International
Virtual
Observatory
Alliance

IVOA Registry Relational Schema

Version 1.2

IVOA Proposed Recommendation 2024-02-27

Working Group
Registry

This version
https://www.ivoa.net/documents/RegTAP/20240227

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

Previous versions
PR-1.2-20240124
WD-1.2-20220519
REC-1.1
PR-20190911
PR-20190529
PR-20190326
PR-20180731
WD-20171206
REC-1.0

Author(s)
Markus Demleitner, Paul Harrison, Marco Molinaro, Gretchen Greene, Theresa Dower, Menelaos Perdikeas

Editor(s)
Markus Demleitner

Version Control
Revision a5ebd89-dirty, 2024-06-05 16:30:53 +0200

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 Proposed Recommendation made available for public review. It is appropriate to reference this document only as a recommended standard that is under review and which may be changed before it is accepted as a full Recommendation.

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

Contents

1 Introduction

In the Virtual Observatory (VO), registries provide a means for discovering useful resources, i.e., data and services. Individual publishers offer the descriptions for their resources ("resource records") in publishing registries. As of March 2024, there are almost 29000 such resource records active within the VO, originating from about 50 publishing registries.

The protocol spoken by these publishing registries, OAI-PMH (Lagoze and de Sompel et al., 2002), 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 ("searchable registries") harvesting the resource records from the publishing registries and offering rich query facilities to Registry clients. Version 1.0 of the IVOA Registry Interfaces specification (Benson and Plante et al., 2009) defined, among other aspects of the VO registry system, a standard interface for such services. Built on SOAP and an early draft of an XML-based query language, this first attempt was quickly obsoleted by parallel developments in the VO. It was then decided to have searchable registries specified outside of Registry Interfaces.

This document provides one such specification, based in particular on TAP (Dowler and Rixon et al., 2010) and ADQL (Mantelet and Morris et al., 2023). It follows the model of ObsCore (Louys and Tody et al., 2017) 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 and Demleitner et al., 2018). The simplification yields a schema with 18 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 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 and Syntactic Conventions

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, with a standard schema name of `rr`.

The specificiation additionally talks about how to embed these into TAP services, gives additional user defined functions, talks about discovering compliant 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.

Since RegTAP mentions concepts from several different but related domains, we try to give typographic hints as to the nature of entities discussed:

- Names of tables, columns, and functions of the relational registry are written in `green typewriter`.
- Names coming from generic TAP are written in `brown typewriter`.

- VOResource concepts are written in CAPS AND SMALL CAPS (where small caps correspond to lowercase letters in element names of the XML serialisation).
- XML literals (like tag, attribute or XSD type names or special values) are written in *cursive typewriter*.

1.2 The Relational Registry within the VO Architecture

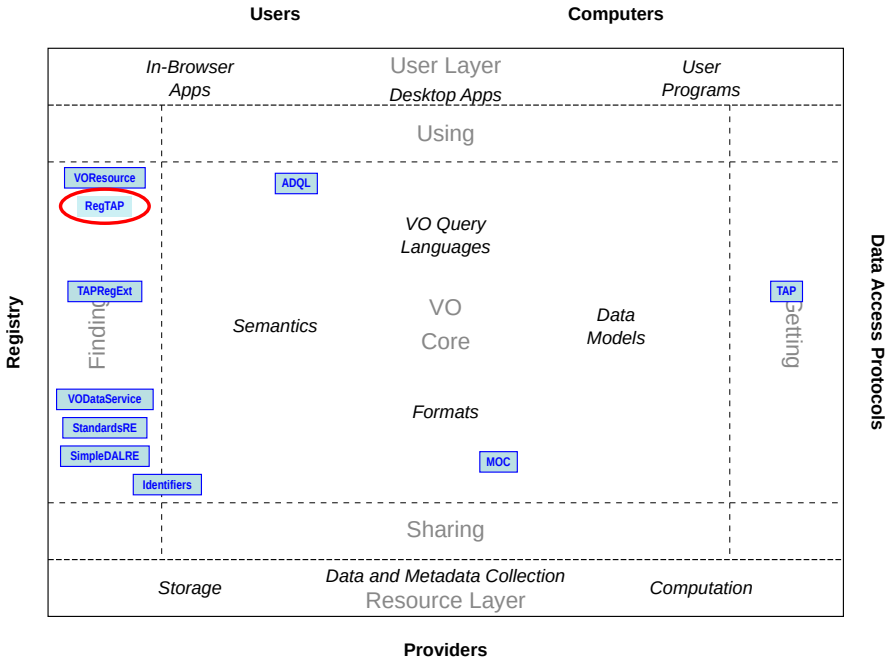

Figure 1: IVOA Architecture diagram with the IVOA Registry Relational Specification (shown as "RegTAP") and the related standards.

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

VOResource, v1.1 (Plante and Demleitner et al., 2018)
: This standard 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). Since its version 1.1, RegTAP incorporates the concepts from VOResource 1.1 but can represent VOResource 1.0 instances (within the limits laid out below) as well.

VODataService, v1.2 (Demleitner and Plante et al., 2021)
: VODataService 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. The present standard should not in general need updates for registry extension updates. For completeness, we note the versions current as of this specification: SimpleDALRegExt 1.2 (Demleitner and Plante et al., 2022), StandardsRegExt 1.0 (Harrison and Burke et al., 2012), TAPRegExt 1.0 (Demleitner and Dowler et al., 2012), Registry Interfaces 1.1 (Dower and Demleitner et al., 2018)

TAP, v1.1 (Dowler and Rixon et al., 2019)
: 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, v2.0 (Demleitner and Plante et al., 2016)
: 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 2.1 (Mantelet and Morris et al., 2023) The rules for ingestion are designed to allow easy queries given the constraints of the IVOA Astronomical Data Query Language. Also, we give some 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 IVOID. 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 `ivoid`, `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 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 that XML schema type (i.e., all whitespace is just a single blank, and there is no leading or trailing whitespace). 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 2.0 has no operators for case-insensitive matching of strings (ILIKE, required by this version of RegTAP, was only defined in ADQL 2.1). Mainly for this reason, RegTAP 1.0 required most columns containing values not usually intended for display to be converted to lower case on ingestion. This also somewhat reduces the likelihood that matches are missed because of different capitalisation, since queries disregarding capitalisation variations will yield empty (rather than partial) results.

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, IVOIDs). 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.

Columns intended for presentation are not case-normalised. When matching against these, queries should use case-insensitive matching using ADQL 2.1's ILIKE or, equivalently, the `ivo_nocasematch` user defined function required by RegTAP.

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 at least up to VOTable 1.5, 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.

4.5 Vocabulary considerations

Since version 1.1, VOResource employs RDF vocabularies to control terms used in several places; in version 1.2, this concerns CONTENT/CONTENTLEVEL, CONTENT/TYPE, CONTENT/SUBJECT, DATE/ROLE, CONTENT/RELATIONSHIP/RELATIONSHIPTYPE. These vocabularies are available from the IVOA vocabulary repository[1] as specified by Vocabularies in the VO, Version 2 (Demleitner and Gray et al., 2023). The relevant vocabulary URIs are given in the VOResource specification and *xs:documentation* elements in the schema file.

For RegTAP, these vocabulary resources are important because the VOResource relationship types and date roles contain some deprecated terms kept for compatibility with VOResource 1.0, together with guidance what to use instead. In order to simplify the usage of vocabulary-controlled RegTAP columns, services MUST translate such deprecated terms when the vocabularies give replacements (i.e., appear as subjects of *ivoasem:useInstead* triples).

[1] https://www.ivoa.net/rdf

Since the vocabularies are expected to develop independently of their originating standards, RegTAP service operators furthermore SHOULD regularly revisit IVOA vocabularies to see if further translations must be done.

In VO practice, many resource records still use subject identifiers that are not taken from the IVOA UAT[2]. Where only the lexical form of the identifier is wrong, RegTAP operators are free to correct the syntax; otherwise, subject identifiers should be ingested as given by the data providers even if they are not drawn from the UAT.

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 and Hollander 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/v1.0}ConeSearch` for the canonical CS:CONESEARCH). This, of course, would make for excessively tedious and error-prone querying.

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

As described in the IVOA endorsed Note "XML schema versioning policies" (Harrison and Demleitner et al., 2018), minor-version updates to XML schemas do not change the namespace URIs. Before the adoption of that note, some schemas introduced namespace URIs that did change on minor versions. For consistency, and because there should not really be discovery use cases based on minor versions of XML schemas, all namespace URIs for the same major version of a standard have the same canonical prefix – e.g., the schema URIs for both SSAP namespaces that SimpleDALRegExt has defined are mapped to `ssap:`.

For reference, table 1 lists the XML namespace URIs and their canonical prefixes for schemata widely used in the VO Registry.

[2] http://www.ivoa.net/rdf/uat

cs	http://www.ivoa.net/xml/ConeSearch/v1.0
dc	http://purl.org/dc/elements/1.1/
oai	http://www.openarchives.org/OAI/2.0/
ri	http://www.ivoa.net/xml/RegistryInterface/v1.0
sia	http://www.ivoa.net/xml/SIA/v1.0
sia	http://www.ivoa.net/xml/SIA/v1.1
slap	http://www.ivoa.net/xml/SLAP/v1.0
ssap	http://www.ivoa.net/xml/SSA/v1.0
ssap	http://www.ivoa.net/xml/SSA/v1.1
tr	http://www.ivoa.net/xml/TAPRegExt/v1.0
vg	http://www.ivoa.net/xml/VORegistry/v1.0
vr	http://www.ivoa.net/xml/VOResource/v1.0
vs	http://www.ivoa.net/xml/VODataService/v1.0
vs	http://www.ivoa.net/xml/VODataService/v1.1
vstd	http://www.ivoa.net/xml/StandardsRegExt/v1.0
xsi	http://www.w3.org/2001/XMLSchema-instance

Table 1: The canonical prefix mapping in the VO Registry as of the publication of 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 in such resource xpaths 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

`xpath:/capability/interface/`

and a column within this table has a utype of

`xpath:accessURL/@use`,

the resulting resource xpath would come out to be

`/capability/interface/accessURL/@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_detail` 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_detail`.

Non-normatively, we give an XSLT sheet[3] producing resource xpaths for suitable VOResource extensions. It is, however, not fully general, as it will only notice direct subclasses of VOResource's Resource, Capability, and Interface classes. If extensions derive from other extensions' subclasses of these classes, the stylesheet would need to be amended.

[3] https://www.ivoa.net/documents/RegTAP/20240227/makeutypes.xslt

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 1.2` with the IVOID

`ivo://ivoa.net/std/RegTAP#1.2`

in the service's capabilities as governed by TAPRegExt (Demleitner and Dowler et al., 2012). Technically, this entails adding

```
<dataModel ivo-id="ivo://ivoa.net/std/RegTAP#1.2"
  >Registry 1.2</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 (sect. 10.8) shows a query that does this.

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.

Section 5.2 of Registry Interfaces 1.1 additionally requires full RegTAP services to register a *vg:Registry*-typed record with a (possibly auxiliary) TAP capability. This record is being used by the RofR, and it opens up a migration path to a data-based discovery pattern[4].

8 RegTAP Tables

All tables making up the RegTAP schema are in the `rr` schema. In both `TAP_SCHEMA` and the VODataService tableset, the `rr` schema MUST be associated with a `utype` matching the data model identifier given in sect. 7, i.e.,

`ivo://ivoa.net/std/RegTAP#1.2`.

In the following table descriptions, the names of tables (cf. Table 2) and columns are normative and MUST be used as given, and all-lowercase. The utypes given in the table descriptions are formed as discussed in section 6 and are subject to the requirements given there. All columns defined in this document MUST have a 1 in the `std` column of the

[4]This would look for schema utypes and appears desirable to enable multiple instances of a data model within one TAP service; it is expected that the recommended discovery pattern in RegTAP 1.3 will be updated accordingly.

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.

Many of the columns specified below are defined as having a "string" data type. This is to be translated into arrays of `char` or `unicodeChar` on VOTable output depending on the service operators' decisions as to the representation of non-ASCII data in the database. For requirements and recommendations regarding national characters in RegTAP, see Sect. 4.4. The length of these arrays is not defined by this standard, where no artificial length limits should be imposed by implementations.

Some of the types are given as "datatype+xtype". In these cases, the xtype MUST be given in VOTable output, and the serialisation rules from DALI (Dowler and Demleitner et al., 2017) apply.

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, miscellaneous normative requirements, typically on case normalization, are given.

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 the number of tables while not complicating common queries significantly.

In VOResource documents, multiple RIGHTS elements are allowed on a single record. This is mainly for compatiblity with DataCite, and multiple RIGHTS elements are discouraged by the VOResource specification at least for use within the VO. RegTAP uses that freedom to include rights and rights_uri columns in rr.resource directly. These columns must be populated, respectively, with the content and the value of the rightsURI attribute of the *first* RIGHTS element within a resource record (falling back to NULL).

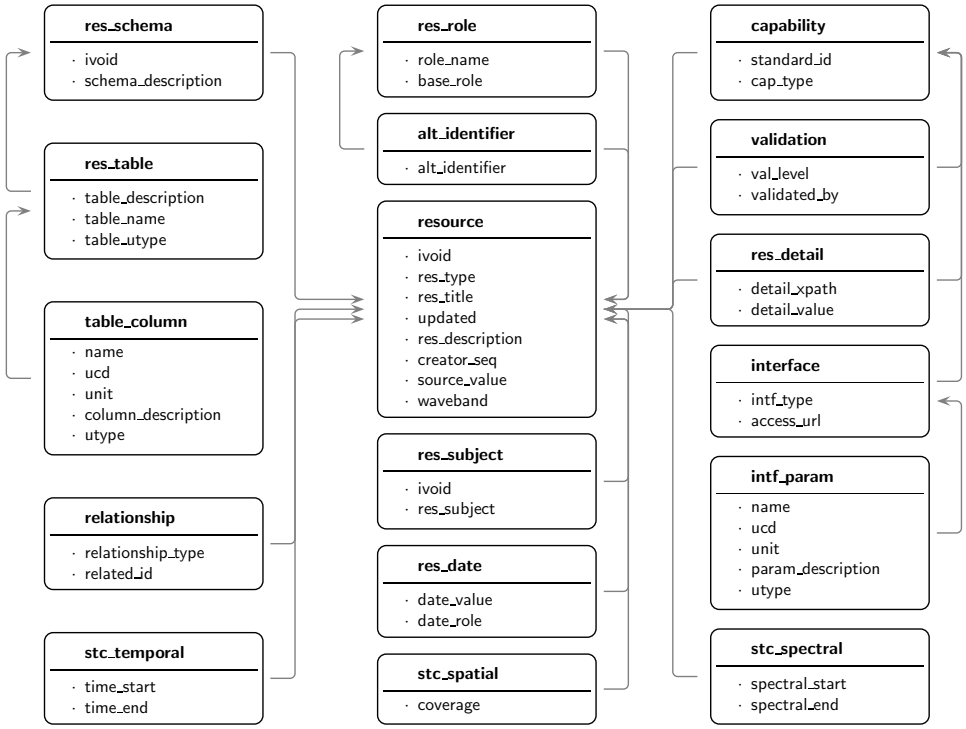

Figure 2: A sketch of the Relational Registry schema. Only the columns considered most interesting for client use are shown. Arrows indicate foreign key-like relationships.

RegTAP services may provide all RIGHTS and RIGHTSURI values through `rr.res_detail` (see sect. 8.13).

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

The `res_type` column reflects the lower-cased value of the RI:RESOURCE element's `xsi:type` attribute, where the canonical prefixes (cf. sect. 5) 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

Name and UType	Description
rr.alt_identifier `xpath:/(curation/creator/|)altIdentifier`	An alternate identifier associated with this record.
rr.capability `xpath:/capability/`	Pieces of behaviour of a resource.
rr.interface `xpath:/capability/interface/`	Information on access modes of a capability.
rr.intf_param `xpath:/capability/interface/param/`	Input parameters for services.
rr.relationship `xpath:/content/relationship/`	Relationships between resources (like mirroring, derivation, serving a data collection).
rr.res_date `xpath:/curation/`	A date associated with an event in the life cycle of the resource.
rr.res_detail	XPath-value pairs for members of resource or capability and their derivations that are less used and/or from VOResource extensions.
rr.res_role	Entities (persons or organizations) operating on resources: creators, contacts, publishers, contributors.
rr.res_schema `xpath:/tableset/schema/`	Sets of tables related to resources.
rr.res_subject `xpath:/content/`	Topics, object types, or other descriptive keywords about the resource.
rr.res_table `xpath:/(tableset/schema/|)table/`	(Relational) tables that are part of schemata or resources.
rr.resource `xpath:/`	The resources (like services, data collections, organizations) present in this registry.
rr.stc_spatial `xpath:/coverage/spatial`	The spatial coverage of resources.
rr.stc_spectral `xpath:/coverage/spectral`	The spectral coverage of resources, given as one or more intervals.
rr.stc_temporal `xpath:/coverage/temporal`	The temporal coverage of resources, given as one or more intervals.
rr.table_column `xpath:/(tableset/schema/|)/table/column/`	Metadata on columns of a resource's tables.
rr.tap_table	TAP-queriable tables.
rr.validation `xpath:/(capability/|)validationLevel`	Validation levels for resources and capabilities.

Table 2: The tables making up the TAP data model `Registry 1.2`

for `res_type`:

vg:authority A naming authority; as described in the IVOA Identifiers specification (Demleitner and Plante et al., 2016), these records are used to guarantee global uniqueness of IVOIDs.

vg:registry A registry. This can be a publishing registry (which have at least

one capability element of type *vg:Harvest*), or a searchable registry (like a RegTAP service). See Registry Interfaces 1.1 on how to apply this resource type.

vr:organisation The main purpose of an organisation as a registered resource is to be referenced by IVOID 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.

vs:catalogservice A service that interacts with one or more specified tables.

vs:catalogresource A resource accessible through collective services (which would typically be declared through auxiliary capabilities) or non-IVOA protocols (typical example: A set of tables accessible within a larger TAP service).

vs:dataservice A service for accessing astronomical data; publishers choosing this over VS:CATALOGSERVICE probably intend to communicate that the resource does not have an intrinsically tabular structure.

vs:dataresource A non-tabular resource accessible through collective services (which would typically be declared through auxiliary capabilities) or non-IVOA protocols.

vs:datacollection A resource type intended by VODataService version 1.1 to be used for data-only resources. Data providers should use VS:CATALOGRESOURCE or VS:DATARESOURCE instead.

vstd:standard A description of a standard specification.

The STATUS attribute of VR:RESOURCE is considered an implementation detail of the XML serialization and is not reflected 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, datatypes, and descriptions for the rr.resource table

ivoid `xpath:identifier`	string	Unambiguous reference to the resource conforming to the IVOA standard for identifiers.
res_type `xpath:@xsi:type`	string	Resource type (something like vg:authority, vs:catalogservice, etc).
created `xpath:@created`	character[19] +timestamp	The UTC date and time this resource metadata description was created.
short_name `xpath:shortName`	string	A short name or abbreviation given to something, for presentation in space-constrained fields (up to 16 characters).
res_title `xpath:title`	string	The full name given to the resource.
updated `xpath:@updated`	character[19] +timestamp	The UTC date this resource metadata description was last updated.
content_level `xpath:content/contentLevel`	string	A hash-separated list of content levels specifying the intended audience.
res_description `xpath:content/description`	string	An account of the nature of the resource.
reference_url `xpath:content/referenceURL`	string	URL pointing to a human-readable document describing this resource.
creator_seq `xpath:curation/creator/name`	string	The creator(s) of the resource in the order given by the resource record author, separated by semicolons.
content_type `xpath:content/type`	string	A hash-separated list of natures or genres of the content of the resource.
source_format `xpath:content/source/@format`	string	The format of source_value. This, in particular, can be "bibcode".
source_value `xpath:content/source`	string	A bibliographic reference from which the present resource is derived or extracted.
res_version `xpath:curation/version`	string	Label associated with creation or availablilty of a version of a resource.
region_of_regard `xpath:coverage/regionOfRegard`	real	A single numeric value representing the angle, given in decimal degrees, by which a positional query against this resource should be "blurred" in order to get an appropriate match.
waveband `xpath:coverage/waveband`	string	A hash-separated list of regions of the electro-magnetic spectrum that the resource's spectral coverage overlaps with.
rights `xpath:/rights`	string	A statement of usage conditions (license, attribution, embargo, etc).
rights_uri `xpath:/rights/@rightsURI`	string	A URI identifying a license the data is made available under.

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.

When querying content_type and content_level, note that resource record authors should restrict themselves to terms from the vocabularies at http://ivoa.net/rdf/voresource/content_type and http://ivoa.net/rdf/voresource/content_level, respectively

The content of incoming CONTENT/TYPE and CONTENT/LEVEL elements must be normalized according to the rules laid down in sect. 4.5 before further processing.

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):

base_role value	contact	publisher	creator	contributor
role_name	co/name	cp	cc/name	cb
role_ivoid	co/name/@ivo-id	cp/@ivo-id	cc/name/@ivo-id	cb/@ivo-id
address	co/address	N/A	N/A	N/A
email	co/email	N/A	N/A	N/A
telephone	co/telephone	N/A	N/A	N/A
logo	co/logo	N/A	cc/logo	N/A

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

When matching against role_name, please be aware that despite the admonitions in section 3.1.2 of VOResource 1.1 (which recommends a format like Last, F. for person names), as of this writing the wide majority of role names in the VO Registry are not in this format. Hence, name matching in RegTAP at this point should be very lenient.

Column names, utypes, datatypes, and descriptions for the rr.res_role table

ivoid `xpath:/identifier`	string	The parent resource.
role_name	string	The real-world name or title of a person or organization.
role_ivoid	string	An IVOA identifier of a person or organization.
street_address	string	A mailing address for a person or organization.
email	string	An email address the entity can be reached at.
telephone	string	A telephone number the entity can be reached at.
logo	string	URL pointing to a graphical logo, which may be used to help identify the entity.
base_role	string	The role played by this entity; this is one of contact, publisher, contributor, or creator.

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

The following columns MUST be lowercased during ingestion: `ivoid`, `role_ivoid`, `base_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, datatypes, and descriptions for the rr.res_subject table

ivoid `xpath:/identifier`	string	The parent resource.
res_subject `xpath:subject`	string	Topics, object types, or other descriptive keywords about the resource.

The `ivoid` column should be an explicit foreign key into `resource`. It is recommended to index the `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`.

The content of incoming SUBJECT elements may be normalized according to the rules laid down in sect. 4.5.

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, datatypes, and descriptions for the rr.capability table

ivoid `xpath:/identifier`	string	The parent resource.
cap_index	(key)	An arbitrary identifier of this capability within the resource.
cap_type `xpath:@xsi:type`	string	The type of capability covered here. If looking for endpoints implementing a certain standard, you should not use this column but rather match against standard_id.
cap_description `xpath:description`	string	A human-readable description of what this capability provides as part of the over-all service.
standard_id `xpath:@standardID`	string	A URI for a standard this capability conforms to.

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

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

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.

Column names, utypes, datatypes, and descriptions for the rr.res_schema table

ivoid `xpath:/identifier`	string	The parent resource.
schema_index	(key)	An arbitrary identifier for the res_schema rows belonging to a resource.
schema_description `xpath:description`	string	A free text description of the tableset explaining in general how all of the tables are related.
schema_name `xpath:name`	string	A name for the set of tables.
schema_title `xpath:title`	string	A descriptive, human-interpretable name for the table set.
schema_utype `xpath:utype`	string	An identifier for a concept in a data model that the data in this schema as a whole represent.

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

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

8.6 The res_table Table

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

The table contains 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.

Column names, utypes, datatypes, and descriptions for the rr.res_table table

ivoid `xpath:/identifier`	string	The parent resource.
schema_index	(key)	Index of the schema this table belongs to, if it belongs to a schema (otherwise NULL).
table_description `xpath:description`	string	A free-text description of the table's contents.
table_name `xpath:name`	string	The fully qualified name of the table. As per VODataService, this includes all catalog or schema prefixes needed to distinguish it in a query, and it comes with SQL delimiters where necessary.
table_index	(key)	An arbitrary identifier for the tables belonging to a resource.
table_title `xpath:title`	string	A descriptive, human-interpretable name for the table.
table_type `xpath:@type`	string	A name for the role this table plays. Recognized values include "output", indicating this table is output from a query; "base_table", indicating a table whose records represent the main subjects of its schema; and "view", indicating that the table represents a useful combination or subset of other tables. Other values are allowed.
table_utype `xpath:utype`	string	An identifier for a concept in a data model that the data in this table as a whole represent.

This table should have an explicit primary key made up of `ivoid` and `table_index`. The `ivoid` column should be an explicit foreign key into `resource`. It is recommended to maintain an index on at least the `table_description` column, ideally one suited for queries with `ivo_hasword`. Since `table_utype` is used in data discovery, it should also be indexed.

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

8.7 The table_column Table

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

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

The `flag` column contains a concatenation of all values of a COLUMN element's FLAG children, separated by hash characters. Use the

ivo_hashlist_has function in queries against flag.

The table_column table also includes information from VODataService's data type concept. VODataService 1.1 includes several type systems (VOTable, ADQL, Simple). The 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. Modern resource records should always use vs:votabletype, but column declarations using the other type systems are still present in the VO.

Column names, utypes, datatypes, and descriptions for the rr.table_column table

ivoid `xpath:/identifier`	string	The parent resource.
table_index	(key)	Index of the table this column belongs to.
name `xpath:name`	string	The name of the column.
ucd `xpath:ucd`	string	A unified content descriptor that describes the scientific content of the column.
unit `xpath:unit`	string	The unit associated with all values in the column.
utype `xpath:utype`	string	An identifier for a role in a data model that the data in this column represents.
std `xpath:@std`	integer	If 1, the meaning and use of this column is reserved and defined by a standard model. If 0, it represents a database-specific column that effectively extends beyond the standard.
datatype `xpath:dataType`	string	The type of the data contained in the column.
extended_schema `xpath:dataType/@extendedSchema`	string	An identifier for the schema that the value given by the extended attribute is drawn from.
extended_type `xpath:dataType/@extendedType`	string	A custom type for the values this column contains.
arraysize `xpath:dataType/@arraysize`	string	The shape of the array that constitutes the value, e.g., 4, *, 4*, 5x4, or 5x*, as specified by VOTable.
delim `xpath:dataType/@delim`	string	The string that is used to delimit elements of an array value when arraysize is not '1'.
type_system `xpath:dataType/@xsi:type`	string	The type system used, as a QName with a canonical prefix; this will ususally be one of vs:simpledatatype, vs:votabletype, and vs:taptype.
flag `xpath:flag`	string	Hash-separated keywords representing traits of the column. Recognized values include "indexed", "primary", and "nullable".
column_description `xpath:description`	string	A free-text description of the column's contents.

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 column'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:ACCESSURL types from VOResource. The integration of ACCESSURL into the interface table means that an interface in the relational registry can only have one access URL, where in VOResource it can have many. 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 more SECURITYMETHOD children to convey support for authentication and authorization methods. Apart from an identifier for an authentication method – usually taken from the SSO document Taffoni and Schaaf et al. (2017) –, no actual content has been specified so far for these elements. Also, there are as of now no actual discovery cases employing this information except "filter out services requiring authentication". Hence, RegTAP 1.2 does not attempt to map SECURITYMETHOD except through the authenticated_only column which is required to be 0 when there is no SECURITYMETHOD or at least one SECURITYMETHOD without a STANDARDID on an interface, 1 otherwise.

Clients not prepared to authenticate to services should always include a authenticated_only=0 condition when retrieving access URLs from RegTAP 1.2 services, as it is conceivable that a future VO will contain many services requiring authentication and users should not have to try out which of them they can actually use.

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 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 "simple" 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, utypes, datatypes, and descriptions for the rr.interface table

ivoid `xpath:/identifier`	string	The parent resource.
cap_index	(key)	The index of the parent capability.
intf_index	(key)	An arbitrary identifier for the interfaces of a resource.
intf_type `xpath:@xsi:type`	string	The type of the interface (vr:webbrowser, vs:paramhttp, etc).
intf_role `xpath:@role`	string	An identifier for the role the interface plays in the particular capability. If the value is equal to "std" or begins with "std:", then the interface refers to a standard interface defined by the standard referred to by the capability's standardID attribute.
std_version `xpath:@version`	string	The version of a standard interface specification that this interface complies with. When the interface is provided in the context of a Capability element, then the standard being refered to is the one identified by the Capability's standardID element.
query_type `xpath:queryType`	string	Hash-joined list of expected HTTP method (get or post) supported by the service.
result_type `xpath:resultType`	string	The MIME type of a document returned in the HTTP response.
wsdl_url `xpath:wsdlURL`	string	The location of the WSDL that describes this Web Service. If NULL, the location can be assumed to be the accessURL with '?wsdl' appended.
url_use `xpath:accessURL/@use`	string	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').
access_url `xpath:accessURL`	string	The URL at which the interface is found.
mirror_url `xpath:mirrorURL`	string	Secondary access URLs of this interface, separated by hash characters.
authenticated_only	integer	A flag for whether an interface is available for anonymous use (=0) or only authenticated clients are served (=1).

This table should have the pair `ivoid`, `cap_index` as an explicit foreign key into `capability`, and the pair `ivoid`, and `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.

Column names, utypes, datatypes, and descriptions for the rr.intf_param table

ivoid xpath:/identifier	string	The parent resource.
intf_index	(key)	The index of the interface this parameter belongs to.
name xpath:name	string	The name of the parameter.
ucd xpath:ucd	string	A unified content descriptor that describes the scientific content of the parameter.
unit xpath:unit	string	The unit associated with all values in the parameter.
utype xpath:utype	string	An identifier for a role in a data model that the data in this parameter represents.
std xpath:@std	integer	If 1, the meaning and use of this parameter is reserved and defined by a standard model. If 0, it represents a database-specific parameter that effectively extends beyond the standard.
datatype xpath:dataType	string	The type of the data contained in the parameter.
extended_schema xpath:dataType/@extendedSchema	string	An identifier for the schema that the value given by the extended attribute is drawn from.
extended_type xpath:dataType/@extendedType	string	A custom type for the values this parameter contains.
arraysize xpath:dataType/@arraysize	string	The shape of the array that constitutes the value, e.g., 4, *, 4*, 5x4, or 5x*, as specified by VOTable.
delim xpath:dataType/@delim	string	The string that is used to delimit elements of an array value when arraysize is not '1'.
param_use xpath:@use	string	An indication of whether this parameter is required to be provided for the application or service to work properly (one of required, optional, ignored, or NULL).
param_description xpath:description	string	A free-text description of the parameter's contents.

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 IVOIDs, in the relational model, the pairs are stored directly. It is straightforward to translate between the two representations in the database ingestor.

Column names, utypes, datatypes, and descriptions for the rr.relationship table

ivoid `xpath:/identifier`	string	The parent resource.
relationship_type `xpath:relationshipType`	string	The type of the relationship; these terms are drawn from a controlled vocabulary and are DataCite-compatible.
related_id `xpath:relatedResource/@ivo-id`	string	The IVOA identifier for the resource referred to.
related_name `xpath:relatedResource`	string	The name of resource that this resource is related to.

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`.

The content of incoming RELATIONSHIPTYPE elements must be normalized according to the rules laid down in sect. 4.5 before lowercasing.

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 and IVOA Resource Registry Working Group 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, datatypes, and descriptions for the rr.validation table

ivoid `xpath:/identifier`	string	The parent resource.
validated_by `xpath:validationLevel/@validatedBy`	string	The IVOA ID of the registry or organisation that assigned the validation level.
val_level `xpath:validationLevel`	integer	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.
cap_index	(key)	If non-NULL, the validation only refers to the capability referenced here.

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, datatypes, and descriptions for the rr.res_date table

ivoid `xpath:/identifier`	string	The parent resource.
date_value `xpath:date`	character[19] +timestamp	A date associated with an event in the life cycle of the resource.
value_role `xpath:date/@role`	string	A string indicating what the date refers to, e.g., created, availability, updated. This value is generally drawn from a controlled vocabulary.

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

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

The content of incoming DATE/@ROLE attributes must be normalized according to the rules laid down in sect. 4.5 before lowercasing.

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:LANGUAGEFEATURE or VSTD:KEY being examples – cannot be kept in this table. If a representation of such information in the relational registry is required, this standard will need to be changed.

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, IVOID 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 `ivo_nocasematch` 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 xpaths, 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_detail` as the extensions require it.

Column names, utypes, datatypes, and descriptions for the rr.res_detail table

ivoid `xpath:/identifier`	string	The parent resource.
cap_index	(key)	The index of the parent capability; if NULL the xpath-value pair describes a member of the entire resource.
detail_xpath	string	The xpath of the data item.
detail_value	string	(Atomic) value of the member.

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

The following column MUST be lowercased during ingestion: `ivoid`. Clients are advised to use `ivo_nocasematch` to search in `detail_value` if the values are to be compared case-insensitively (e.g., all IVOIDs).

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 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 identifiers are stored in URI form and hence identifiers of different types cannot clash, RegTAP does not keep track where an alternate identifier was encountered. Instead, the `alt_identifier` table just links IVOIDs and alternate identifiers:

Column names, utypes, datatypes, and descriptions for the rr.alt_identifier table

ivoid `xpath:/identifier`	string	The parent resource.
alt_identifier	string	An identifier for the resource or an entity related to the resource in URI form.

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

8.15 The stc_spatial Table

Since VODataService 1.2, registry records can represent their resource's spatial coverage using spatial MOCs (Fernique and Nebot et al., 2022). The `stc_spatial` table is a direct reflection of this metadata:

Column names, utypes, datatypes, and descriptions for the rr.stc_spatial table

ivoid xpath:/identifier	string	The parent resource.
coverage xpath:.	string +moc	A geometry representing the area a resource contains data for; this should be tight at least with a resolution of degrees.
ref_system_name xpath:@frame	string	The reference frame coverage is written in. This is currently reserved and fixed to NULL. Clients should always add a constraint to NULL for this to avoid matching non-celestial resources later.

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

The details of how the MOC-valued coverage is entered and retrieved will be given in version 1.2 of DALI (Dowler and Demleitner et al., 2017). Implementations MUST evaluate the ADQL CONTAINS and INTERSECTS predicates with coverage as one argument and ADQL CIRCLEs and POLYGONs as the other, and they must support CONTAINS with an ADQL POINT in the first argument. There are no expectations that the predicates are computed exactly, but implementations should strive to limit the number of false positives; clients are advised that on services supporting MOC literals, it is probably much faster and more exact to use MOC-MOC comparisons to query `coverage`.

8.16 The stc_temporal Table

Since VODataService 1.2, registry records can represent their resource's temporal coverage as a union of time intervals. The `stc_temporal` table is a direct reflection of this metadata:

Column names, utypes, datatypes, and descriptions for the rr.stc_temporal table

ivoid xpath:/identifier	string	The parent resource.
time_start xpath:.	real	Lower limit of a time interval covered by the resource in MJD.
time_end xpath:.	real	Upper limit of a time interval covered by the resource in MJD.

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

Clients are advised that the `ivo_interval_overlaps` user defined function is available to conveniently compare a user-specified interval of interest to `time_start` ⋯ `time_end`.

Since VODataService temporal coverage is given in MJD, the rows for `time_start` and `time_end` in `TAP_SCHEMA.columns` MUST have the appropriate VOUnits (Gray and Cecconi et al., 2023) string, `d`, in their `unit`

column.

8.17 The stc_spectral Table

Since VODataService 1.2, registry records can represent their resource's spectral coverage as a union of energy intervals. The `stc_spectral` table is a direct reflection of this metadata:

Column names, utypes, datatypes, and descriptions for the rr.stc_spectral table

ivoid `xpath:/identifier`	string	The parent resource.
spectral_start `xpath:.`	real	Lower limit (in Joules) of messenger energy interval covered by the resource (for the solar system barycenter).
spectral_end `xpath:.`	real	Upper limit (in Joules) of messenger energy interval covered by the resource (for the solar system barycenter).

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

Clients are advised that the `ivo_interval_overlaps` user defined function is available to conveniently compare a user-specified intervals of interest to `spectral_start` ··· `spectral_end`.

Since VODataService spectral coverage gives energies in Joules, the rows for `spectral_start` and `spectral_end` in `TAP_SCHEMA.columns` MUST have the appropriate VOUnits string, `J`, in their `unit` column.

8.18 The tap_table View

Several Registry clients need to easily obtain metadata on tables queriable through TAP. Since the Registry data model gives services some freedom registering these – they can occur in tablesets of TAP services, in tablesets of records having auxiliary TAP capabilities (Demleitner and Taylor, 2019), or both – it is hard to write ADQL producing such a relation. Hence, starting with version 1.2 of RegTAP, implementing services must provide a view encapsulating a query yielding all tables in `rr.res_table`

1. that are accessible through a TAP service
2. and are not declared as *output* tables (which by definition cannot be queried)
3. exactly once for each actual table (i.e., there cannot be two rows in the view having the same (`svcid`, `table_name`))
4. with references to both a full metadata record and the record of the TAP service publishing the resource.

Condition 4 requires an explanation: A given table can be both in the tableset of the TAP service (that will in general have very little additional information on the table) and in the tableset of a specific resource (which will contain rich metadata on the table). In the latter case, `tap_table` must reference the specific resource as the full metadata record (the `resid` column). Tables only present in their TAP services' tableset will have identical `resid` and `svcid`.

The `tap_table` view has the following columns:

Column names, utypes, datatypes, and descriptions for the rr.tap_table table

resid	string	IVOA identifier of the resource this table was taken from (where there is a dedicated resource containing this table in its tableset, that resource is preferred over a TAP service).
svcid	string	IVOA identifier of the TAP service making this table queriable.
table_name `xpath:name`	string	The fully qualified name of the table. As per VODataService, this includes all catalog or schema prefixes needed to distinguish it in a query, and it comes with SQL delimiters where necessary.
table_title `xpath:title`	string	A descriptive, human-interpretable name for the table.
table_description `xpath:description`	string	A free-text description of the table's contents.
table_utype `xpath:utype`	string	An identifier for a concept in a data model that the data in this table as a whole represent.

Since `rr.tap_table` is (at least conceptually; this specification does not forbid making it a materialised view or a physical table) a view, it inherits the properties of the contributing tables. This means that `table_title` and `table_description` should be queried using `ivo_hasword`, and that `table_utype` should have an index. By construction, (`svcid`, `table_name`) is suitable as a primary key of the relation.

Appendix C gives a standard SQL query that will produce the view specified here from other RegTAP tables.

9 RegTAP Requirements on TAP services

Since RegTAP 1.2, implementing services MUST implement at least version 2.1 of ADQL.

Since RegTAP deals with text much more intensively than is usual for the astrophysical data that TAP and ADQL were designed for and some query patterns uncommon in astrophysics significantly help writing RegTAP queries, TAP services implementing RegTAP MUST implement some ADQL

extensions, partly specified as ADQL optional features, partly in ADQL User Defined Functions (UDFs).

9.1 ADQL Optional Features Required for RegTAP

TAP Servers implementing the `ivo://ivoa.net/std/RegTAP#1.2` data model MUST implement the following optional features defined in ADQL 2.1 (Mantelet and Morris et al., 2023):

COALESCE
: Primarily in order to make the use of `ivo_string_agg` predictable in the presence of NULL values in columns like `standard_id`, RegTAP services MUST provide the COALESCE feature in ivo://ivoa.net/std/TAPRegExt#features-adql-conditional.

ILIKE
: As a standard alternative to `ivo_nocasematch` as employed by RegTAP earlier than 1.2, RegTAP services MUST provide the ILIKE feature in ivo://ivoa.net/std/TAPRegExt#features-adql-string.

WITH
: To let clients more clearly structure their queries, RegTAP services MUST implement common table expressions as per the WITH feature in ivo://ivoa.net/std/TAPRegExt#features-adql-common-table.

9.2 User Defined Functions Required for RegTAP

TAP Servers implementing the `ivo://ivoa.net/std/RegTAP#1.2` data model MUST implement the following User Defined Functions in their ADQL language, given here with signatures written as recommended in TAPRegExt (Demleitner and Dowler et al., 2012):

`ivo_nocasematch(value VARCHAR(*), pat VARCHAR(*))->INTEGER`
: The function returns 1 if `pat` matches `value`, 0 otherwise. `pat` is defined as for the SQL LIKE operator, but the match is performed case-insensitively. Clients that only talk to RegTAP 1.2 and later should prefer the ILIKE operator.

`ivo_hasword(haystack VARCHAR(*), needle VARCHAR(*)) -> INTEGER`
: The function takes two strings and returns 1 if the second is contained in the first one in a "word" sense, i.e., delimited by non-letter characters or the beginning or end of the string, where case is ignored. It returns 0 otherwise. Additionally, servers MAY employ techniques to improve recall, in particular stemming. Registry clients must hence expect different results from different servers when using `ivo_hasword`; for such queries trying them on multiple registries may improve recall.

`ivo_hashlist_has(hashlist VARCHAR(*), item VARCHAR(*)) -> INTEGER`
: The function takes two strings; the first is a list of words not containing the hash sign (#), concatenated by hash signs, the second is a word not containing the hash sign. It returns 1 if, compared case-insensitively, the second argument is in the list of words encoded in the first argument, 0 otherwise. The behavior for second arguments containing a hash sign is undefined.

`ivo_string_agg(expr VARCHAR(*), delim VARCHAR(*)) -> VARCHAR(*)`
: An aggregate function returning all values of `expr` within a GROUP concatenated with `delim`. NULLs in the aggregate do not contribute, an empty aggregate yields an empty string.

`ivo_interval_overlaps(l1 T, h1 T, l2 T, h2 T) -> INTEGER`
: The function returns 1 if the interval [l1...h1] overlaps with the interval [l2...h2]. For the purposes of this function, the case l1=h2 or l2=h1 is treated as overlap. The function returns 0 for non-overlapping intervals. The function must be available for both integers and floating point numbers; on most systems, this will mean that `T` is `NUMERIC`.

Reference implementations of the functions for the PostgreSQL database system are given in Appendix B. As required for UDFs with an *ivo_* prefix, these functions are also listed in the Catalogue of ADQL User Defined Functions (Campillo and Demleitner, 2023).

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 limit the matches to interfaces declared with ROLE equal to `std`. This filters out WEBBROWSER interfaces that some data providers add in SCS or SSAP capabilities (a practice not recommended). Future standards might require more specific strings starting with `std:` in this place; discovery for those needs to be adapted accordingly.

In RegTAP 1.0, this filtering was effected by constraining the interface type to VS:PARAMHTTP. As discussed in that specification, this adopted existing discovery patterns and worked around missing metadata in VOResource records. This workaround is no longer necessary, and future standards should be free to use other interface types rather than VR:PARAMHTTP.

Note that it still is possible that a single resource will return multiple access URLs with the query patterns given here. Clients can assume that all

access URLs returned in this way correspond to their constraints. Therefore, it is legal to randomly pick one of those.

Service standards can give discovery patterns different from the ones shown here if their particular use cases require them.

10.1 TAP accessURLs

Problem: Find all TAP services; return their accessURLs

As the capability type is in `rr.capability`, whereas the access URL can be found from `rr.interface`, this requires a (natural) join.

Clients communicating with a RegTAP 1.1 or later service should request the `authenticated_only` column. If this is 1, the service requires some sort of authentication and should only presented to users if a client has the necessary infrastructure for the authentication system.

Hence, clients only interested in services not requiring authentication should use

```
SELECT ivoid, access_url
FROM rr.capability
NATURAL JOIN rr.interface
WHERE standard_id like 'ivo://ivoa.net/std/tap%'
  AND intf_role='std'
  AND authenticated_only=0
```

Analogous considerations apply to the other example queries

Other `standard_id`s 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
FROM rr.capability
  NATURAL JOIN rr.resource
  NATURAL JOIN rr.interface
  NATURAL JOIN rr.res_subject
WHERE standard_id like 'ivo://ivoa.net/std/sia%'
  AND intf_role='std'
  AND (
    res_subject ILIKE '%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.

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

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`.

```
SELECT ivoid, access_url
FROM rr.capability
  NATURAL JOIN rr.table_column
  NATURAL JOIN rr.interface
WHERE standard_id LIKE 'ivo://ivoa.net/std/conesearch%'
  AND intf_role='std'
  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 usally 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 1.2.

```
SELECT access_url
FROM rr.interface
NATURAL JOIN rr.capability
NATURAL JOIN rr.res_detail
WHERE standard_id LIKE 'ivo://ivoa.net/std/tap%'
  AND intf_role='std'
  AND detail_xpath='/capability/dataModel/@ivo-id'
  AND 1=ivo_nocasematch(detail_value,
    'ivo://ivoa.net/std/regtap#1.%')
  AND authenticated_only=0
```

Also note the remarks on the likely evolution of data model query patterns in sect. 7.

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
```

```
FROM rr.capability
  NATURAL JOIN rr.interface
  NATURAL JOIN rr.table_column
  NATURAL JOIN rr.res_table
WHERE standard_id LIKE 'ivo://ivoa.net/std/tap%'
  AND intf_role='std'
  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`:

```
SELECT access_url
FROM rr.res_detail
  NATURAL JOIN rr.capability
  NATURAL JOIN rr.interface
WHERE detail_xpath='/capability/dataSource'
  AND intf_role='std'
  AND standard_id LIKE '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 IVOID 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'`.

```
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 data collections either as such or with "auxiliary capabilities" that are fully described elsewhere; a practice for

doing that is discussed in an Endorsed Note on discovering data collections within services (Demleitner and Taylor, 2019).

When following this pattern, data collections records should provide an *isServedBy* relationship to the resources providing the access services for the data collction (like a TAP or a SIAP service).

While the access URLs can typically be established from the auxiliary capabilities themselves, several use cases require finding out more about the publishing service. To locate its metadata, inspect `rr.relationship` and use it to select records from `rr.capability`; this requires an explicit join condition, as in this case the capabilities are for the *related* record, not for the originally matched one.

```
SELECT *
FROM rr.relationship AS a
  JOIN rr.capability AS b
    ON (a.related_id=b.ivoid)
WHERE
  relationship_type='isservedby'
  AND a.ivoid='ivo://cds.vizier/j/a+a/649/a25'
```

10.13 Constraints on Space, Time, and Spectrum

Problem: Give me resources that cover M 101 ($\alpha = 210.80$, $\delta = 54.35$, Diameter about 0.3°) in the mid-infrared around 5 μm in August 2010.

Without further database support, clients need to manually convert the spectral coordinate to energy ($hc/\lambda \approx 3.97 \times 10^{-20}$ J) and time (August 1st, 2010 starts MJD 55409.0) to the quantities RegTAP expects.

This would yield a query like (the explicit MOC conversion is a common device to speed the query up; without it, the database would convert the circle once for each coverage, to the respective order):

```
SELECT ivoid
FROM rr.stc_spatial
  NATURAL JOIN rr.stc_spectral
  NATURAL JOIN rr.stc_temporal
WHERE
  1=CONTAINS(MOC(8, CIRCLE(210.80, 54.35, 0.3)), coverage)
  AND 1=ivo_interval_overlaps(time_start, time_end, 55409, 55440)
  AND 3.97e-20 between spectral_start and spectral_end
```

In particular when more complex geometries are desired, clients will want to pass in MOCs directly. Conversely, RegTAP services may provide the additional user-defined functions that allow specifying temporal and spectral constraints in different, perhaps human-friendlier ways. For instance, once support for the relevant UDFs is established using the TAP capabilities, the

above query could also be written as (the MOC given is the circle above at order 8):

```
SELECT ivoid
FROM rr.stc_spatial
  NATURAL JOIN rr.stc_spectral
  NATURAL JOIN rr.stc_temporal
WHERE
  1=CONTAINS(MOC('8/182947 182950 182952-182953 182955-182956 8/'), coverage)
  AND 1=ivo_interval_overlaps(
    time_start, time_end,
    gavo_to_mjd('2010-08-01'), gavo_to_mjd('2010-08-31'))
  AND gavo_specconv(5e-6, 'm', 'J') between spectral_start and spectral_end
```

10.14 Reliably Doing Arrays of Strings

In particular libraries and user interfaces often want to retrieve something like "all access options" for a set of resources; this would mean at least access URLs and their standard identifiers. Since RegTAP does not require support for string arrays, this is not entirely trivial, since with the traditional workaround, `ivo_string_agg`, NULLs do not appear at all; that way, re-relating, say, access URLs and standard ids (some of which may be NULL) becomes impossible for a client.

The solution to this problem is to turn NULLs to empty strings using `COALESCE`. The following example also illustrates how to use CTEs to structure complex queries of this sort, in this case by pre-selecting ivoids of interest and then using that pre-selected list in a NATURAL JOIN to restrict the query to, in this case, resources matching a certain keyword. This is sometimes simpler than putting the condition in a WHERE clause, in particular when a table may contribute more than one output row.

```
WITH candidates AS (
SELECT ivoid
FROM rr.res_subject
WHERE res_subject='solar-system-planets')
SELECT
  ivoid,
  ivo_string_agg(COALESCE(access_url, ''), '<sep>') AS access_urls,
  ivo_string_agg(COALESCE(standard_id, ''), '<sep>') AS standard_ids
FROM
  rr.capability
  NATURAL JOIN rr.interface
  NATURAL JOIN candidates
GROUP BY ivoid
```

A XPaths for res_detail

This appendix defines the `res_detail` 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 legacy VODataService vs:DataCollection-typed records, this is the URL that can be used to download the data contained. Do *not* enter accessURLs from interface elements 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 IVOID 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/securityMethod/@standardID (!) A standard identifier for an authentication method supported on an interface (vr).

/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 IVOID 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 maximum image size in the latitude (Dec.) direction in pixels (sia-1.0).

/capability/maxImageSize/long The maximum 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 IVOID 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 IVOID 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_detail`, 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 (!) IVOID 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; clients should generally use the `rights` column in `rr.resource` (vr).

/rights/@rightsURI A formal identifier for a license a resource is made available under; clients should generally use the `rights_uri` column in `rr.resource` (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 justifiable just so they can be exposed.

B The Extra UDFs in PL/pgSQL

What follows are (non-normative) implementations of four of the User Defined Functions specificed in section 9.2 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 functions given here with simple boolean expressions in the ADQL translation layer whenever possible.

```
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 $func$
  -- postgres can't RE-escape a user string; hence, we'll have
  -- to work on the hashlist (this assumes hashlist is already
  -- lowercased).
  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_interval_overlaps(l1 NUMERIC, h1 NUMERIC,
    l2 NUMERIC, h2 NUMERIC)
RETURNS BOOLEAN AS $func$
  SELECT h1>=l2 AND h2>=l1 AND l1<=h1 AND l2<=h2
$func$ LANGUAGE SQL STABLE;

-- ivo_string_agg directly corresponds to string_agg; this translation
-- should be done in the ADQL translator.
```

C A View Definition for tap_table (non-normative)

While RegTAP operators are free to implement `tap_table` as convenient on their platform, here is a standard SQL query that produces a result compliant to the constraints in Sect. 8.18 assuming 2024 Registry conventions:

```
WITH
  fromres AS (
    -- tables coming in through relationships; only those declaring
    -- an auxiliary capability *and* a relationship will be considered
    -- The GROUP BY and MIN hack is necessary since multiple of these
    -- may declare the same table (e.g., ivoa.obscore for data collections
    -- published through obscore).
    SELECT
        MIN(tabcap.ivoid) as resid,
        related_id as svcid,
        table_name,
      MIN(table_title) as table_title,
      MIN(table_description) as table_description,
      MIN(table_utype) as table_utype
    FROM rr.res_table as tab
    NATURAL JOIN rr.capability as tabcap
    NATURAL JOIN rr.relationship
    JOIN rr.capability AS svccap
      ON (svccap.ivoid=related_id)
    WHERE
      (table_type!='output' OR table_type IS NULL)
      AND svccap.standard_id='ivo://ivoa.net/std/tap'
      AND tabcap.standard_id='ivo://ivoa.net/std/tap#aux'
      AND relationship_type='isservedby'
    GROUP BY related_id, table_name),

  fromtap AS (
    -- tables directly attached to the TAP service
    SELECT rt.ivoid as resid, ivoid as svcid,
      table_name, table_title,
      table_description, table_utype
    FROM rr.res_table AS rt
    NATURAL JOIN rr.capability
    WHERE
      (table_type!='output' OR table_type IS NULL)
      AND standard_id='ivo://ivoa.net/std/tap'
      AND NOT EXISTS (SELECT 1 FROM fromres as fr
        WHERE rt.ivoid=fr.svcid
        AND rt.table_name=fr.table_name))

-- using WITH here to allow for a lateral union
SELECT * FROM fromtap UNION ALL
SELECT * FROM fromres) q)
```

D Changes from Previous Versions

D.1 Changes from PR-1.2-20240124

- Removed hedging language from sect. 4.5, "Vocabulary Considerations", since Vocabularies in the VO 2 is now a REC.
- Consequently, removed Appendix D ("Mandatory translations"). Ingestors should take these directly from the vocabulary (e.g., via desise).
- Now explicitly requiring ADQL 2.1 or later on the underlying TAP service.

D.2 Changes from WD-1.2-20220519

- Names in `rr.res_table` are no longer lowercased (this picks up RegTAP 1.1 erratum 1)
- Several editorial changes like slightly improved column descriptions.

D.3 Changes from REC-1.1

- Adding `stc_spatial`, `stc_temporal`, and `stc_spectral` tables and a sample query illustrating their use.
- Adding a `tap_table` view of TAP-queriable tables.
- Requiring ADQL COALESCE, ILIKE, and WITH constructs.
- Requiring an `ivo_interval_overlaps` ADQL User Defined Function.
- Including VODataService 1.2 resource types.
- `table_name` is no longer case-folded (RegTAP 1.1 Erratum 1).
- Now recommending an index on `res_table.table_utype` (for discovering EPN-TAP, LineTAP, ObsLocTAP...).

D.4 Changes from REC-1.0

- Added the `alt_identifier` table.
- Added `rights_uri` to `resource`. In `rights`, we now only take data from the first rights element as hash-joining is not reliable with free text. This technically might constitute an API change, but since we don't believe rights has (properly) been used anywhere, we still believe we are within the limits of a minor change.

- Added an xpath `/capability/interface/testQueryString` for use in `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 capabilitities. Running a validator off a RegTAP service really requires an extra table.
- Added a `mirror_url` column to `rr.interface`.
- Made type information in the schema tables more generic; we now have string, integer, real, and string+timestamp.
- Added a column `authenticated_only` in `interface` that is true when the interface cannot be used without authentication. Added this to the recommended discovery patterns.
- Recommending that when discovering standard services clients should (again) constrain `intf_role` to `std` rather than `intf_type` to `vs:ParamHTTP`. An investigation on 2019-09-01 showed that the workaround from RegTAP 1.0 is no longer necessary.
- Now requiring that services map deprecated vocabulary terms to preferred ones.
- Now requiring the data model URI as the utype of the `rr` schema.
- No longer claiming that RegTAP services do not use the *`vg:registry`* resource type any more, instead referring to RI 1.1.
- Dropping the appendix with recommended string sizes.
- Replaced inline XSLT utype maker with a link to an external resource.
- Updated example queries to match standard ids as recommended by Identifiers 2.0; also included RegTAP 1.0 erratum 1, and repaired the bad order of arguments in ivo_hashlist_has in query 10.3.

D.5 Changes from PR-2014-10-30

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

D.6 Changes from PR-20140627

- Removed reference to a future STC extension.
- Migrated to ivoatex.

D.7 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).

D.8 Changes from WD-20131203

- To match our usage with what will later be in the standards record, changed the data model identifier to ivo://ivoa.net/std/RegTAP#1.0.
- 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.

D.9 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 accound, 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.

D.10 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 `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):

Old	**New**
resource.version	resource.res_version
res_role.address	res_role.street_address
subject.*	res_subject.*
res_subject.res_subject	res_subject.res_subject
table_column.description	table_column.column_description
intf_param.description	intf_param.param_description
intf_param.use_param	intf_param.param_use
validation.level	validation.val_level

- 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 IVOID-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 explicitely deprecate multiple access URLs per interface and announce that single access URLs will be enforced in future VOResource versions.

D.11 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_detail utypes was moved to an appendix since it was too long to be included in the running text.
- Redaction for WD publication.

D.12 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_detail 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

Benson, K., Plante, R., Auden, E., Graham, M., Greene, G., Hill, M., Linde, T., Morris, D., O'Mullane, W., Rixon, G., Stébé, A. and Andrews, K. (2009), 'IVOA Registry Interfaces Version 1.0', IVOA Recommendation 04 November 2009, arXiv:1110.0513. doi:10.5479/ADS/bib/2009ivoa.spec.1104B, https://ui.adsabs.harvard.edu/abs/2009ivoa.spec.1104B.

Bray, T., Hollander, D., Layman, A., Tobin, R. and Thompson, H. S. (2009), 'Namespaces in XML 1.0 (third edition)', W3C Recommendation. http://www.w3.org/TR/2009/REC-xml-names-20091208/.

Campillo, J. J. and Demleitner, M. (2023), 'Catalogue of ADQL User Defined Functions Version 1.1', IVOA Endorsed Note 17 November 2023. https://ui.adsabs.harvard.edu/abs/2023ivoa.spec.1117C.

Clark, J. and DeRose, S. (1999), 'XML path language (XPath), version 1.0', W3C Recommendation. http://www.w3.org/TR/1999/REC-xpath-19991116.

Demleitner, M., Dowler, P., Plante, R., Rixon, G. and Taylor, M. (2012), 'TAPRegExt: a VOResource Schema Extension for Describing TAP Services Version 1.0', IVOA Recommendation 27 August 2012, arXiv:1402.4742. doi:10.5479/ADS/bib/2012ivoa.spec.0827D, https://ui.adsabs.harvard.edu/abs/2012ivoa.spec.0827D.

Demleitner, M., Gray, N. and Taylor, M. (2023), 'Vocabularies in the VO Version 2.1', IVOA Recommendation 06 February 2023. https://ui.adsabs.harvard.edu/abs/2023ivoa.spec.0206D.

Demleitner, M., Plante, R., Linde, T., Williams, R. and Noddle, K. (2016), 'IVOA Identifiers Version 2.0', IVOA Recommendation 23 May 2016, arXiv:1605.07501. doi:10.5479/ADS/bib/2016ivoa.spec.0523D, https://ui.adsabs.harvard.edu/abs/2016ivoa.spec.0523D.

Demleitner, M., Plante, R., Salgado, J., Harrison, P. and Tody, D. (2022), 'Describing Simple Data Access Services Version 1.2', IVOA Recommendation 22 February 2022. doi:10.5479/ADS/bib/2022ivoa.spec.0222D, https://ui.adsabs.harvard.edu/abs/2022ivoa.spec.0222D.

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

Demleitner, M. and Taylor, M. (2019), 'Discovering Data Collections Within Services Version 1.1', IVOA Endorsed Note 20 May 2019. doi:10.5479/ADS/bib/2019ivoa.rept.0520D, https://ui.adsabs.harvard.edu/abs/2019ivoa.spec.0520D.

Dower, T., Demleitner, M., Benson, K., Plante, R., Auden, E., Graham, M., Greene, G., Hill, M., Linde, T., Morris, D., O'Mullane, W., Rixon, G., Stébé, A. and Andrews, K. (2018), 'Registry Interfaces Version 1.1', IVOA Recommendation 23 July 2018. doi:10.5479/ADS/bib/2018ivoa.spec.0723D, https://ui.adsabs.harvard.edu/abs/2018ivoa.spec.0723D.

Dowler, P., Demleitner, M., Taylor, M. and Tody, D. (2017), 'Data Access Layer Interface Version 1.1', IVOA Recommendation 17 May 2017. doi:10.5479/ADS/bib/2017ivoa.spec.0517D, https://ui.adsabs.harvard.edu/abs/2017ivoa.spec.0517D.

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

Dowler, P., Rixon, G., Tody, D. and Demleitner, M. (2019), 'Table Access Protocol Version 1.1', IVOA Recommendation 27 September 2019. doi:10.5479/ADS/bib/2019ivoa.spec.0927D, https://ui.adsabs.harvard.edu/abs/2019ivoa.spec.0927D.

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

Gray, N., Cecconi, B., Demleitner, M., Derrière, S., Gray, N., Louys, M. and Ochsenbein, F. (2023), 'Units in the VO Version 1.1', IVOA Recommendation 15 December 2023. https://ui.adsabs.harvard.edu/abs/2023ivoa.spec.1215G.

Hanisch, R., IVOA Resource Registry Working Group and NVO Metadata Working Group (2007), 'Resource Metadata for the Virtual Observatory Version 1.12', IVOA Recommendation 02 March 2007, arXiv:1110.0514. doi:10.5479/ADS/bib/2007ivoa.spec.0302H, https://ui.adsabs.harvard.edu/abs/2007ivoa.spec.0302H.

Harrison, P., Burke, D., Plante, R., Rixon, G., Morris, D. and IVOA Registry Working Group (2012), 'StandardsRegExt: a VOResource Schema Extension for Describing IVOA Standards Version 1.0', IVOA Recommendation 08 May 2012, arXiv:1402.4745. doi:10.5479/ADS/bib/2012ivoa.spec.0508H, https://ui.adsabs.harvard.edu/abs/2012ivoa.spec.0508H.

Harrison, P., Demleitner, M., Major, B. and Dowler, P. (2018), 'XML Schema Versioning Policies Version 1.0', IVOA Endorsed Note 29 May 2018. doi:10.5479/ADS/bib/2018ivoa.spec.0529H, https://ui.adsabs.harvard.edu/abs/2018ivoa.spec.0529H.

Lagoze, C., de Sompel, H. V., Nelson, M. and Warner, S. (2002), 'The open archives initiative protocol for metadata harvesting, version 2.0'. http://www.openarchives.org/OAI/openarchivesprotocol.html.

Louys, M., Tody, D., Dowler, P., Durand, D., Michel, L., Bonnarel, F., Micol, A. and IVOA DataModel Working Group (2017), 'Observation Data Model Core Components, its Implementation in the Table Access Protocol Version 1.1', IVOA Recommendation 09 May 2017. doi:10.5479/ADS/bib/2017ivoa.spec.0509L, https://ui.adsabs.harvard.edu/abs/2017ivoa.spec.0509L.

Mantelet, G., Morris, D., Demleitner, M., Dowler, P., Lusted, J., Nieto-Santisteban, M. A., Ohishi, M., O'Mullane, W., Ortiz, I., Osuna, P., Shirasaki, Y. and Szalay, A. (2023), 'Astronomical Data Query Language Version 2.1', IVOA Recommendation 15 December 2023. https://ui.adsabs.harvard.edu/abs/2023ivoa.spec.1215M.

Plante, R., Demleitner, M., Benson, K., Graham, M., Greene, G., Harrison, P., Lemson, G., Linde, T. and Rixon, G. (2018), 'VOResource: an XML Encoding Schema for Resource Metadata Version 1.1', IVOA Recommendation 25 June 2018. doi:10.5479/ADS/bib/2018ivoa.spec.0625P, https://ui.adsabs.harvard.edu/abs/2018ivoa.spec.0625P.

Postgres Global Development Group (2013), 'PostgreSQL 9.2.1 documentation', Web page, retrieved 2023-06-27. http://www.postgresql.org/docs/9.2/static/index.html.

Taffoni, G., Schaaf, A., Rixon, G. and Major, B. (2017), 'SSO - Single-Sign-On Profile: Authentication Mechanisms Version 2.0', IVOA Recommendation 24 May 2017, arXiv:1709.00171. doi:10.5479/ADS/bib/2017ivoa.spec.0524T, https://ui.adsabs.harvard.edu/abs/2017ivoa.spec.0524T.

The Unicode Consortium (2012), 'The Unicode standard, version 6.1 core specification'. http://www.unicode.org/versions/Unicode6.1.0.