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/.

Contents

1 Introduction 4
   1.1 Terminology ................................................. 4
   1.2 The Relational Registry within the VO Architecture .... 5

2 Design Considerations 6

3 Primary Keys 7

4 Notes on string handling 8
   4.1 Whitespace normalization ................................. 8
   4.2 NULL/Empty string normalization ......................... 8
   4.3 Case normalization ..................................... 9
   4.4 Non-ASCII characters ................................. 9

5 QNames in VOResource attributes 10

6 Xpaths 11

7 Discovering Relational Registries 12

8 VOResource Tables 13
   8.1 The resource Table ................................. 14
   8.2 The res_role Table ........................................... 18
   8.3 The res_subject Table ................................. 19
   8.4 The capability Table ................................. 20
   8.5 The res_schema Table ................................. 20
   8.6 The res_table Table ................................. 21
   8.7 The table_column Table ................................. 22
   8.8 The interface Table ................................. 24
   8.9 The intf_param Table ................................. 27
   8.10 The relationship Table ................................. 28
8.11 The validation Table ........................................... 28
8.12 The res_date Table .............................................. 29
8.13 The res_detail Table ............................................ 29
8.14 The alt_identifier Table ........................................ 31

9 ADQL User Defined Functions .......................... 31

10 Common Queries to the Relational Registry ...... 32
  10.1 TAP accessURLs .............................................. 32
  10.2 Image Services with Spirals ............................... 33
  10.3 Infrared Image Services .................................... 34
  10.4 Catalogs with Redshifts .................................... 34
  10.5 Names from an Authority .................................... 34
  10.6 Records Published by X .................................... 35
  10.7 Records from Registry ....................................... 35
  10.8 Locate RegTAP services ..................................... 36
  10.9 TAP with Physics ............................................ 36
  10.10 Theoretical SSA ............................................. 36
  10.11 Find Contact Persons ....................................... 37
  10.12 Related Capabilities ......................................... 37

A XPaths for res_details ...................................... 38

B The Extra UDFs in PL/pgSQL .............................. 42

C Implementation notes ........................................ 43

D XSLT to enumerate Relational Registry XPaths ...... 47

E Changes from Previous Versions ....................... 51
  E.1 Changes from REC-1.0 ......................................... 51
  E.2 Changes from PR-2014-10-30 ............................... 51
  E.3 Changes from PR-20140627 ................................... 51
  E.4 Changes from PR-20140227 ................................... 52
  E.5 Changes from WD-20131203 ................................... 52
  E.6 Changes from WD-20130909 ................................... 53
  E.7 Changes from WD-20130411 ................................... 53
  E.8 Changes from WD-20130305 ................................... 54
  E.9 Changes from WD-20121112 ................................... 54
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 about 20 publishing registries.

The protocol spoken by these publishing registries, OAI-PMH, only allows restricting queries by modification date and identifier and is hence not suitable for data discovery. 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 rich query languages. The IVOA Registry Interfaces specification (Benson et al., 2009) 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 (“searchable registry”) part of that specification (chapter 2), aimed towards usage with current VO standards, in particular TAP (Dowler et al., 2010) and ADQL (Ortiz et al., 2008). It follows the model of ObsCore (Louys et al., 2011) of defining a representation of a data model within a relational database. In this case, the data model is a simplification of the VO’s resource metadata interchange representation, the VOResource XML format (Plante et al., 2008). The simplification yields 13 tables. For each table, TAP_SCHEMA metadata is given together with rules for how to fill these tables from VOResource-serialized metadata records as well as conditions on foreign keys and recommendations on indexes.

The resulting set of tables has a modest size by today’s standards, but is still non-trivial. The largest table, table_column, has about half a million rows at the time of writing.

The architecture laid out here allows client applications to perform “canned” queries on behalf of their users as well as complex queries formulated directly by advanced users, using the same TAP clients they employ to query astronomical data servers.

1.1 Terminology

The set of tables and their metadata specified here, together with the mapping from VOResource (“ingestion rules”) is collectively called “relational registry schema” or “relational registry” for short.

The specification additionally talks about how to embed these into TAP services, gives additional user defined functions, talks about discovering com-
pliant services, etc. Since all this is tightly coupled to the “relational registry” as defined above, we do not introduce a new term for it. Hence, the entire standard is now known as “IVOA registry relational schema”.

Historically, we intended to follow the ObsCore/ObsTAP model and talked about RegTAP. As changing this acronym is technically painful (e.g., identifiers and URLs would need to be adapted), we kept it even after the distinction between the schema and its mapping on the one hand and its combination with a TAP service on the other went away. This means that the official acronym for “IVOA registry relational schema” is RegTAP. This aesthetic defect seems preferable to causing actual incompatibilities.

1.2 The Relational Registry within the VO Architecture

![IVOA Architecture diagram with the IVOA Registry Relational Specification (tagged with “Relational Registry”) and the related standards marked up.]

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

**VOResource, v1.03 (Plante et al., 2008)** 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 xpaths (Clark and DeRose, 1999).

---

5
**VODataService, v1.1 (Plante et al., 2010)** The VODataService standard describes several concepts and resource types extending VOResource’s data model, including tablesets, data services and data collections. These concepts and types are reflected in the database schema. Again xpaths 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 xpaths into the registry documents. This document should not in general need updates for registry extension updates. For completeness, we note the versions current as of this specification: SimpleDALRegExt 1.0 (Plante et al., 2012), StandardsRegExt 1.0 (Harrison et al., 2012), TAPRegExt 1.0 (Demleitner et al., 2012), Registry Interfaces 1.0 (Benson et al., 2009).

**TAP, v1.0 (Dowler et al., 2010)** The queries against the schema defined in the present document, 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 (Plante et al., 2007)** IVOA identifiers are essentially the primary keys within the VO registry; as such, they are actual primary keys of the central table of the relational 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 (Ortiz et al., 2008)** The rules for ingestion are designed to allow easy queries given the constraints of ADQL 2.0. Also, we give four functions that extend ADQL using the language’s built-in facility for user-defined functions.

## 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 (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, a DESCRIPTION element within a resource ends up in a column named res_description, whereas the same element from a CAPABILITY becomes cap_description.

We further renamed some columns and most tables with respect to their VOResource counterparts to avoid clashes with reserved words in popular database management systems. The alternatives would have been to either recommend quoting them or burden ADQL translation layers with the task of automatically converting them to delimited identifiers. Both alternatives seemed more confusing and less robust than the renaming proposed here.

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.

Another design goal of this specification has been that different registries operating on the same set of registry records will return identical responses for most queries; hence, we try to avoid relying on features left not defined by ADQL (e.g., the case sensitivity of string matches). However, with a view to non-uniform support for information retrieval-type queries in database systems, the ivo_hasword user defined function is not fully specified here; queries employing it may yield different results on different implementations, even if they operate on the same set of resource records.

3 Primary Keys

The primary key in the Registry as an abstract concept is a resource record’s IVORN. Hence, for all tables having primary keys at all, the ivoid column is part of its primary key. This specification does not require implementations to actually declare primary keys in the underlying database, and no aspect of user-visible behavior depends on such explicit declarations; in particular, this specification makes no requirements on the contents of tap_schema.keys.

We nevertheless make recommendations on explicit primary keys, as we expect definitions according to our recommendations will enhance robustness of services.

In several RegTAP tables – capability, res_schema, res_table, and interface – artificial primary keys are necessary, as in VOResource XML sibling elements are not otherwise distinguished. To allow such artificial primary keys, a column is added to each table, the name of which ends in _index (cap_index, schema_index, table_index, and intf_index).

The type and content of these X_index columns is implementation-defined, and clients must not make assumptions on their content except that
the pair ivo_id, X_index is a primary key for the relation (plus, of course, that references from other tables correctly resolve). In the tables of columns given below, the X_index columns have “(key)” given for type. Implementors obviously have to insert whatever ADQL type is appropriate for their choice or X_index implementation.

Obvious implementations for X_index include having X_index enumerate the sibling elements or using some sort of UUID.

4 Notes on string handling

In the interest of consistent behavior between different RegTAP implementations regardless of their technology choices, this section establishes some rules on the treatment of strings – both those obtained from attributes and those obtained from element content – during ingestion from VOResource XML to database tables.

4.1 Whitespace normalization

Most string-valued items in VOResource and extensions are of type xs:token, with the clear intent that whitespace in them is to be normalized in the sense of XML schema. For the few exceptions that actually are directly derived from xs:string (e.g., vstd:EndorsedVersion, vs:Waveband) it does not appear that the intent regarding whitespace is different.

In order to provide reliable querying and simple rules for ingestors even when these do not employ schema-aware XML parsers, this standard requires that during ingestion, leading and trailing whitespace MUST be removed from all strings; in particular, there are no strings consisting exclusively of whitespace in RegTAP. The treatment of internal whitespace is implementation-defined. This reflects the expectation that, wherever multi-word items are queried, whitespace-ignoring constraints will be used (e.g., LIKE-based regular expressions or the ivo_hasword user defined function defined below).

4.2 NULL/Empty string normalization

While empty strings and NULL values are not usually well distinguished in VO practice – as reflected in the conventional TABLEDATA and BINARY serializations of VOTable – , the distinction must be strictly maintained in the database tables to ensure reproduceable queries across different RegTAP implementations.

Ingestors therefore MUST turn empty strings (which, by section 4.1, include strings consisting of whitespace only in VOResource’s XML serialization) into NULL values in the database. Clients expressing constraints
on the presence (or absence) of some information must therefore do so using SQL’s IS NOT NULL (or IS NULL) operators.

### 4.3 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). 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 the Unicode Standard (The Unicode Consortium, 2012). 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 the Unicode Standard.

### 4.4 Non-ASCII characters

Neither TAP nor ADQL mention non-ASCII in service parameters – in particular the queries – or returned values. For RegTAP, that is unfortunate, as several columns will contain relevant non-ASCII characters. Columns for which extra care is necessary include all descriptions, res_title and creator_seq in rr.resource, as well as role_name and street_address in rr.res_role.

RegTAP implementations SHOULD be able to faithfully represent all characters defined in the latest version of the Unicode standard (The Unicode Consortium, 2012) at any given time and allow querying using them (having support for UTF-8 in the database should cover this requirement) for at least the fields mentioned above.

On VOResource ingestion, non-ASCII characters that a service cannot faithfully store MUST be replaced by a question mark character ("?").

RegTAP services MUST interpret incoming ADQL as encoded in UTF-8, again replacing unsupported characters with question marks.

We leave character replacement on result generation unspecified, as best-effort representations (e.g., “Angstrom” instead of “Ångström”) should not
impact interoperability but significantly improve user experience over consistent downgrading. In VOTable output, implementations SHOULD support full Unicode in at least the fields enumerated above. Clients are advised to retrieve results in VOTable or other encoding-aware formats.

Note that with VOTable 1.3, non-ASCII in char-typed fields, while supported by most clients in TABLEDATA serialization, is technically illegal; it is essentially undefined in other serializations. To produce standards-compliant VOTables, columns containing non-ASCII must be of type unicodeChar. We expect that future versions of VOTable will change the definitions of char and unicodeChar to better match modern standards and requirements. RegTAP implementors are encouraged to take these up.

5 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 employing prefixes declared using the xmlns mechanism as discussed in Bray et al. (2009). Within an ADQL-exposed 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/v1.0ConeSearch 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:
6 Xpaths

This specification piggybacks on top of the well-established VOResource standard. This means that it 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, i.e., VOResource and its extensions as defined by the XML Schema documents, and the fields within this database schema, is provided by xpaths, which are here slightly abbreviated for both brevity and generality.

All xpaths given in this specification are assumed to be relative to the enclosing vr:Resource element; these are called “resource xpaths” in the following. If resource xpaths are to be applied to an OAI-PMH response, the Xpath expression *//*/oai:metadata/ri:Resource must be prepended to it, with the canonical prefixes from section 5 implied. The resource xpaths themselves largely do not need explicit namespaces since VOResource elements are by default unqualified. Elements and attributes from non-VOResource schemata have the canonical namespace prefixes, which in this specification only applies to several xsi:type attribute names.

Some tables draw data from several different VOResource elements. For those, we have introduced an extended syntax with additional metacharacters (, ), and |, where the vertical bar denotes an alternative and the parentheses grouping. For instance, our notation /(tableset/schema/|)table/ corresponds to the two xpaths /table and /tableset/schema/table.

Within the Virtual Observatory, the link between data models and concrete data representations is usually made using utypes. Since VOResource is directly modelled in XML Schema, the choice of XPath as the bridging
formalism is compelling, though, and utypes themselves are not necessary for the operation of a TAP service containing the relational registry. TAP, however, offers fields for utypes in its TAP_SCHEMA. Since they are not otherwise required, this specification takes the liberty of using them to denote the xpaths.

In the metadata for tables and columns below, the utypes given are obtained from the xpaths by simply prepending them with xpath:. To avoid repetition, we allow relative xpaths: when the xpath in a column utype does not start with a slash, it is understood that it must be concatenated with the table utype to obtain the full xpath.

For illustration, if a table has a utype of

\[ \text{xpath}:/\text{capability}/\text{interface}/ \]

and a column within this table has a utype of

\[ \text{xpath}:/\text{accessURL}/@\text{use} \]

the resulting resource xpath would come out to be

\[ /\text{capability}/\text{interface}/\text{accessURL}/@\text{use} \]

to match this in an OAI-PMH response, the XPath would be

\[ */*/oai:metadata/ri:Resource/capability/interface/accessURL/@use. \]

While clients MUST NOT rely on these utypes in either TAP_SCHEMA or the metadata delivered with TAP replies, service operators SHOULD provide them, in particular when there are local extensions to the relational registry in their services. Giving xpaths for extra columns and tables helps human interpretation of them at least when the defining schema files are available.

Resource xpaths are also used in the res_details table (section 8.13). These are normal xpaths (although again understood relative to the enclosing Resource element), which, in particular, means that they are case sensitive. On the other hand, to clients they are simply opaque strings, i.e., clients cannot just search for any xpaths into VOResource within res_details.

7 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

\[ \text{ivo://ivoa.net/std/RegTAP#1.0} \]
in the service’s capabilities (cf. Demleitner et al. (2012)). Technically, this entails adding

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

as a child of the capability element with the type `tr:TableAccess`.

A client that knows the access URL of one TAP service containing a relational registry can thus discover all other services exposing one. The “Find all TAP endpoints offering the relational registry” example in section 10 shows a query that does this.

It is recommended to additionally register a relational registry as a VO-DataService data collection and connect this and the TAP services with a pair of service-for/served-by relations. This allows giving more metadata on the data content like, for example, the frequency of harvesting.

Services implementing this data model that do not (strive to) offer the full data content of the VO registry (like domain-specific registries or experimental systems) MUST NOT declare the above data model in order to not invite clients expecting the VO registry to send queries to it.

### 8 VOResource Tables

In the following table descriptions, the names of tables (cf. Table 1) and columns are normative and MUST be used as given, and all-lowercase. On the values in the `utype` columns within `TAP_SCHEMA`, see section 6. All columns defined in this document MUST have a 1 in the `std` column of the `TAP_SCHEMA.table_columns` table. Unless otherwise specified, all values of `ucd` and `unit` in `TAP_SCHEMA.table_columns` are NULL for columns defined here. 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 indexes and relationships. Finally, lowercasing requirements (normative) are given.
Figure 2: A sketch of the Relational Registry schema, adapted from Demleitner (2014). Only the columns considered most interesting for client use are shown. Arrows indicate foreign key-like relationships.

8.1 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 8.13). The `content_level`, `content_type`, and `waveband`, members are 1:n but still appear here. If there are multiple values, they are concatenated 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.
Name and UType | Description
--- | ---
rr.alt_identifier | An alternate identifier associated with this record.
 xpath:/(curation/creator/)altIdentifier
rr.capability | Pieces of behaviour of a resource.
 xpath:/capability/
rr.interface | Information on access modes of a capability.
 xpath:/capability/interface/
rr.intf_param | Input parameters for services.
 xpath:/capability/interface/param/
rr.registries | Administrative table: publishing registries we harvest, together with the dates of last full and incremental harvests.
 xpath://content/relationship/
rr.relationship | Relationships between resources (like mirroring, derivation, serving a data collection).
 xpath:/content/relationship/
rr.res_date | A date associated with an event in the life cycle of the resource.
 xpath:/curation/
rr.res_detail | XPath-value pairs for members of resource or capability and their derivations that are less used and/or from VOResource extensions.
 xpath:/content/
rr.res_role | Entities (persons or organizations) operating on resources: creators, contacts, publishers, contributors.
 xpath:/tableset/schema/
rr.res_schema | Sets of tables related to resources.
 rr.res_subject | Topics, object types, or other descriptive keywords about the resource.
 xpath:/content/
rr.res_table | (Relational) tables that are part of schemata or resources.
 xpath:/(tableset/schema/)table/
rr.resource | The resources (like services, data collections, organizations) present in this registry.
 xpath://
rr.table_column | Metadata on columns of a resource’s tables.
 xpath:/(tableset/schema/)table/column/
rr.validation | Validation levels for resources and capabilities.
 xpath:/(validation/)validationLevel

| Table 1: The tables making up the TAP data model Registry 1.0 |

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 lead to more or less arbitrary strings in that column, VOResource and its IVOA-recommended extensions at the time of writing define the following values for res_type:

- **vg:authority** A naming authority (these records allow resolving who is responsible for IVORNs with a certain authority; cf. Plante et al. (2007)).

- **vg:registry** A registry is considered a publishing registry if it contains a capability element with xsi:type="vg:Harvest". Old, RegistryInter-
face 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 7 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 an 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.

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

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 /accessURL in 8.13.

datastandard 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 metadata on active records is being kept in the relational registry. In other words, users can expect a resource to exist and work if they find it in a relational registry.
**Column names, utypes, ADQL types, and descriptions for the rr.resource table**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Utype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivooid</td>
<td>char(*)</td>
<td>Unambiguous reference to the resource conforming to the IVOA standard for identifiers.</td>
</tr>
<tr>
<td>xpath:identifier</td>
<td>char(*)</td>
<td>Resource type (something like vs:datadecollection, vs:catalogservice, etc).</td>
</tr>
<tr>
<td>res_type</td>
<td>char(*)</td>
<td>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.</td>
</tr>
<tr>
<td>created</td>
<td>char(*)</td>
<td>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 it to refer to the resource in a compact display. One word or a few letters is recommended. No more than sixteen characters are allowed.</td>
</tr>
<tr>
<td>short_name</td>
<td>char(*)</td>
<td>The full name given to the resource.</td>
</tr>
<tr>
<td>xpath:shortName</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>content_level</td>
<td>char(*)</td>
<td>A hash-separated list of content levels specifying the intended audience.</td>
</tr>
<tr>
<td>res_description</td>
<td>unicodeChar(*)</td>
<td>An account of the nature of the resource.</td>
</tr>
<tr>
<td>reference_url</td>
<td>char(*)</td>
<td>URL pointing to a human-readable document describing this resource.</td>
</tr>
<tr>
<td>creator_seq</td>
<td>unicodeChar(*)</td>
<td>The creator(s) of the resource in the order given by the resource record author, separated by semicolons.</td>
</tr>
<tr>
<td>content_type</td>
<td>char(*)</td>
<td>A hash-separated list of natures or genres of the content of the resource.</td>
</tr>
<tr>
<td>source_format</td>
<td>char(*)</td>
<td>The format of source_value. Recognized values include &quot;bibcode&quot;, referring to a standard astronomical bibcode (<a href="http://cdsweb.u-strasbg.fr/simbad/refcode.html">http://cdsweb.u-strasbg.fr/simbad/refcode.html</a>).</td>
</tr>
<tr>
<td>source_value</td>
<td>unicodeChar(*)</td>
<td>A bibliographic reference from which the present resource is derived or extracted.</td>
</tr>
<tr>
<td>res_version</td>
<td>char(*)</td>
<td>Label associated with creation or availability of a version of a resource.</td>
</tr>
<tr>
<td>region_of_regard</td>
<td>float(1)</td>
<td>A single numeric value representing the angle, given in decimal degrees, by which a positional query against this resource should be &quot;blurred&quot; in order to get an appropriate match.</td>
</tr>
<tr>
<td>waveband</td>
<td>char(*)</td>
<td>A hash-separated list of regions of the electro-magnetic spectrum that the resource’s spectral coverage overlaps with.</td>
</tr>
</tbody>
</table>
This table should have the **ivoid** column explicitly set as its primary key.

The following columns MUST be lowercased during ingestion: **ivoid**, **res_type**, **content_level**, **content_type**, **source_format**, **waveband**. 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**, **content_type**, and **waveband**.

The row for **region_of_regard** in TAP_SCHEMA.columns MUST have **deg** in its **unit** column.

### 8.2 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**, **contributor**, or **creator**. Depending on this value, here are the xpaths for the table fields (we have abbreviated /CURATION/PUBLISHER as **cp**, /CURATION/CONTACT as **co**, /CURATION/CREATOR as **cc**, and /CURATION/CONTRIBUTOR as **cb**):

<table>
<thead>
<tr>
<th><strong>base_role value</strong></th>
<th><strong>contact</strong></th>
<th><strong>publisher</strong></th>
<th><strong>creator</strong></th>
<th><strong>contributor</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>role_name</td>
<td>co/name</td>
<td>cp</td>
<td>cc/name</td>
<td>cb</td>
</tr>
<tr>
<td>role_ivoid</td>
<td>co/name/@ivo-id</td>
<td>cp/@ivo-id</td>
<td>cc/name/@ivo-id</td>
<td>cb/@ivo-id</td>
</tr>
<tr>
<td>address</td>
<td>co/address</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>email</td>
<td>co/email</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>telephone</td>
<td>co/telephone</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>logo</td>
<td>co/logo</td>
<td>N/A</td>
<td>cc/logo</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Not all columns are available for each role type in VOResource. For example, contacts have no logo, and creators no telephone members. Unavailable metadata (marked with **N/A** in the above table) 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_name**, you will have to use rather lenient regular expressions. Changing this, admittedly regrettable, situation would probably require a change in the VOResource schema.
### Column names, utypes, ADQL types, and descriptions for the rr.res_role table

<table>
<thead>
<tr>
<th>Column</th>
<th>ADQL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivoid</td>
<td>char(*)</td>
<td>The parent resource.</td>
</tr>
<tr>
<td>xpath:/identifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>role_name</td>
<td>unicodeChar(*)</td>
<td>The real-world name or title of a person or organization.</td>
</tr>
<tr>
<td>role_ivoid</td>
<td>char(*)</td>
<td>An IVOA identifier of a person or organization.</td>
</tr>
<tr>
<td>street_address</td>
<td>unicodeChar(*)</td>
<td>A mailing address for a person or organization.</td>
</tr>
<tr>
<td>email</td>
<td>char(*)</td>
<td>An email address the entity can be reached at.</td>
</tr>
<tr>
<td>telephone</td>
<td>char(*)</td>
<td>A telephone number the entity can be reached at.</td>
</tr>
<tr>
<td>logo</td>
<td>char(*)</td>
<td>URL pointing to a graphical logo, which may be used to help identify the entity.</td>
</tr>
<tr>
<td>base_role</td>
<td>char(*)</td>
<td>The role played by this entity; this is one of contact, publisher, and creator.</td>
</tr>
</tbody>
</table>

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_role. Clients are advised to query the remaining columns, in particular role_name, case-insensitively, e.g., using ivo_nocasematch.

### 8.3 The res_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.

### Column names, utypes, ADQL types, and descriptions for the rr.res_subject table

<table>
<thead>
<tr>
<th>Column</th>
<th>ADQL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivoid</td>
<td>char(*)</td>
<td>The parent resource.</td>
</tr>
<tr>
<td>xpath:/identifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>res_subject</td>
<td>char(*)</td>
<td>Topics, object types, or other descriptive keywords about the resource.</td>
</tr>
<tr>
<td>xpath:subject</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The ivoid column should be an explicit foreign key into resource. It is recommended to index the res_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 to query the res_subject column case-insensitively, e.g., using ivo_nocasematch.
8.4 The capability Table

The capability table describes a resource’s modes of interaction; it only contains the members of the base type vr:CAPABILITY. Members of derived types are kept in the res_detail table (see 8.13).

The table has a cap_index to disambiguate multiple capabilities on a single resource. See section 3 for details.

| Column names, utypes, ADQL types, and descriptions for the rr.capability table |
|----------------------------------|---------------------------------|-------------------------|
| ivoid                            | char(*)                         | The parent resource.    |
| xpath:/identifier                |                                 |                         |
| cap_index                        | short(1)                        | An arbitrary identifier of this capability within the resource. |
| cap_type                         | char(*)                         | The type of capability covered here. |
| xpath:@xsi:type                  |                                 |                         |
| cap_description                  | unicodeChar(*)                  | A human-readable description of what this capability provides as part of the over-all service. |
| xpath:descriptor                 |                                 |                         |
| standard_id                      | char(*)                         | A URI for a standard this capability conforms to. |
| xpath:@standardID               |                                 |                         |

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.

8.5 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 a column schema_index to disambiguate multiple schema elements on a single resource. See section 3 for details.
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.

### 8.6 The `res_table` Table

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

VODaService 1.0 had a similar element that was a direct child of resource. Ingestors should also accept such tables, as there are still numerous active VODaService 1.0 resources in the Registry at the time of writing (this is the reason for the alternative in the table xpath).

The table contains a column `table_index` to disambiguate multiple tables on a single resource. See section 3 for details. Note that if the sibling count is used as implementation of `table_index`, the count must be per resource and not per schema, as `table_index` MUST be unique within a resource.
<table>
<thead>
<tr>
<th>Column names, utypes, ADQL types, and descriptions for the \texttt{rr.res_table} table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivoid</td>
</tr>
<tr>
<td>xpath:/identifier</td>
</tr>
<tr>
<td>schema_index</td>
</tr>
<tr>
<td>table_description</td>
</tr>
<tr>
<td>table_index</td>
</tr>
<tr>
<td>table_name</td>
</tr>
<tr>
<td>table_title</td>
</tr>
<tr>
<td>table_type</td>
</tr>
<tr>
<td>table_utype</td>
</tr>
<tr>
<td>xpath:identifier</td>
</tr>
<tr>
<td>xpath:utype</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

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

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

### 8.7 The \texttt{table_column} Table

The \texttt{table_column} table models the content of VODatasetService’s \texttt{COLUMN} element. The table has been renamed to avoid a name clash with the SQL reserved word \texttt{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, ucd, utype, etc).

The \texttt{flag} column contains a concatenation of all values of a \texttt{COLUMN} element’s \texttt{FLAG} children, separated by hash characters. Use the \texttt{ivo_hashlist_has} function in queries against \texttt{flag}.

The \texttt{table_column} table also includes information from VODatasetService’s data type concept. VODatasetService 1.1 includes several type sys-
tems (VOTable, ADQL, Simple). The type_system 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.

<table>
<thead>
<tr>
<th>Column names, utypes, ADQL types, and descriptions for the rr:table_column table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivoid</td>
</tr>
<tr>
<td>xpath:/identifier</td>
</tr>
<tr>
<td>table_index</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>unit</td>
</tr>
<tr>
<td>utype</td>
</tr>
<tr>
<td>std</td>
</tr>
<tr>
<td>datatype</td>
</tr>
<tr>
<td>extended_schema</td>
</tr>
<tr>
<td>extended_type</td>
</tr>
<tr>
<td>arraysize</td>
</tr>
<tr>
<td>delim</td>
</tr>
<tr>
<td>type_system</td>
</tr>
<tr>
<td>flag</td>
</tr>
<tr>
<td>column_description</td>
</tr>
</tbody>
</table>

The pair ivoid, table_index should be an explicit foreign key into res_table. It is recommended to maintain indexes on at least the column_description, name, ucd, and utype columns, where the index on column_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. The boolean value of the col-
umn’s std attribute must be converted to 0 (False), 1 (True), or NULL (not given) on ingestion. 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.

8.8 The interface Table

The interface table subsumes both the vr:INTERFACE and vr:ACCESS-URL 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. VOResource 1.1 deprecated that capability (that was never really used in practice anyway) and replaced it with MIRRORURL. In the unlikely case multiple accessURL are defined in a single interface nevertheless, implementation behavior for a RegTAP service is undefined.

The table contains a column intf_index to disambiguate multiple interfaces of one resource. See section 3 for details.

In VOResource, interfaces can have zero or one SECURITYMETHOD children to convey support for authentication and authorization methods. Plain SECURITYMETHOD is an empty element that merely has a standardID attribute giving a URL of an authentication method required on a service\(^1\). In RegTAP, the value of this attribute is kept in the (somewhat irregularly-named) security_method_id column of the interface table. If an interface does not give a SECURITYMETHOD, this column MUST be NULL.

Clients not prepared to authenticate to services should always include a security_method_id IS NULL condition when retrieving access URLs from RegTAP 1.1 services. Other clients should retrieve the security_method_id together with access_url and compare the retrieved values against authentication methods they support.

The query_type column is a hash-joined list (analogous to waveband in the resource table), as the XML schema allows listing up to two request methods.

The mirror_url column is used to keep all mirror URLs in one field, again separating values with hash characters. This design was chosen over a native array since arrays of variable-length strings are not supported by VOTable, and emulating them is a major implementation liability. It was chosen over a separate database table implementing the 1:n relation because the hash – a fragment identifier in URIs, and access fragments are meaningless for access URLs – happens to be a safe and convenient separator for

\(^1\)In VOResource 1.0, multiple SECURITYMETHODs were allowed. No records using this should exist; RegTAP implementations should nevertheless be prepared to discard extra SECURITYMETHOD declarations.
the datatype, and thus there is no semantic cost attached to using an array emulation that is simpler on both client and server. Note that contrary to query_type and similar hash-joined lists of enumerated values, no case normalisation may take place in mirror_url.

This table only contains interface elements from within capabilities. Interface elements in StandardsRegExt records are ignored in the relational registry, and they must not be inserted in this table, since doing so would disturb the foreign key from interface into capability. In other words, the relational registry requires every interface to have a parent 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, usually with some form of structured parameters. This type is used for interfaces speaking standard IVOA protocols.

- 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.
### Column names, u types, ADQL types, and descriptions for the rr.interface table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivoid</td>
<td>char(*)</td>
<td>The parent resource.</td>
</tr>
<tr>
<td>xpath:/identifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cap_index</td>
<td>short(1)</td>
<td>The index of the parent capability.</td>
</tr>
<tr>
<td>intf_index</td>
<td>short(1)</td>
<td>An arbitrary identifier for the interfaces of a resource.</td>
</tr>
<tr>
<td>intf_type</td>
<td>char(*)</td>
<td>The type of the interface (vr:webbrowser, vs:paramhttp, etc).</td>
</tr>
<tr>
<td>xpath:@xsi:type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>intf_role</td>
<td>char(*)</td>
<td>An identifier for the role the interface plays in the particular capability.</td>
</tr>
<tr>
<td>xpath:@role</td>
<td></td>
<td></td>
</tr>
<tr>
<td>std_version</td>
<td>char(*)</td>
<td>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 referred to is the one identified by the Capability’s standardID element.</td>
</tr>
<tr>
<td>xpath:@version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>query_type</td>
<td>char(*)</td>
<td>Hash-joined list of expected HTTP method (get or post) supported by the service.</td>
</tr>
<tr>
<td>xpath:queryType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result_type</td>
<td>char(*)</td>
<td>The MIME type of a document returned in the HTTP response.</td>
</tr>
<tr>
<td>xpath:resultType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wsdl_url</td>
<td>char(*)</td>
<td>The location of the WSDL that describes this Web Service. If NULL, the location can be assumed to be the accessURL with ‘?wsdl’ appended.</td>
</tr>
<tr>
<td>xpath:wsdlURL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>url_use</td>
<td>char(*)</td>
<td>A flag indicating whether this should be interpreted as a base URL (‘base’), a full URL (‘full’), or a URL to a directory that will produce a listing of files (‘dir’). The URL at which the interface is found.</td>
</tr>
<tr>
<td>xpath:accessURL/@use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>access_url</td>
<td>char(*)</td>
<td>Secondary access URLs of this interface, separated by hash characters.</td>
</tr>
<tr>
<td>xpath:accessURL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mirror_url</td>
<td>char(*)</td>
<td>An identifier of an authentication method required on this interface, or NULL for interfaces publicly available. The identifiers of authentication schemes recommended in the VO are declared in the IVOA recommendation “SSO: Authentication Mechanisms.”</td>
</tr>
<tr>
<td>xpath:mirrorURL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>security_method_id</td>
<td>char(*)</td>
<td></td>
</tr>
<tr>
<td>xpath:securityMethod/@standardId</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This table should have the pair `ivoid, cap_index` as an explicit foreign key into `capability`, and the pair `ivoid, intf_index` as an explicit primary key. Additionally, 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`. Clients are advised to query `query_type` using the the `ivo_hashlist_has` function.
8.9 The intf_param Table

The intf_param table keeps information on the parameters available 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.

<table>
<thead>
<tr>
<th>Column names, utypes, ADQL types, and descriptions for the rr.intf_param table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivoid</td>
</tr>
<tr>
<td>intf_index</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>ucd</td>
</tr>
<tr>
<td>unit</td>
</tr>
<tr>
<td>utype</td>
</tr>
<tr>
<td>std</td>
</tr>
<tr>
<td>datatype</td>
</tr>
<tr>
<td>extended_schema</td>
</tr>
<tr>
<td>extended_type</td>
</tr>
<tr>
<td>arraysize</td>
</tr>
<tr>
<td>delim</td>
</tr>
<tr>
<td>param_use</td>
</tr>
<tr>
<td>param_description</td>
</tr>
</tbody>
</table>

The pair ivoid, intf_index should be an explicit foreign key into interface.

The remaining requirements and conventions are as per section 8.7 where applicable, and param_description taking the role of column_description.
8.10 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.

<table>
<thead>
<tr>
<th>Column names, utypes, ADQL types, and descriptions for the rr.relationship table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivoid</td>
</tr>
<tr>
<td>xpath:/identifier</td>
</tr>
<tr>
<td>relationship_type</td>
</tr>
<tr>
<td>xpath:relationshipType</td>
</tr>
<tr>
<td>related_id</td>
</tr>
<tr>
<td>xpath:relatedResource/@ivo-id</td>
</tr>
<tr>
<td>related_name</td>
</tr>
<tr>
<td>xpath:relatedResource</td>
</tr>
</tbody>
</table>

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

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

8.11 The validation Table

The validation table subsumes the vr:VALIDATIONLEVEL-typed members of both vr:RESOURCE and vr:CAPABILITY.

If the cap_index column is 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, it is valuable to gather validation results from various sources. Hence, harvesters for the relational registry may choose to obtain validation data from the OAI-PMH endpoints of various registries by not harvesting just for the ivo_managed set and generate rr.validation rows from these records. This can trigger potentially problematic behavior when the original registry updates its resource record in that naive implementations will lose all third-party validation rows; this may actually be the correct behavior, since an update of the registry record might very well indicate validation-relevant changes in the underlying services. Implementations are free to handle or ignore validation results as they see fit, and they may add validation results of their own.

The validation levels are defined in Hanisch et al. (2007) and currently range from 0 (description stored in a registry) to 4 (inspected by a human to be technically and scientifically correct).
Column names, utypes, ADQL types, and descriptions for the rr.validation table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Utype</th>
<th>ADQL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivoid</td>
<td>char(*)</td>
<td>short(1)</td>
<td>The parent resource.</td>
</tr>
<tr>
<td>xpath:/identifier</td>
<td>char(*)</td>
<td></td>
<td>The IVOA ID of the registry or organisation that assigned the validation level.</td>
</tr>
<tr>
<td>validated_by</td>
<td>char(*)</td>
<td></td>
<td>A numeric grade describing the quality of the resource description, when applicable, to be used to indicate the confidence an end-user can put in the resource as part of a VO application or research study.</td>
</tr>
<tr>
<td>xpath:validationLevel/@validatedBy</td>
<td>char(*)</td>
<td></td>
<td>If non-NULL, the validation only refers to the capability referenced here.</td>
</tr>
</tbody>
</table>

The ivoid column 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.

8.12 The res_date Table

The res_date table contains information gathered from VR:Curation’s date children.

Column names, utypes, ADQL types, and descriptions for the rr.res_date table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Utype</th>
<th>ADQL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivoid</td>
<td>char(*)</td>
<td></td>
<td>The parent resource.</td>
</tr>
<tr>
<td>xpath:/identifier</td>
<td>char(*)</td>
<td></td>
<td>A date associated with an event in the life cycle of the resource.</td>
</tr>
<tr>
<td>date_value</td>
<td>char(*)</td>
<td></td>
<td>A string indicating what the date refers to, e.g., created, availability, updated.</td>
</tr>
</tbody>
</table>

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

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

8.13 The res_detail Table

The res_detail table is the relational registry’s primary means for extensibility as well as a fallback for less-used simple metadata. Conceptually, it stores triples of resource entity references, resource xpaths, and values, where resource entities can be resource records themselves or capabilities. Thus, metadata with values that are either string-valued or sets of strings 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 necessary as these are introduced. Instead, the extension itself simply defines new xpaths to be added in `res_detail`.

Some complex metadata – `tr:LANGUAGE_FEATURE` 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.

Appendix A gives a list of resource xpaths from the registry extensions that were recommendations at the time of writing. For the resource xpaths marked with an exclamation mark there, xpath/value pairs MUST be generated whenever the corresponding metadata items are given in a resource record. For the remaining resource xpaths, these pairs should be provided if convenient; they mostly concern test queries and other curation-type information that, while unlikely to be useful to normal users, may be relevant to curation-type clients that, e.g., ascertain the continued operation of services.

Some detail values must be interpreted case-insensitively; this concerns, in particular, IVORNs like the TAP data model type. For other rows – the test queries are immediate examples –, changing the case will likely break the data. In order to avoid having to give and implement case normalization rules by detail xpath, no case normalization is done on detail values at all, and users and clients will have to use the `ivo_nocasematch` user defined function (see section 9) when locating case-insensitive values. For the resource xpaths given in Appendix A, this concerns all items with xpaths ending in `@ivo-id`.

Individual ingestors MAY choose to expose additional metadata using other utypes, provided they are formed according to the rules in section 6 (this rule is intended to minimize the risk of later clashes).

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

| Column names, utypes, ADQL types, and descriptions for the `rr.res_detail` table |
|---------------------------------|---------------------------------|--------------------------------------------------|
| ivoid                           | char(*)                         | The parent resource.                             |
| xpath:/identifier               | char(*)                         | The index of the parent capability; if NULL the xpath-value pair describes a member of the entire resource. |
| cap_index                       | short(1)                        | The xpath of the data item.                      |
| detail_xpath                    | char(*)                         | (Atomic) value of the member.                    |
| detail_value                    | unicodeChar(*)                  |                                                  |

The `ivoid` column should be an explicit foreign key into `resource`. It is recommended to maintain indexes on at least the columns `detail_xpath` and `detail_value`, where the index on `detail_value` should ideally work for both direct comparisons and searches using the `ivo_nocasematch` function.

The following column MUST be lowercased during ingestion: `ivoid`. 30
Clients are advised to use the \texttt{ivo_nocasematch} function to search in detail value if the values are to be compared case-insensitively (e.g., all IVORNs).

8.14 The alt_identifier Table

Since its version 1.1, VOResource allows the annotation of various elements (initially, the record itself and creators) with alternate identifiers (the \texttt{ALTIDENTIFIER} element). Examples of these are DOIs, ORCIDs, and bibcodes.

Considering that that the typical query against the alternate identifiers can be expected to be of the type “records having to do with identifier” and since the different identifier types cannot not clash since they are stored in URI form, RegTAP does not keep track where an alternate identifier was encountered. Instead, the alt_identifier table just links IVOIDs and alternate identifiers:

<table>
<thead>
<tr>
<th>Column names, utypes, ADQL types, and descriptions for the \texttt{rr.alt_identifier} table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivoid</td>
</tr>
<tr>
<td>xpath:/identifier</td>
</tr>
</tbody>
</table>

The \texttt{ivoid} column should be an explicit foreign key into resource. It is recommended to maintain an index on the alt_identifier column.

9 ADQL User Defined Functions

TAP Servers implementing the \texttt{ivo://ivoa.net/std/RegTAP#1.0} data model MUST implement the following four functions in their ADQL 2.0 language, with signatures written as recommended in Demleitner et al. (2012):

\begin{verbatim}
ivo_nocasematch(value VARCHAR(*), pat VARCHAR(*)) -> INTEGER
\end{verbatim}

The function returns 1 if \texttt{pat} matches \texttt{value}, 0 otherwise. \texttt{pat} is defined as for the SQL LIKE operator, but the match is performed case-insensitively.

\begin{verbatim}
ivo_hasword(haystack VARCHAR(*), needle VARCHAR(*)) -> INTEGER
\end{verbatim}

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. Additionally, servers MAY employ techniques to improve recall, in particular stemming. Registry clients must hence expect different results from different servers when using \texttt{ivo_hasword}; for such queries trying them on multiple registries may improve recall.
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 behavior for second arguments containing a hash sign is undefined.

An aggregate function returning all values of \texttt{expr} within a GROUP concatenated with \texttt{delim}. NULLs in the aggregate do not contribute, an empty aggregate yields an empty string.

Reference implementations of the four functions for the PostgreSQL database system are given in Appendix B.

10 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.

When locating access URLs for capabilities of standard services, these sample queries look for interfaces of type \texttt{vs:ParamHTTP} within the embedding capabilities. This is not strictly as intended by VOResource, which has the special \texttt{role} attribute to mark the interface on which a standard protocol is exposed within a capability. In RegTAP, this method of locating standard interfaces would be effected by constraining \texttt{intf\_role LIKE 'std%'}.

In actual VO practice, too many standard interfaces are lacking a proper declaration of their role, and actual clients locate the standard interfaces as given here. Following widespread practice client designers are encouraged to compare against the interface types rather than rely on \texttt{INTERFACE/@ROLE}, and resource record authors should make sure clients can discover standard interfaces both by the interfaces’ roles and types.

10.1 TAP accessURLs

\textbf{Problem:} Find all TAP services; return their accessURLs

As the capability type is in \texttt{rr\_capability}, whereas the access URL can be found from \texttt{rr\_interface}, this requires a (natural) join.

Clients communicating with a RegTAP 1.1 service should request the new \texttt{security\_method\_id} column. If this is non-null, the service requires some sort of authentication and should only presented to users if a client has the necessary infrastructure for the authentication system.
Clients only interested in services not requiring authentication should now use

SELECT ivoid, access_url
FROM rr.capability
NATURAL JOIN rr.interface
WHERE standard_id like 'ivo://ivoa.net/std/tap%'
   AND intf_type='vs:paramhttp'

Analogous considerations apply to the other example queries

Other standard_ids relevant here 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.

10.2 Image Services with Spirals

Problem: 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 (from rr.res_subject), the description, or the title (which are in rr.resource)”, the query would become:

SELECT ivoid, access_url, security_method_id
FROM rr.capability
   NATURAL JOIN rr.resource
   NATURAL JOIN rr.interface
   NATURAL JOIN rr.res_subject
WHERE standard_id='ivo://ivoa.net/std/sia'
   AND intf_type='vs:paramhttp'
   AND (1=ivo_nocasematch(res_subject, '%spiral%')
        OR 1=ivo_hasword(res_description, 'spiral')
        OR 1=ivo_hasword(res_title, 'spiral'))
10.3 Infrared Image Services

**Problem:** Find all SIA services that provide infrared images

The waveband information in `rr.resource` comes in hash-separated atoms (which can be radio, millimeter, infrared, optical, uv, euv, x-ray, or gamma-ray). For matching those, use the `ivo_hashlist_has` function as below. The access URL and the service standard come from `rr.interface` and `rr.capability`, respectively.

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

10.4 Catalogs with Redshifts

**Problem:** Find all searchable catalogs (i.e., cone search services) that provide a column containing redshifts

Metadata on columns exposed by a service are contained in `rr.table_column`. Again, this table can be naturally joined with `rr.capability` and `rr.interface`:

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

Sometimes you want to match a whole set of ucds. Frequently the simple regular expressions of SQL will help, as in `AND ucd LIKE 'pos.parallax%'`. When that is not enough, use boolean OR expressions.

10.5 Names from an Authority

**Problem:** Find all the resources published by a certain authority

An “authority” within the VO is something that hands out identifiers. You can tell what authority a record came from by looking at the “host part” of the IVO identifier, most naturally obtained from `rr.resource`. Since ADQL cannot actually parse URIs, we make do with simple string matching:
SELECT ivoid FROM rr.resource WHERE ivoid LIKE 'ivo://org.gavo.dc%'

10.6 Records Published by X

**Problem:** What registry records are there from a given publisher?

This uses the `rr.res_role` table both to match names (in this case, a publisher that has “gavo” in its name) and to ascertain the named entity actually publishes the resource (rather than, e.g., just being the contact in case of trouble). The result is a list of ivoids in this case. You could join this with any other table in the relational registry to find out more about these services.

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

10.7 Records from Registry

**Problem:** What registry records are there originating from registry X?

This is mainly a query interesting for registry maintainers. Still, it is a nice example for joining with the `rr.res_detail` table, in this case to first get a list of all authorities managed by the CDS registry.

```
SELECT ivoid FROM rr.resource
RIGHT OUTER JOIN (SELECT 'ivo://' || detail_value || '%' AS pat
    FROM rr.res_detail
    WHERE detail_xpath='/managedAuthority'
        AND ivoid='ivo://cds.vizier/registry') AS authpatterns
ON (1=ivo_nocasematch(resource.ivoid, authpatterns.pat))
```
10.8 Locate RegTAP services

**Problem:** Find all TAP endpoints offering the relational registry

This is the discovery query for RegTAP services themselves; note how this combines `rr.res_detail` pairs with `rr.capability` and `rr.interface` to locate the desired protocol endpoints. As clients should not usually be concerned with minor versions of protocols unless they rely on additions made in later versions, this query will return endpoints supporting “version 1” rather than exactly version 1.0.

```
SELECT access_url
FROM rr.interface
NATURAL JOIN rr.capability
NATURAL JOIN rr.res_detail
WHERE standard_id='ivo://ivoa.net/std/tap'
  AND intf_type='vs:paramhttp'
  AND detail_xpath='/capability/dataModel/@ivo-id'
  AND 1=ivo_nocasematch(detail_value, 'ivo://ivoa.net/std/regtap#1.%')
  AND security_method_id IS NULL
```

10.9 TAP with Physics

**Problem:** 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”. Either way, this kind of query fairly invariably combines the usual `rr.capability` and `rr.interface` for service location with `rr.table_column` for the column metadata and `rr.res_table` for properties of tables.

```
SELECT ivoid, name, ucd, column_description, access_url, security_method_id
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 intf_type='vs:paramhttp'
  AND 1=ivo_hasword(table_description, 'quasar')
  AND ucd='phot.mag;em.opt.v'
```

10.10 Theoretical SSA

**Problem:** Find all SSAP services that provide theoretical spectra.
The metadata required to solve this problem is found in the SSAP registry extension and is thus kept in `rr.res_detail`:

```sql
SELECT access_url, security_method_id
FROM rr.res_detail
  NATURAL JOIN rr.capability
  NATURAL JOIN rr.interface
WHERE detail_xpath='/capability/dataSource'
  AND intf_type='vs:paramhttp'
  AND standard_id='ivo://ivoa.net/std/ssa'
  AND detail_value='theory'
```

### 10.11 Find Contact Persons

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

This uses the `rr.res_role` table and returns all information on it based on the IVORN of a service that in turn was obtained from `rr.interface`. You could restrict to the actual technical contact person by requiring `base_role='contact'`.

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

### 10.12 Related Capabilities

**Problem:** Get the capabilities of all services serving a specific resource (typically, a data collection).

In the VO, data providers can register pure data collections without access options (or just furnished with a link to a download). They can then declare that their data can be “served-by” some other resource, typically a TAP service or some collective service for a number of instruments. To locate the access options to the data itself, inspect `rr.relationship` and use it to select records from `rr.capability`.

```sql
SELECT *
FROM rr.relationship AS a
  JOIN rr.capability AS b
  ON (a.related_id=b.ivoid)
WHERE relationship_type='IsServiceFor'
```
A XPaths for res_details

This appendix defines the res_details table (see section 8.13 for details) by giving xpaths for which xpath/value pairs MUST (where marked with an exclamation mark) or SHOULD be given if the corresponding data is present in the resource records. This list is normative for metadata defined in IVOA recommendations current as of the publication of this document (see section 1.2). As laid down in section 8.13, new VOResource extensions or new versions of existing VOResource extensions may amend this list.

In case there are conflicts between this list and xpaths derived from schema files using the rules given in section 6, the conflict must be considered due to an editorial oversight in the preparation of this list, and the xpaths from the schema files are normative. Errata to this list will be issued in such cases.

The xpaths are sufficient for locating the respective metadata as per section 6. With the explanations we give the canonical prefixes for the XML namespaces from which the items originate, which is where further information can be found.

/accessURL (!) For data collection resources, this is the URL that can be used to download the data contained. Do not enter accessURLs from interfaces into res_detail (vs).

/capability/executionDuration/hard The hard run time limit, given in seconds (tr).

/capability/complianceLevel The category indicating the level to which this instance complies with the SSA standard (ssap).

/capability/creationType (!) The category that describes the process used to produce the dataset; one of archival, cutout, filtered, mosaic, projection, specialExtraction, catalogExtraction (ssap).

/capability/dataModel (!) The short, human-readable name of a data model supported by a TAP service; for most applications, clients should rather constrain /capability/dataModel/@ivo-id (tr).

/capability/dataModel/@ivo-id (!) The IVORN of the data model supported by a TAP service (tr).

/capability/dataSource (!) The category specifying where the data originally came from; one of survey, pointed, custom, theory, artificial (ssap).

/capability/defaultMaxRecords (!) The largest number of records that the service will return when the MAXREC parameter is not specified in the query input (ssap).
/capability/executionDuration/default The run time limit for newly-created jobs, given in seconds (tr).

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

/capability/interface/testQueryString A query string that can be used to validate one of the interfaces of a capability (vr).

/capability/language/name (!) A short, human-readable name of a language understood by the TAP service (tr).

/capability/language/version/@ivo-id (!) The IVORN of a language supported by a TAP service (tr).

/capability/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).

/capability/maxFileSize (!) The maximum image file size in bytes (sia).

/capability/maxImageExtent/lat The maximum size in the latitude (Dec.) direction (sia).

/capability/maxImageExtent/long The maximum size in the longitude (R.A.) direction (sia).

/capability/maxImageSize/lat The image size in the latitude (Dec.) direction in pixels (sia-1.0).

/capability/maxImageSize/long The image size in the longitude (R.A.) direction in pixels (sia-1.0).

/capability/maxImageSize A measure of the largest image the service can produce given as the maximum number of pixels along the first or second axes. (sia).

/capability/maxQueryRegionSize/lat The maximum size in the latitude (Dec.) direction (sia).

/capability/maxQueryRegionSize/long The maximum size in the longitude (R.A.) direction (sia).

/capability/maxRecords (!) The largest number of items (records, rows, etc.) that the service will return (cs, sia, vg, ssap).

/capability/maxSearchRadius (!) The largest search radius, in degrees, that will be accepted by the service without returning an error condition. Not providing this element or specifying a value of 180 indicates that there is no restriction. (ssap)
/capability/maxSR (!) The largest search radius of a cone search service (cs).
/capability/outputFormat/@ivo-id (!) An IVORN of an output format (tr).
/capability/outputFormat/alias A short, mnemonic identifier for a service’s output format (tr).
/capability/outputFormat/mime (!) The MIME type of an output format (tr).
/capability/outputLimit/default The maximal output size for newly-created jobs (tr).
/capability/outputLimit/default/@unit The unit (rows/bytes) in which the service’s default output limit is given (tr).
/capability/outputLimit/hard The hard limit of a service’s output size (tr).
/capability/outputLimit/hard/@unit The unit of this service’s hard output limit (tr).
/capability/retentionPeriod/default The default time between job creation and removal on this service, given in seconds (tr).
/capability/retentionPeriod/hard The hard limit for the retention time of jobs on this services (tr).
/capability/supportedFrame (!) The STC name for a world coordinate system frame supported by this service (ssap).
/capability/testQuery/catalog The catalog to query (cs).
/capability/testQuery/dec Declination in a test query (cs)
/capability/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, sia).
/capability/testQuery/pos/lat The Declination of the center of the search position in decimal degrees (ssap, sia).
/capability/testQuery/pos/long The Right Ascension of the center of the search position in decimal degrees (ssap, sia).
/capability/testQuery/pos/refframe A coordinate system reference frame name for a test query. If not provided, ICRS is assumed (ssap).
/capability/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 (ssap).
/capability/testQuery/ra Right ascension in a test query (cs).
/capability/testQuery/size The size of the search radius in an SSA search query (ssap).
/capability/testQuery/size/lat Region size for a SIA test query in declination (sia).
/capability/testQuery/size/long Region size for a SIA test query in RA (sia).
/capability/testQuery/sr Search radius of a cone search service’s test query (cs).
/capability/testQuery/verb Verbosity of a service’s test query (cs, sia).
/capability/uploadLimit/default An advisory size above which user agents should reconfirm uploads to this service (tr).
/capability/uploadLimit/default/@unit The unit of the limit specified (tr).
/capability/uploadLimit/hard Hard limit for the size of uploads on this service (tr).
/capability/uploadLimit/hard/@unit The unit of the limit specified (tr).
/capability/uploadMethod/@ivo-id The IVORN of an upload method supported by the service (tr).
/capability/verbosity (!) true if the service supports the VERB keyword; false, otherwise (cs).
/coverage/footprint (!) A URL of a footprint service for retrieving precise and up-to-date description of coverage (vs).
/coverage/footprint/@ivo-id (!) The URI form of the IVOA identifier for the service describing the capability refered to by this element (vs).
/deprecated (!) A sentinel that all versions of the referenced standard are deprecated. The value is a human-readable explanation for the designation (vstd).
/endorsedVersion (!) A version of a standard that is recommended for use (vstd).
/facility (!) The observatory or facility used to collect the data contained or managed by this resource (vs).
/format (!) The physical or digital manifestation of the information supported by a (DataCollection) 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 (vs).
/format/@isMIMEType If `true`, then an accompanying `/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).

/full If `true`, the registry attempts to collect all resource records known to the IVOA (vg).

/instrument (!) The instrument used to collect the data contained or managed by a resource (vr).

/instrument/@ivo-id (!) IVORN of the instrument used to collect the data contained or managed by a resource (vr).

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

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

/rights Free-text information on usage conditions for a resource (vr).

/rights/@rightsURI (!) A formal identifier for a license a resource is made available under (vr).

/schema/@namespace (!) An identifier for a schema described by a standard (vstd).

Note that the representation of StandardsRegExt’s `status` and `use` attributes as well as its `key` would require sequences of complex objects, which is impossible using `res_detail`. Hence, the respective metadata is not queriable within 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 and are geared towards other uses of the respective VOResource extensions, a more complex model does not seem warranted just so they can be exposed.

B The Extra UDFs in PL/pgSQL

What follows are (non-normative) implementations of three of the user defined functions specified in section 9 in the SQL dialect of PostgreSQL (e.g., Postgres Global Development Group (2013)).

Note that PostgreSQL cannot use fulltext indexes on the respective columns if the functions are defined in this way for (fairly subtle) reasons connected with NULL value handling. While workarounds are conceivable,
they come with potentially unwelcome side effects, at least as of PostgreSQL
9.x. It is therefore recommended to replace expressions involving the func-
tions given here with simple boolean expressions in the ADQL translation
layer whenever possible.

```sql
CREATE OR REPLACE FUNCTION
  ivo_hasword(haystack TEXT, needle TEXT)
RETURNS INTEGER AS $func$
  SELECT CASE WHEN
to_tsvector($1) @@ plainto_tsquery($2)
    THEN 1
    ELSE 0
  END
$func$ LANGUAGE SQL;

CREATE OR REPLACE FUNCTION
  ivo_hashlist_has(hashlist TEXT, item TEXT)
RETURNS INTEGER AS $func$
  SELECT CASE WHEN lower($2) = ANY(string_to_array($1, '#'))
    THEN 1
    ELSE 0
  END
$func$ LANGUAGE SQL;

CREATE OR REPLACE FUNCTION
  ivo_nocasematch(value TEXT, pattern TEXT)
RETURNS INTEGER AS $func$
  SELECT CASE WHEN
    $1 ILIKE $2
    THEN 1
    ELSE 0
  END
$func$ LANGUAGE SQL;

CREATE OR REPLACE FUNCTION
  ivo_string_agg(value TEXT, item TEXT)
RETURNS INTEGER AS $func$
  SELECT CASE WHEN
    substring($1 FROM $2)
    THEN 1
    ELSE 0
  END
$func$ LANGUAGE SQL;
```

C Implementation notes

This appendix contains a set of constraints and recommendations for im-
plementing the relational registry model on actual RDBMSes, originating
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 docu-
mens. This concerns, in particular, minimum lengths for columns of type
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 strat-
ey 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.

The `ivo_id` 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 stored in the relational registry should probably be treated analogously. Given that IVORNs in the 2013 data fields have a maximum length of roughly 100 characters, we propose that a maximum length of 255 should be sufficient even when taking possible fragment identifiers into account.

<table>
<thead>
<tr>
<th>Field type</th>
<th>Datatype</th>
<th>Pertinent Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ivo-id</code></td>
<td><code>VARCHAR(255)</code></td>
<td><code>all_tables.ivoId</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>res_role.role_id</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>capability.standard_id</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>relationship.related_id</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>validation.validated_by</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>res_detail.detail_value</code> for several values of <code>detail_xpath</code></td>
</tr>
</tbody>
</table>

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., `REAL`, `SMALLINT`, and `INTEGER`. The following table summarizes these:

<table>
<thead>
<tr>
<th>Field type</th>
<th>Datatype</th>
<th>Pertinent Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating point</td>
<td><code>REAL</code></td>
<td><code>resource.region_of_regard</code></td>
</tr>
<tr>
<td>small integer</td>
<td><code>SMALLINT</code></td>
<td><code>table_column.std</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>intf_param.std</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>validation.level</code></td>
</tr>
</tbody>
</table>

The fields containing Utypes, UCDs, and Units are treated in parallel here. The 2013 registry data indicates that 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.

In this category, we also summarize our resource xpaths. With the conventions laid down in this document, it seems unlikely that future extensions to VOResource will be so deeply nested that 128 characters will not be sufficient for their resource xpaths.
<table>
<thead>
<tr>
<th>Field type</th>
<th>Datatype</th>
<th>Pertinent Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utype</td>
<td>VARCHAR(128)</td>
<td>res_schema.schema_utype</td>
</tr>
<tr>
<td></td>
<td></td>
<td>res_table.table_utype</td>
</tr>
<tr>
<td></td>
<td></td>
<td>table_column.utype</td>
</tr>
<tr>
<td></td>
<td></td>
<td>intf_param.utype</td>
</tr>
<tr>
<td>UCD</td>
<td>VARCHAR(128)</td>
<td>table_column.ucd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>intf_param.ucd</td>
</tr>
<tr>
<td>Unit</td>
<td>VARCHAR(128)</td>
<td>table_column.unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>intf_param.unit</td>
</tr>
<tr>
<td>xpath</td>
<td>VARCHAR(128)</td>
<td>res_detail.detail_xpath</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Field type</th>
<th>Datatype</th>
<th>Pertinent Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-mail</td>
<td>VARCHAR(128)</td>
<td>res_role.email</td>
</tr>
<tr>
<td>URLs</td>
<td>VARCHAR(255)</td>
<td>resource.reference_url</td>
</tr>
<tr>
<td></td>
<td></td>
<td>res_role.logo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interface.wsdl_url</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interface.access_url</td>
</tr>
</tbody>
</table>

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.
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
frequently-searched fields, we give the maximal length of the fields in the
2013 registry in parentheses after the column designations for the TEXT
datatype:
<table>
<thead>
<tr>
<th>Field type</th>
<th>Datatype</th>
<th>Pertinent Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>free string values</td>
<td>TEXT</td>
<td>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)</td>
</tr>
<tr>
<td>titles, etc.</td>
<td>VARCHAR(255)</td>
<td>resource.res_title res_role.address res_schema.schema_title res_table.table_title relationship.related_name res_detail.detail_value</td>
</tr>
<tr>
<td>expressions</td>
<td>VARCHAR(128)</td>
<td>resource.version resource.source_value res_subject.res_subject</td>
</tr>
<tr>
<td>names</td>
<td>VARCHAR(64)</td>
<td>res_table.table_name table_column.name intf_param.name</td>
</tr>
<tr>
<td>misc. short strings</td>
<td>VARCHAR(32)</td>
<td>resource.source_format res_role.telephone res_schema.schema_name interface.intf_type interface.intf_role relationship_type res_date.value_role</td>
</tr>
<tr>
<td>misc. particles</td>
<td>VARCHAR(16)</td>
<td>resource.short_name table_column.arraysize intf_param.arraysize interface.std_version intf_param.use</td>
</tr>
</tbody>
</table>

**D  XSLT to enumerate Relational Registry XPaths**

The (non-normative) following XSL stylesheet emits xpaths as discussed in section 6 when applied to a VOResource extension. Considering readability and limitations of XSLT, this is not fully general – if VOResource extensions started to inherit from other extensions’ subclasses of Resource, Capability,
or Interface, this stylesheet will need to be extended.

Still, it is a useful tool when evaluating how to map a given extension to the relational registry.

```xml
<?xml version="1.0" encoding="utf-8"?>
<stylesheet version="1.0"
 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">
<!−− extract RegTAP xpath from VOResource and related XML schema files. 
This file is in the public domain. −−>
<!−− The basic strategy here is: start from all discernible types 
derived from vr:Resource, vr:Capability, and vr:Interface; we need to 
handle capability and interface separately since they may not be 
explicitly reachable from a resource due to the way capability and 
interface types are declared in VOResource (i.e., through xsi:type ).

Each root is the start element for a recursion yielding the utypes. 
During this recursion, attributes yield utypes by concatenating the 
parent name with the attribute name, and elements yield utypes by 
concatenating the element name with the parent name. Elements take part 
in the recursion, where their utype becomes the new parent name.

Note that the xpaths generated here are the ones used in the 
res_details table. For the xpaths used in lieu of utypes in 
TAP_SCHEMA and VOSI tables, you will want to make them relative.

Hack alert: We need to traverse the type tree; however, due to 
(practical) limitations of XSLT1, we do not 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 
lose to impossible with XSLT1. −−>
</key>
<template name="add-doc">
</template>
</stylesheet>
```
<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">
    <value of select="concat($parent-path,'/@',@name)"/>
    <call template name="add-doc"/>
</for-each>
<for-each select="descendant::xs:element">
    <!-- capability and interface are roots of their own -->
    <if test="@name='capability'/and/@name='interface'">
        <value of select="concat($parent-path,'/',@name)"/>
        <call template name="add-doc"/>
        <variable name="child-type" select="substring-after(@type,':')"/>
        <if test="key('definitions',/usr$child-type')">
            <variable name="child-path" select="concat($parent-path,'/',@name)"/>
            <for-each select="key('definitions',/usr$child-type')">
                <call template name="walk">
                    <with param name="parent-path" select="$child-path">
                    </call template>
                </for-each>
            </if>
        </if>
    </if>
</for-each>
<!-- collect attributes and elements for our node from the types we are derived from, too -->
<for-each select="descendant::xs:extension|descendant::xs:restriction">
    <for-each select="key('definitions',substring-after(@base,':')')">
        <call template name="walk">
            <with param name="parent-path" select="$parent-path">
            </call template>
        </for-each>
    </for-each>
</for-each>
</template>
<template name="walk-for-type">
<!-- iterates over the types and elements derived from base-type -->
<param name="base-type"/>
<param name="parent-path"/>
</template>
<for-each select="key('of-type',@base-type)">
  <call-template name="walk">
    <with-param name="parent-path" select="$parent-path"/>
  </call-template>
</for-each>

<!-- templates cover the immediate types; the derived types are handled in explicit call-templates in the the template for root below. -->
<template match="/xs:complexType[@name='Resource']">
  <call-template name="walk">
    <with-param name="parent-path" select="Resource"/>
  </call-template>
</template>

<template match="/xs:complexType[@name='Capability']">
  <call-template name="walk">
    <with-param name="parent-path" select="Capability"/>
  </call-template>
</template>

<template match="/xs:complexType[@name='Interface']">
  <call-template name="walk">
    <with-param name="parent-path" select="Interface"/>
  </call-template>
</template>

<template match="/">
  <apply-templates/>
</template>

<call-template name="walk-for-type">
  <with-param name="base-type" select="Resource"/>
  <with-param name="parent-path" select="/Resource"/>
</call-template>

<call-template name="walk-for-type">
  <with-param name="base-type" select="Service"/>
  <with-param name="parent-path" select="/Service"/>
</call-template>

<call-template name="walk-for-type">
  <with-param name="base-type" select="Capability"/>
  <with-param name="parent-path" select="/Capability"/>
</call-template>

<call-template name="walk-for-type">
  <with-param name="base-type" select="Interface"/>
  <with-param name="parent-path" select="/Interface"/>
</call-template>
E Changes from Previous Versions

E.1 Changes from REC-1.0

- Added the `alt_identifier` table.
- Removed the `rights` column from the `resource` table (it’s now free text, so hash-separation doesn’t work any more), moved it and VOResource 1.1’s rightsURI to `res_detail`. This technically might constitute an API change, but since we don’t believe rights has been used anywhere, we believe we are still within a minor change.
- Added a `/capability/interface/testQueryString` xpath for `res_detail` to cover VOResource 1.1’s `TESTQUERYSTRING` interface child. Note that this is not really enough to feed validators, as a capability can have multiple interfaces and `res_detail` only tells apart capabilities. Consider alternative extra column in `interface` (but: `TESTQUERYSTRING` would have to be single).
- `SECURITYMETHOD/@standardID` is now included in interface in a `security_method_id` column. Consequently, the `/capability/interface/securityMethod/@standardID` `res_detail` key has been removed (but services can of course still provide it).
- Now recommending the evaluation of `security_method_id` in the discovery of standard interfaces.

E.2 Changes from PR-2014-10-30

- No changes to specification content (only minor typo fixes).

E.3 Changes from PR-20140627

- Removed reference to a future STC extension.
- Migrated to ivoatex.
E.4 Changes from PR-20140227

- Added a /full details xpath from VORegistry (this had been forgotten due to limitations in the makeutypes stylesheet).

- Added a /capability/interface/securityMethod/@standardID details xpath from vr:Interface.

- Added requirement to implement the ivo_string_agg user defined function.

- Added a section specifying the treatment of non-ASCII characters in RegTAP columns.

- New rules on string normalization: strings must be whitespace-stripped, empty strings must be mapped to NULL.

- Dropped requirements that the _index columns are integers (let alone small integers); added a section discussing in what sense they are implementation defined.

- Dropped adql: prefixes on TAP_SCHEMA.columns datatypes.

- Now declaring a precedence of xpaths generated by rules over the list in Appendix A.

- Clarified translation of column/@std and param/@std.

- Now recommending to constrain on intf_type (rather than intf_role, as before) when locating standard interfaces.

- Redactional changes from RFC (e.g., in column descriptions, some clarifications, typo fixes).

E.5 Changes from WD-20131203

- Changed the data model identifier to ivo://ivoa.net/std/RegTAP#1.0 to match usage with a later standards record.

- Fixed a typo in a column name: schema.schemaname is now schema.schema_name as in the prose.

- Recovered /capability/uploadMethod/@ivo-id res_detail keys that was accidentally lost in a previous version.

- Clarification of nomenclature.
E.6 Changes from WD-20130909

- Updates for REC of SimpleDALRegExt, which contains versions 1.1 of both the sia and the ssap XML schemas; this means there are now additional namespace URIs to take into account, as well as new res_detail xpaths /capability/maxSearchRadius, /capability/maxImageSize, and /capability/testQuery/pos/refframe.

- Reinstated makeutypes.xslt script; it’s useful even with the new xpaths.

E.7 Changes from WD-20130411

- The final utype reform: most of our ex-utype strings aren’t called utypes any more, they’re fairly plain xpaths. Consequently, res_detail.detail_utype has been renamed res_detail.detail_xpath.

- Renamed some columns and the subject table to relieve the need of quoting in MS SQL Server (or, in the case or use_param, maintain consistency after the renaming):

<table>
<thead>
<tr>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource.version</td>
<td>resource.res_version</td>
</tr>
<tr>
<td>res_role.address</td>
<td>res_role.street_address</td>
</tr>
<tr>
<td>subject.*</td>
<td>res_subject.*</td>
</tr>
<tr>
<td>res_subject.res_subject</td>
<td>res_subject.res_subject</td>
</tr>
<tr>
<td>table_column.description</td>
<td>table_column.column_description</td>
</tr>
<tr>
<td>intf_param.description</td>
<td>intf_param.param_description</td>
</tr>
<tr>
<td>intf_param.use_param</td>
<td>intf_param.param_use</td>
</tr>
<tr>
<td>validation.level</td>
<td>validation.val_level</td>
</tr>
</tbody>
</table>

- rr.intf_param grew the arraysize and delim columns that before accidentally were only present in rr.table_column.

- Added warnings about having to match case-insensitively in res_detail.detail_value for IVORN-valued rows.

- Restored the foreign key from interface to capability. Mandating ignoring interface elements from StandardsRegExt records really is the lesser evil.

- resource.region_of_regard now must have unit metadata declared.

- We now explicitly deprecate multiple access URLs per interface and announce that single access URLs will be enforced in future VOResource versions.
E.8 Changes from WD-20130305

- intf_index is now required to be unique within a resource, not a capability; this is because StandardsRegExt has interfaces outside of capabilities. In consequence, the intf_param no longer has a cap_index column, and its foreign key is just ivoid and intf_index.

- Added handling for the StandardsRegExt schema element.

- The list of res_details utypes was moved to an appendix since it was too long to be included in the running text.

- Redaction for WD publication.

E.9 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 fragment; 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


Clark, J. and DeRose, S. (1999), ‘Xml path language (xpath), version 1.0’, W3C Recommendation. URL: http://www.w3.org/TR/1999/REC-xpath-19991116
Demleitner, M. (2014), RegTAP – a new API to the VO Registry. Poster presented at ADASS XIV.


URL: http://www.ivoa.net/documents/REC/Identifiers/Identifiers-20070302.html


URL: http://www.ivoa.net/documents/VODataService/

URL: http://www.postgresql.org/docs/9.2/static/index.html

URL: http://www.unicode.org versions/Unicode6.1.0