `resource`. Note, however, that `ivoid`, `cap_index` is *not* a foreign key into `capability` since `cap_index` may be `NULL` (in case the validation addresses the entire resource).

The following columns MUST be lowercased during ingestion: `ivoid`, `validated_by`.

7.12. The res_date Table

The `res_date` table contains information gathered from `vr:Curation`'s date children.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
date_value vr:Date	adql:TIMESTAMP	A date associated with an event in the life cycle of the resource.
value_role vr:Date.role	adql:VARCHAR(*)	A string indicating what the date refers to.

The `ivoid` column should be an explicit foreign key into `resource`.

The following columns MUST be lowercased during ingestion: `ivoid`, `value_role`.

7.13. The res_detail Table

The `res_detail` table is RegTAP's primary means for extensability as well as a fallback for less-used simple metadata. Conceptually, it stores triples of resource entity references, utypes, and values, where resource entities can be resource records themselves or capabilities. Thus, metadata with values that are either atomic or sets of atoms can be represented in this table.

As long as the metadata that needs to be represented in the relational registry for new VOResource extensions is simple enough, no changes to the schema defined here will be neccessary as these are introduced. Instead, the extension itself simply defines new utypes to be added in `res_detail`.

Some complex metadata—`tr:languageFeature` or `vstd:key` being examples—cannot be kept in this table. If a representation of such information in the relational registry is required, this standard will need to be changed.

The following list gives utypes from the registry extensions that were recommendations at the time of writing. The utypes double here as partial XPath-like expressions into the VOResource XML trees, which should make the generation of `res_details` rows fairly straightforward (see also the rules for utype generation given in section 5).

From the following list, operators MUST put all rows generatable for the utypes marked with an exclamation mark into `res_details`. The remaining metadata may be provided if convenient; it mostly concerns test queries and other curation-type information not likely to be useful to normal users. Individual ingestors MAY choose to expose additional metadata using other utypes.

In addition to the metadata listed here, metadata defined in future IVOA-approved VOResource extensions MUST or SHOULD be present in `res_details` as the extensions require it.

cs:ConeSearch.maxRecords (!)
: The largest number of records that the service will return.

cs:ConeSearch.testQuery.catalog
: the catalog to query.

cs:ConeSearch.testQuery.dec
: the declination of the search cone's center in decimal degrees.

cs:ConeSearch.testQuery.extras
: any extra (non-standard) parameters that must be provided (apart from what is part of base URL given by the accessURL element).

cs:ConeSearch.testQuery
: A query that will result in at least on matched record that can be used to test the service.

cs:ConeSearch.testQuery.ra
: the right ascension of the search cone's center in decimal degrees.

cs:ConeSearch.testQuery.sr
: the radius of the search cone in decimal degrees.

cs:ConeSearch.testQuery.verb
: the verbosity level to use where 1 means the bare minimum set of columns and 3 means the full set of available columns.

cs:ConeSearch.verbosity (!)
: True if the service supports the VERB keyword; false, otherwise.

sia:SimpleImageAccess.imageServiceType (!)
: The class of image service: Cutout, Mosaic, Atlas, Pointed

sia:SimpleImageAccess.maxFileSize (!)
: The maximum image file size in bytes.

sia:SimpleImageAccess.maxImageExtent.lat
: The maximum size in the latitdude (Dec.) direction

sia:SimpleImageAccess.maxImageExtent.long
: The maximum size in the longitude (R.A.) direction

sia:SimpleImageAccess.maxImageSize.lat
: The image size in the latitdude (Dec.) direction in pixels

sia:SimpleImageAccess.maxImageSize.long
: The image size in the longitude (R.A.) direction in pixels

sia:SimpleImageAccess.maxQueryRegionSize.lat
: The maximum size in the latitdude (Dec.) direction

sia:SimpleImageAccess.maxQueryRegionSize.long
: The maximum size in the longitude (R.A.) direction

sia:SimpleImageAccess.maxRecords (!)
: The largest number of records that the Image Query web method will return.

sia:SimpleImageAccess.testQuery.extras
: any extra (particularly non-standard) parameters that must be provided (apart from what is part of base URL given by the accessURL element).

sia:SimpleImageAccess.testQuery
: a set of query parameters that is expected to produce at least one matched record which can be used to test the service.

sia:SimpleImageAccess.testQuery.pos.lat
: The sky position in the latitdude (Dec.) direction

sia:SimpleImageAccess.testQuery.pos.long
: The sky position in the longitude (R.A.) direction

sia:SimpleImageAccess.testQuery.size.lat
: The maximum size in the latitdude (Dec.) direction

sia:SimpleImageAccess.testQuery.size.long
: The maximum size in the longitude (R.A.) direction

sia:SimpleImageAccess.testQuery.verb
: the verbosity level to use where 0 means the bare minimum set of columns and 3 means the full set of available columns.

ssap:SimpleSpectralAccess.complianceLevel
: The category indicating the level to which this instance complies with the SSA standard.

ssap:CreationType (!)
: The category that describes the process used to produce the dataset.

ssap:DataSource (!)
: The category specifying where the data originally came from.

ssap:SupportedFrame (!)
: The STC name for a world coordinate system frame supported by this service.

ssap:SimpleSpectralAccess.defaultMaxRecords (!)
: The largest number of records that the service will return when the MAXREC parameter not specified in the query input.

ssap:SimpleSpectralAccess.maxAperture
: The largest aperture that can be supported upon request via the APERTURE input parameter by a service that supports the special extraction creation method.

ssap:SimpleSpectralAccess.maxRecords (!)
: The hard limit on the largest number of records that the query operation will return in a single response

ssap:SimpleSpectralAccess.testQuery
: a set of query parameters that is expected to produce at least one matched record which can be used to test the service.

ssap:SimpleSpectralAccess.testQuery.pos.lat
: The latitude (e.g. Declination) of the center of the search position in decimal degrees.

ssap:SimpleSpectralAccess.testQuery.pos.long
: The longitude (e.g. Right Ascension) of the center of the search position in decimal degrees.

ssap:SimpleSpectralAccess.testQuery.pos.refframe
: the coordinate system reference frame name indicating the frame to assume for the given position. If not provided, ICRS is assumed.

ssap:SimpleSpectralAccess.testQuery.queryDataCmd
: Fully specified test query formatted as an URL argument list in the syntax specified by the SSA standard. The list must exclude the REQUEST argument which is assumed to be set to "queryData".

ssap:SimpleSpectralAccess.testQuery.size
: the size of the search radius.

tr:DataModelType.ivoid (!)
: The IVORN of the data model supported by a TAP service.

tr:DataModelType (!)
: The short, human-readable name of a data model supported by a TAP service; for most applications, clients should rather constrain tr:datamodeltype.ivoid.

tr:Language.name (!)
: An IVOA defined data model, identified by an IVORN intended for machine consumption and a short label intended for human comsumption.

tr:OutputFormat.ivoid (!)
: An optional IVORN of the output format.

tr:OutputFormat.mime (!)
: The MIME type of a format.

tr:TableAccess.executionDuration.default
: The value of this limit for newly-created jobs, given in seconds.

tr:TableAccess.executionDuration.hard
: The value this limit cannot be raised above, given in seconds.

tr:TableAccess.outputLimit.default
: The value of this limit for newly-created jobs.

tr:TableAccess.outputLimit.default.unit
: The unit of the limit specified.

tr:TableAccess.outputLimit.hard
: The value this limit cannot be raised above.

tr:TableAccess.outputLimit.hard.unit
: The unit of the limit specified.

tr:TableAccess.retentionPeriod.default
: The value of this limit for newly-created jobs, given in seconds.

tr:TableAccess.retentionPeriod.hard
: The value this limit cannot be raised above, given in seconds.

tr:TableAccess.uploadLimit.default
: The value of this limit for newly-created jobs.

tr:TableAccess.uploadLimit.default.unit
: The unit of the limit specified.

tr:TableAccess.uploadLimit.hard
: The value this limit cannot be raised above.

tr:TableAccess.uploadLimit.hard.unit
: The unit of the limit specified.

vg:Authority (!)
: A naming authority; an assertion of control over a namespace represented by an authority identifier.

vg:Authority.managingOrg (!)
: The organization that manages or owns this authority.

vg:Harvest.maxRecords
: The largest number of records that the registry search method will return. A value greater than one implies that an OAI continuation token will be provided when the limit is reached. A value of zero or less indicates that there is no explicit limit and thus, continuation tokens are not supported.

vg:Registry.managedAuthority (!)
: An authority identifier managed by a registry

vr:Organisation.facility (!)
: The observatory or facility used to collect the data contained or managed by this resource.

vr:Organisation.instrument (!)
: The Instrument used to collect the data contain or managed by a resource.

vs:DataCollection.accessURL (!)
: The URL that can be used to download the data contained in this data collection.

vs:DataCollection.coverage.footprint.ivoid (!)
: The URI form of the IVOA identifier for the service describing the capability refered to by this element.

vs:DataCollection.coverage.footprint (!)
: A reference to a footprint service for retrieving precise and up-to-date description of coverage.

vs:DataCollection.facility (!)
: The observatory or facility used to collect the data contained or managed by this resource.

vs:DataCollection.instrument (!)
: The Instrument used to collect the data contain or managed by a resource.

vs:DataService.coverage.footprint.ivoid (!)
: The URI form of the IVOA identifier for the service describing the capability refered to by this element.

vs:DataService.coverage.footprint (!)
: A reference to a footprint service for retrieving precise and up-to-date description of coverage.

vs:DataService.facility (!)
: The observatory or facility used to collect the data contained or managed by this resource.

vs:DataService.instrument (!)
: The Instrument used to collect the data contain or managed by a resource.

vs:Format.isMIMEType
: If `true`, then an accompanying `vs:Format` item is a MIME Type. Within res_details, this does not work for services that give more than one format; since furthermore the literal of vs:Format allows a good guess whether or not it is a MIME type, this does not appear a dramatic limitation.

vs:Format (!)
: The physical or digital manifestation of the information supported by a resource. MIME types should be used for network-retrievable, digital data, non-MIME type values are used for media that cannot be retrieved over the network.

vstd:EndorsedVersion (!)
: A version of a standard that is recommended for use.

vstd:Standard.deprecated (!)
: When present, this element indicates that all versions of the standard are considered deprecated by the publisher. The value is a human-readable explanation for the designation.

Note that within RegTAP tables, all utypes are stored lowercase only; the camel case notation above is for readability as well as easier relation to VOResource schema files (in which the same capitalization as given here is used for type and attribute names).

Also note that VOStandard `status` and `use` attributes cannot be represented here, and hence their values are not accessible in the relational registry. Similarly, complex TAPRegExt metadata on languages, user defined functions, and the like cannot be represented in this table. Since these pieces of metadata do not seem relevant to resource discovery, a more complex model does not seem warrented just so they can be exposed.

Name Utype	Type	Description
ivoid vr:Resource.identifier	adql:VARCHAR(*)	The parent resource.
cap_index N/A	adql:SMALLINT	The index of the parent capability; if NULL the utype-value pair describes a member of the entire resource.
detail_utype N/A	adql:VARCHAR(*)	The utype of the member
detail_value N/A	adql:VARCHAR(*)	(atomic) value of the member

The `ivoid` column should be an explicit foreign key into `resource`. It is recommended to maintain indexes on at least the columns `detail_utype` and `detail_value`.

The following columns MUST be lowercased during ingestion: `ivoid`, `detail_utype`. Clients are advised to use the `ivo_nocasematch` function to search in `detail_value` if the values have textual content.

8. ADQL User Defined Functions

TAP Servers implementing the `ivo://ivoa.net/std/RegTAP/vor` data model MUST implement the following three functions in their ADQL 2.0 language:

`ivo_nocasematch(value VARCHAR(*), pattern VARCHAR(*))->INTEGER`
: The function returns 1 if `pattern` matches `value` , 0 otherwise. `pattern` is defined as for the SQL LIKE operator, but the match is performed case-insensitively.

`ivo_hasword(haystack VARCHAR(*), needle VARCHAR(*)) -> INTEGER`
: The function takes two strings and returns 1 if the second is contained in the first one in a "word" sense, i.e., delimited by non-letter characters or the beginning or end of the string, where case is ignored. Addi-

tionally, servers MAY employ techniques to improve recall, in particular stemming. Registry clients must hence expect different results from different servers when using `ivo_hasword`; for such queries trying them on multiple registries may improve recall.

`ivo_hashlist_has(haslist VARCHAR(*), item VARCHAR(*))->INTEGER`

The function takes two strings; the first is a list of words not containing the hash sign (#), concatenated by hash signs, the second is a word not containing the hash sign. It returns 1 if, compared case-insensitively, the second argument is in the list of words encoded in the first argument. The behaviour for second arguments containing a hash sign is undefined.

Reference implementations of the three functions for the PostgreSQL data base system are given in Appendix B.

9. Common Queries to the Relational Registry

This section contains sample queries to the relational registry, mostly contributed as use cases by various members of the IVOA Registry working group. They are intended as an aid in designing relational registry queries, in particular for users new to the data model.

Find all TAP services; return their accessURLs

```
SELECT ivoid, access_url
FROM rr.capability
NATURAL JOIN rr.interface
WHERE standard_id='ivo://ivoa.net/std/tap'
```

Other `standard_id`s relevant include:

- `ivo://ivoa.net/std/registry` for OAI-PMH services,
- `ivo://ivoa.net/std/sia` for SIA services,
- `ivo://ivoa.net/std/conesearch` for SCS services, and
- `ivo://ivoa.net/std/ssa` for SSA services.

Find all SIA services that might have spiral galaxies

This is somewhat tricky since it is probably hard to image a part of the sky guaranteed not to have some, possibly distant, spiral galaxy in it. However, translating the intention into "find all SIA services that mention spiral in either the subject, the description, or the title", the query would become:

```
SELECT ivoid, access_url
FROM rr.capability
  NATURAL JOIN rr.resource
  NATURAL JOIN rr.interface
  NATURAL JOIN rr.subject
WHERE standard_id='ivo://ivoa.net/std/sia'
  AND (
    1=ivo_nocasematch(subject, '%spiral%')
    OR 1=ivo_hasword(res_description, 'spiral')
    OR 1=ivo_hasword(res_title, 'spiral'))
```

Find all SIA services that provides infrared images

```
SELECT ivoid, access_url
FROM rr.capability
  NATURAL JOIN rr.resource
  NATURAL JOIN rr.interface
WHERE standard_id='ivo://ivoa.net/std/sia'
  AND 1=ivo_hashlist_has('infrared', waveband)
```

Find all searchable catalogs that provide a column containing redshift

```
SELECT ivoid, access_url
FROM rr.capability
  NATURAL JOIN rr.table_column
  NATURAL JOIN rr.interface
WHERE standard_id='ivo://ivoa.net/std/conesearch'
  AND ucd='src.redshift'
```

Find all the resources published by a certain authority

```
SELECT ivoid
FROM rr.resource
WHERE ivoid LIKE 'ivo://org.gavo.dc%'
```

What registry records are there from a given publisher?

```
SELECT ivoid
FROM rr.resource
  NATURAL JOIN rr.res_role
WHERE 1=ivo_nocasematch(role_name, '%gavo%')
  AND base_role='publisher'
```

or, if the publisher actually gives its ivo-id in the registry records,

```
SELECT ivoid
FROM rr.resource
  NATURAL JOIN rr.res_role
WHERE role_ivoid='ivo://ned.ipac/ned'
  AND base_role='publisher'
```

What registry records are there originating from registry X?

This is using the CDS registry as an example. Note how this query builds conditions on values in `res_detail`:

```
SELECT ivoid FROM rr.resource
RIGHT OUTER JOIN (
  SELECT 'ivo://' || detail_value || '%' AS pat FROM rr.res_detail
  WHERE detail_utype='vg:registry.managedauthority'
    AND ivoid='ivo://cds.vizier/registry')
  AS authpatterns
ON (resource.ivoid LIKE authpatterns.pat)
```

Find all TAP endpoints offering the relational registry

```
SELECT access_url
FROM rr.interface
NATURAL JOIN rr.capability
NATURAL JOIN rr.res_detail
WHERE standard_id='ivo://ivoa.net/std/tap'
  AND detail_utype='tr:datamodeltype.ivoid'
  AND detail_value='ivo://ivoa.net/std/regtap/vor'
```

Find all TAP services exposing a table with certain features

"Certain features" could be "have some word in their description and having a column with a certain UCD". Then, a query could look like this:

```
SELECT ivoid, access_url, name, ucd, description
FROM rr.capability
  NATURAL JOIN rr.interface
  NATURAL JOIN rr.table_column
  NATURAL JOIN rr.res_table
```

```
WHERE standard_id='ivo://ivoa.net/std/tap'
  AND 1=ivo_hasword(table_description, 'quasar')
  AND ucd='phot.mag;em.opt.v'
```

Find all SSAP services that provide theoretical spectra

```
SELECT access_url
FROM rr.res_detail
  NATURAL JOIN rr.capability
  NATURAL JOIN rr.interface
WHERE detail_utype='ssap:datasource'
  AND standard_id='ivo://ivoa.net/std/ssa'
  AND detail_value='theory'
```

Find all ConeSearch services that provide stellar distance information

```
SELECT access_url, name, ucd, description
FROM rr.capability
  NATURAL JOIN rr.interface
  NATURAL JOIN rr.table_column
WHERE standard_id='ivo://ivoa.net/std/conesearch'
  AND ucd LIKE 'pos.parallax%'
```

Since there are multiple UCDs that might fit the concept of "stellar distance", one might need to give multiple UCDs here.

The service at http://dc.zah.uni-heidelberg.de/__system__/tap/run/tap is down, who can fix it?

```
SELECT DISTINCT base_role, role_name, email
FROM rr.res_role
  NATURAL JOIN rr.interface
WHERE access_url='http://dc.zah.uni-heidelberg.de/__system__/tap/run/tap'
```

A. A Stylesheet Generating Utypes

```
<stylesheet version="1.0" xmlns:xml="http://www.w3.org/XML/1998/namespace" xmlns:="http://
www.w3.org/1999/XSL/Transform" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:vr="http://
www.ivoa.net/xml/VOResource/v1.0" xmlns:vs="http://www.ivoa.net/xml/VODataService/v1.1"
xmlns:vm="http://www.ivoa.net/xml/VOMetadata/v0.1">
  <!-- extract utypes from VOResource and related XML schema files -->
  <!-- This program is free software: you can redistribute it and/or modify it under the terms
  of the GNU General Public License as published by the Free Software Foundation, either ver-
  sion 3 of the License, or (at your option) any later version. This program is distributed in
  the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warran-
  ty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
  for more details. For the complete text of the GPL, see http://www.gnu.org/licenses/. -->
  <!-- The basic strategy here is: start from all discernible types derived from vr:Resource,
  vr:Capability, and vr:Interface; we should catch everything in a VOResource extension even if
  we process file by file, unless one extension were to derive from a type defined in a differ-
  ent extension. In emulation of the utype generation done by VO-DML, we concatenate along axes
  that are simple attribuutes. When something is actually a collection, we start a new utype.
  The utype fragment is the *name* of a type or attribute; this means a certain amount of poly-
  morphism. In addition, for each schema, the prefix chosen for the target name space becomes
  the data model name for utype purposes; when, during inheritance, we leave a file, the mod-
  el name changes, too. Hack alert: We need to traverse the type tree; however, due to (practi-
  cal) limitations of XSLT1, we don't do that across files. So, if a type were to inherit from
  a class derived from VOResource or Capability in another document, this stylesheet would not
  notice. Also, we kill all namespace prefixes in attributes. Proper handling of that probably
  is close to impossible with XSLT1. -->
  <output method="text"/>
  <strip-space elements="*"/>
  <!-- root classes for utype generation; this must include all types that extensions derive
  from, since we don't see VOResource when processing them -->
  <variable name="UTYPEROOTS">+Resource+Interface+Capability+Service+DataType+</variable>
```

```
<!-- An index used to retrieve the type definitions for the child elements in walk; note that
this only spans one file, which is the primary limitation of this program -->
<key name="definitions" match="xs:simpleType|xs:complexType" use="@name"/>
<!-- types directly derived from another (the key is the name of the base type); we need this
to identify start points for our tree traversal. -->
<key name="derivations" match="xs:simpleType|xs:complexType" use="substring-af-
ter(./xs:complexContent/*/@base, ':')"/>
<template name="add-doc">
  <!-- retrieve the "short" doc and add them in parens if present, emit an lf either way -->
  <if test="boolean(xs:annotation[1])">
    <value-of select="concat(' (', normalize-space(xs:annotation[1]/xs:documentation), ')')"/
    >
  </if>
  <text> </text>
</template>
<template name="concat-dot">
  <!-- concatenate to parts with a dot unless there's another non-letter at the end of the
  first part -->
  <param name="first"/>
  <param name="second"/>
  <choose>
    <when test="substring($first, string-length($first), 1)=':'">
      <value-of select="$first"/>
      <value-of select="$second"/>
    </when>
    <otherwise>
      <value-of select="$first"/>
      .
      <value-of select="$second"/>
    </otherwise>
  </choose>
</template>
<template name="emit-utype-for-current">
  <!-- prints a utype for the current element -->
  <param name="parent-path"/>
  <call-template name="concat-dot">
    <with-param name="first" select="$parent-path"/>
    <with-param name="second" select="@name"/>
  </call-template>
  <call-template name="add-doc"/>
</template>
<template name="format-for-attribute">
  <!-- emits utypes for an within a type if the element is 0..1:1 -->
  <param name="parent-path"/>
  <!-- capability and interface are roots of their own -->
  <if test="@name!='capability' and @name!='interface'">
    <call-template name="emit-utype-for-current">
      <with-param name="parent-path" select="$parent-path"/>
    </call-template>
    <variable name="child-type" select="substring-after(@type, ':')"/>
    <if test="key('definitions', $child-type)">
      <variable name="child-path">
        <call-template name="concat-dot">
          <with-param name="first" select="$parent-path"/>
          <with-param name="second" select="@name"/>
        </call-template>
      </variable>
      <for-each select="key('definitions', $child-type)">
        <call-template name="walk">
          <with-param name="parent-path" select="$child-path"/>
        </call-template>
      </for-each>
    </if>
  </if>
</template>
<template name="walk">
```

```
  <!-- the central (recursive) template building up utypes by collecting subelements; the
  current node here is a type definition -->
  <param name="parent-path"/>
  <for-each select="descendant::xs:attribute">
    <call-template name="concat-dot">
      <with-param name="first" select="$parent-path"/>
      <with-param name="second" select="@name"/>
    </call-template>
    <call-template name="add-doc"/>
  </for-each>
  <for-each select="descendant::xs:element">
    <choose>
      <when test="@maxOccurs='unbounded'">
        <!-- emit a utype of the collection, then start a new utype hierarchy -->
        <call-template name="emit-utype-for-current">
          <with-param name="parent-path" select="$parent-path"/>
        </call-template>
        <variable name="new-root" select="concat(substring-before(@type, ':'), ':')"/>
        <for-each select="key('definitions', substring-after(@type, ':'))">
          <call-template name="format-for-type">
            <with-param name="parent-path" select="$new-root"/>
          </call-template>
        </for-each>
      </when>
      <otherwise>
        <call-template name="format-for-attribute">
          <with-param name="parent-path" select="$parent-path"/>
        </call-template>
      </otherwise>
    </choose>
  </for-each>
</template>
<template name="format-for-type">
  <!-- generates utypes for the current type and the classes it is derived from -->
  <param name="parent-path"/>
  <variable name="type-path">
    <call-template name="concat-dot">
      <with-param name="first" select="$parent-path"/>
      <with-param name="second" select="@name"/>
    </call-template>
  </variable>
  <value-of select="$type-path"/>
  <call-template name="add-doc"/>
  <call-template name="walk">
    <with-param name="parent-path" select="$type-path"/>
  </call-template>
  <call-template name="walk-in-hierarchy">
    <with-param name="base-type" select="@name"/>
    <with-param name="parent-path" select="$parent-path"/>
  </call-template>
</template>
<template name="walk-in-hierarchy">
  <!-- generates utypes for stuff derived from base-type -->
  <param name="base-type"/>
  <param name="parent-path"/>
  <for-each select="key('definitions', substring-after(./xs:complexContent|xs:simpleContent/
  */@base, ':'))">
    <call-template name="format-for-type">
      <with-param name="parent-path" select="$parent-path"/>
    </call-template>
  </for-each>
  <for-each select="key('derivations', @name)">
    <call-template name="format-for-type">
      <with-param name="parent-path" select="$parent-path"/>
    </call-template>
  </for-each>
</template>
```

```
  <template name="isUtypeRoot">
    <!-- generates true if the current type is eligible as a root of a utype hierarchy, nothing
    otherwise. -->
    <variable name="uqbase" select="substring-after(./xs:complexContent/*/@base, ':')"/>
    <!-- keep the explicit tests for the base since in VOResource extensions we won't see the
    base classes -->
    <choose>
      <when test="contains($UTYPEROOTS, concat('+', @name, '+')) or contains($UTYPEROOTS, con-
      cat('+', $uqbase, '+'))">true</when>
      <otherwise>
        <for-each select="key('definitions', $uqbase)">
          <call-template name="isUtypeRoot"/>
        </for-each>
      </otherwise>
    </choose>
  </template>
  <template match="xs:complexType">
    <!-- initiates a walk through the tree of child constructs if the current element is a re-
    source, capability, or interface, or directly derived from one of those -->
    <variable name="isRoot">
      <call-template name="isUtypeRoot"/>
    </variable>
    <if test="$isRoot='true'">
      <call-template name="format-for-type">
        <with-param name="parent-path" select="concat(//xs:appinfo/vm:targetPrefix, ':')"/>
      </call-template>
    </if>
  </template>
  <template match="/">
    <apply-templates/>
  </template>
  <template match="text()"/>
</stylesheet>
<!-- vi:et:sw=2:sta -->
```

B. The Extra UDFs in PL/pgSQL

What follows are implementations of the three user defined functions specificed in section 8 in the dialect the PostgreSQL server uses for SQL procedures, PL/pgSQL. For more details on PostgreSQL, see, e.g., *[doc:Postgres92]*.

```
CREATE OR REPLACE FUNCTION ivo_hasword(haystack TEXT, needle TEXT)
RETURNS INTEGER AS $func$
BEGIN
 IF to_tsvector(haystack) @@ plainto_tsquery(needle) THEN
  RETURN 1;
 ELSE
  RETURN 0;
 END IF;
END;
$func$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION ivo_hashlist_has(hashlist TEXT, item TEXT)
RETURNS INTEGER AS $func$
BEGIN
 -- postgres can't RE-escape a user string; hence, we'll have
 -- to work on the hashlist
 IF lower(item) = ANY(string_to_array(lower(hashlist), '#')) THEN
  RETURN 1;
 ELSE
  RETURN 0;
 END IF;
END;
$func$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION ivo_nocasematch(value TEXT, pattern TEXT)
RETURNS INTEGER AS $func$
BEGIN
 IF value ILIKE pattern THEN
  RETURN 1;
 ELSE
  RETURN 0;
 END IF;
END;
$func$ LANGUAGE plpgsql;
```

C. Implementation notes

This appendix contains a set of constraints and recommendations for implementing the relational registry model on actual RDBMSes; this concerns, in particular, minimum lengths for columns of type adql:VARCHAR. Implementations MUST NOT truncate strings of length equal to or smaller than the minimal lengths given here; the limitations are not, however, upper limits, and indeed, when choosing an implementation strategy it is in general preferable to not impose artificial length restrictions, in particular if no performance penalty is incurred.

These notes can also be useful with a view to preparing user interfaces for the relational registry, since input forms and similar user interface elements invariably have limited space; the limits here give reasonable defaults for the amount of data that should minimally be manipulatable by a user with reasonable effort.

These guidelines and rules discussed here originiate partly from an analysis of the data content of the VO registry in February 2013, partly from a consideration of limits in various XML schema documents.

The `ivoid` field present in every table of this specification merits special consideration, on the one hand due to its frequency, but also since other IVOA Identifiers present in the relational registry should probably be treated analoguously. Given that IVORNs in the 2013 data fields have a maximum length of roughly 100 characters, we propose a maximum length of 255 should be sufficient even when taking possible fragment identifiers into account.

Field type	Datatype suggested	Pertinent Fields
ivo-id	`VARCHAR(255)`	{all_table}.ivoid resource.footprint_ivoid (resource.harvested_from) - not standard? res_role.role_ivoid capability.standard_id relationship.related_id validation.validated_by

The relational registry also contains some date-time values. The most straightforward implementation certainly is to use SQL timestamps. Other relational registry fields that straightforwardly map to common SQL types are those that require numeric values, viz., `adql:REAL`, `adql:SMALLINT`, and `adql:INTEGER`. The following table summarizes these:

Field type	Datatype	Pertinent Fields
floating point	`adql:REAL`	resource.region_of_regard
small integer	`adql:SMALLINT`	capability.cap_index res_schema.schema_index res_table.schema_index table_column.std interface.cap_index

Field type	Datatype	Pertinent Fields
		interface.intf_index intf_param.cap_index intf_param.intf_index intf_param.std validation.level validation.cap_index res_detail.cap_index res_table.table_index table_column.table_index

The fields containing Utypes, UCDs, and Units are treated in parallel here. The 2013 registry data indicates a length of 128 characters is sufficient for real-world purposes; actually, at least UCDs and Units could of course grow without limitations, but it seems unreasonable anything longer than a typical line might actually be useful. As far as utypes are concerned, we expect those to shrink rather than grow with new standardization efforts.

Field type	Datatype suggested	Pertinent Fields
Utype(s) UCD(s) Units	VARCHAR(128)	res_schema.schema_utype res_table.table_utype table_column.ucd table_column.unit table_column.utype intf_param.ucd intf_param.unit intf_param.utype res_detail.detail_utype

The relational registry further has an e-mail field, for which we chose 128 characters as a reasonable upper limit (based on a real maximum of 40 characters in the 2013 data). There are furthermore URLs (in addition to access and reference URLs, there are also URLs for the WSDL of SOAP services and logos for roles). Due to the importance of in particular the access URLs we strongly recommend to use non-truncating types here. Empirically, there are access URLs of up to 224 characters in 2013 (reference URLs are less critical at a maximum of 96 characters). Expecting that with REST-based services, URL lengths will probably rather tend down than up, we still permit truncation at 255 characters.

Field type	Datatype suggested	Pertinent Fields
e-mail	VARCHAR(128)	res_role.email
URLs	VARCHAR(255)	resource.reference_url res_role.logo interface.wsdl_url interface.access_url

The next group of columns comprises those that have values taken from a controlled or finite vocabulary. Trying to simplify the view, lengths in the form of powers of two are considered.

Field type	Datatype suggested	Pertinent Fields
predefined values	VARCHAR(255)	resource.content_level resource.content_type
	VARCHAR(64)	resource.waveband
	VARCHAR(32)	resource.res_type resource.rights capability.cap_type

Field type	Datatype suggested	Pertinent Fields
		res_table.table_type table_column.flag table_column.datatype table_column.extended_schema table_column.extended_type table_column.type_system interface.result_type intf_param.datatype intf_param.extended_schema intf_param.extended_type
	VARCHAR(4)	interface.query_type interface.url_use

Finally, there are the fields that actually contain what is basically free text. For these, we have made a choice from reasonable powers of two lengths considering the actual lengths in the 2013 registry data. A special case are fields that either contain natural language text (the descriptions) or those that have grown without limit historically (resource.creator_seq, and, giving in to current abuses discussed above, res_role.role_name). For all such fields, no upper limit can sensibly be defined. However, since certain DBMSes (e.g.,MySQL older than version 5.6) cannot index fields with a TEXT datatype and thus using VARCHAR may be necessary at least for frequenly-searched fields, we give the maximal length of the fields in the 2013 registry in parentheses after the column designations for the TEXT datatype:

Field type	Datatype suggested	Interested Fields
free string values	TEXT	resource.res_description (7801) resource.creator_seq (712) res_role.role_name (712) res_schema.schema_description (934) res_table.table_description (934) table_column.description (3735) intf_param.description (347) capability.cap_description (100)
	VARCHAR(255)	resource.res_title res_role.address res_schema.schema_title res_table.table_title relationship.related_name res_detail.detail_value
	VARCHAR(128)	resource.version resource.source_value resource.facility resource.instrument subject.subject
	VARCHAR(64)	res_table.table_name table_column.name intf_param.name
	VARCHAR(32)	resource.source_format res_role.telephone res_schema.schema_name interface.intf_type interface.intf_role relationship_type

Field type	Datatype suggested	Interested Fields
		res_date.value_role
	VARCHAR(16)	resource.short_name table_column.arraysize interface.std_version intf_param.use

D. Changes from Previous Versions

D.1. Changes from WD-20121112

- Adapted all utypes to better match future VO-DML utypes.
- footprint, data_url, facility, and instrument are no longer in rr.resource but are instead kept in rr.res_details rows.
- For VOResource compliance, intf_param has no flag column any more.
- res_role.base_utype is renamed to res_role.base_role and no longer pretends to be a utype fragments; also, the content is now a simple word..
- intf_param.use is now called intf_param.use_param to avoid possible clashes with reserved SQL words.
- Removed all material on STC coverage.
- Added an appendix recommending field sizes.

References

[std:RI1] Kevin Benson, Ray Plante, Elizabeth Auden, Matthew Graham, Gretchen Greene, Martin Hill, Tony Linde, Dave Morris, Wil O'Mullane, Guy Rixon, Aurélien Stébé, and Kona Andrews, Kevin Benson and Ray Plante, editors.
IVOA registry interfaces version 1.0. IVOA Recommendation, 2009.

[std:RFC2119] S Bradner.
Key words for use in RFCs to indicate requirement levels. RFC 2119, March 1997.

[std:XMLNS] Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin, and Henry S. Thompson.
Namespaces in XML 1.0 (third edition). W3C Recommendation, December 2009.

[std:UNICODE] The Unicode Consortium.
The unicode standard, version 6.1 core specification, 2012.

[std:TAPREGEXT] Markus Demleitner, Patrick Dowler, Ray Plante, Guy Rixon, and Mark Taylor.
TAPRegExt: a VOResource schema extension for describing TAP services, version 1.0. IVOA Recommendation, August 2012.

[std:TAP] Patrick Dowler, Guy Rixon, and Doug Tody.
Table access protocol version 1.0. IVOA Recommendation, March 2010.

[std:STDREGEXT] Paul Harrison, Douglas Burke, Ray Plante, Guy Rixon, and Dave Morris.
StandardsRegExt: a VOResource schema extension for describing IVOA standards, version 1.0. IVOA Recommendation, May 2012.

[doc:Postgres92] http://www.postgresql.org/docs/9.2/static/index.html.
PostgreSQL 9.2.1 documentation. [Online].

[std:OBSCORE] Mireille Louys, Francois Bonnarel, David Schade, Patrick Dowler, Alberto Micol, Daniel Durand, Doug Tody, Laurent Michel, Jesus Salgado, Igor Chilingarian, Bruno Rino, Juan de Dios Santander, and Petr Skoda, Doug Tody, Alberto Micol, Daniel Durand, and Mireille Louys, editors.
Observation data model core components and its implementation in the Table Access Protocol, version 1.0. IVOA Recommendation, 2011.

[std:ADQL] Iñaki Ortiz, Jeff Lusted, Pat Dowler, Alexander Szalay, Yuji Shirasaki, Maria A. Nieto-Santisteba, Masatoshi Ohishi, William O'Mullane, Pedro Osuna, the VOQL-TEG, and the VOQL Working Group, Pedro Osuna and Iñaki Ortiz, editors.
Ivoa astronomical data query language. IVOA Recommendation, 2008.

[std:VOR] Raymond Plante, Kevin Benson, Matthew Graham, Gretchen Greene, Paul Harrison, Gerard Lemson, Tony Linde, Guy Rixon, and Aurélien Stébé.
VOResource: an XML encoding schema for resource metadata version 1.03. IVOA Recommendation, February 2008.
[std:DALREGEXT] Raymond Plante, Jesus Delago, Paul Harrison, and Doug Tody.
SimpleDALRegExt: Describing simple data access services, version 1.0. IVOA Proposed Recommendation, May 2012.
[std:VOID] Raymond Plante, Tony Linde, Roy Williams, and Keith Noddle.
IVOA identifiers, version 1.03. IVOA Recommendation, March 2007.
[std:VODS11] Raymond Plante, Aurélien Stébé, Kevin Benson, Patrick Dowler, Matthew Graham, Gretchen Greene, Paul Harrison, Gerard Lemson, Tony Linde, and Guy Rixon.
VODataService: a VOResource schema extension for describing collections and services version 1.1. IVOA Recommendation, December 2010.