Catalogue of ADQL User Defined Functions

Version 1.2

Proposed Endorsed Note 2024-08-07

Working Group
DAL
This version
https://www.ivoa.net/documents/udf-catalogue/20240807
Latest version
https://www.ivoa.net/documents/udf-catalogue
Previous versions
EN-1.1
EN-1.0
PEN-20200806
PEN-20190925
Author(s)
Jon Juaristi Campillo, Markus Demleitner, Tamara Civera
Editor(s)
Jon Juaristi Campillo
Version Control
Revision 7439876, 2024-09-09 09:42:58 +0200

Abstract

The IVOA Astronomical Data Query Language (ADQL) has the extension facility of user defined functions (UDFs). While service operators are free to define these as they see fit, it is clearly advantageous if identical functionality is available under identical names in different services. Therefore, ADQL 2.1 has introduced the `ivo_` prefix, reserved for functions with agreed-upon semantics. This endorsed note contains this agreement and, via its updates, is the means of updating it.

Status of this document

This is an IVOA Proposed Endorsed Note for review by IVOA members and other interested parties. It is appropriate to reference this document only as a Proposed Endorsed Note that is under review and may change before it is endorsed or may not be endorsed.

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

Contents

Conformance-related definitions

The words “MUST”, “SHALL”, “SHOULD”, “MAY”, “RECOMMENDED”, and “OPTIONAL” (in upper or lower case) used in this document are to be interpreted as described in IETF standard RFC2119 (Bradner, 1997).

The *Virtual Observatory (VO)* is a general term for a collection of federated resources that can be used to conduct astronomical research, education, and outreach. The International Virtual Observatory Alliance (IVOA) is a global collaboration of separately funded projects to develop standards and infrastructure that enable VO applications.

1 Introduction

The Astronomical Data Query Language (ADQL) (Mantelet and Morris et al., 2023) has the notion of user defined functions (UDF). These provide a light-weight extension mechanism for operators of TAP services. For instance, Taylor and Mantelet et al. (2016) show how they enable the construction of HEALPix maps although ADQL 2.0 itself entirely lacks facilities to deal with them.

In order to avoid different signatures or semantics on functions offering identical or similar functionality, this document defines UDFs with names prefixed with `ivo_`. If services have functions with this names, they must work as defined here.

In version 1.0, this note documents a consensus on such functions where it has been reached within other standards or in the review of this document.

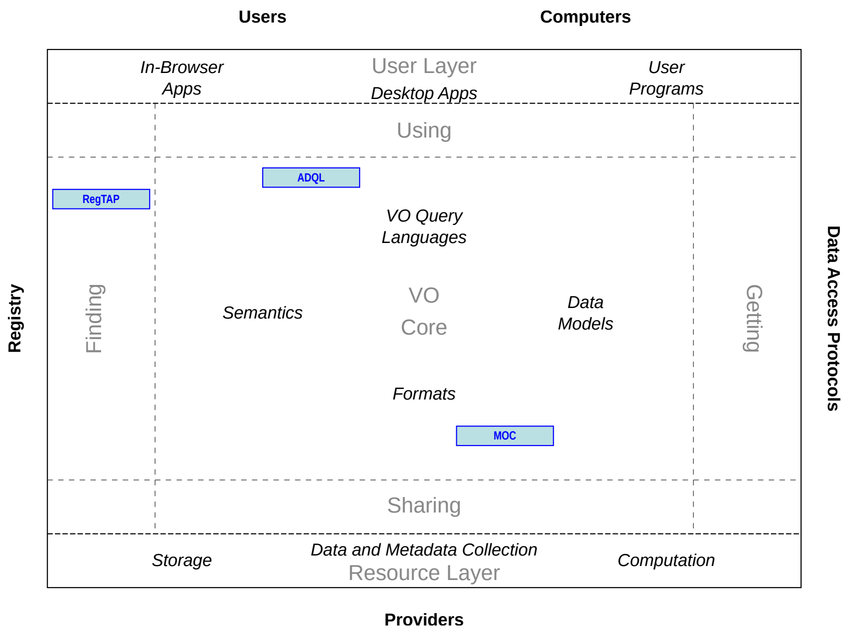

Figure 1: Architecture diagram for the UDF catalogue.

In the future, where UDFs are not naturally associated with a particular standard, they can be proposed by editing this document and starting the Endorsed Note process (Genova and Arviset et al., 2017) on the changes. In effect, this endorsed note is the management tool for the `ivo_` "namespace" in ADQL user defined functions.

Note that no function given here is required to be present in a generic TAP service (though other standards may pose such requirements; for instance, `ivo_hashlist_has` is required by both RegTAP and EPN-TAP). However, if a service implements any UDF with a name mentioned in this document, its semantics must be as specified here.

1.1 Role within the VO Architecture

Fig. 1 shows the role this document plays within the IVOA architecture (Dowler and Evans et al., 2021). It relates to the following other standards:

ADQL (Mantelet and Morris et al., 2023)
: This endorsed note defines the user defined functions with the `ivo_` prefix. While ADQL 2.0 does not treat those as special, they can be (and have been) used there already. Normative language on `ivo_` is expected to become part of ADQL 2.1.

RegTAP (Demleitner and Harrison et al., 2019)
: RegTAP first defined some of the UDFs defined here. It is expected

that later versions of RegTAP will refer to this note rather than maintaining a second definition, and deferring the actual definition of UDFs to this note should be the standard way of introducing UDFs used or required by a standard in the future.

MOC (Fernique and Nebot et al., 2022)
: The HEALPix functions defined here build on terminology introduced in the MOC specification.

2 List of IVOA user defined functions

The functions are defined through a brief human-readable description of what the function does, followed by a closer discussion of the parameters, the return value, and the authority the UDF was drawn from.

In the parameter definitions, we do not distinguish between different precisions of floating point arguments. Where we write `REAL`, the expectation is that the functions accept floating point values of any precision. Similarly, we write `TEXT` for anything sufficiently string-like, be it `CHAR(n)`, `VARCHAR(*)`, or something comparable.

Most functions are accompanied by examples, which are intended to make the effects of the functions clearer and give implementors a starting point for tests. Where examples disagree with the specification text, the text is normative.

While ADQL does not support standalone evaluation of functions, a query like

```
SELECT TOP 1 <example> AS res FROM TAP_SCHEMA.tables
```

will return one row with the function result for simple, non-aggregate functions.

2.1 HEALPix-related

In this section, order and npix are used as in the Multi-Order Coverage map (MOC) recommendation (Fernique and Nebot et al., 2022).

2.1.1 ivo_healpix_index(hpxOrder, long, lat)

Returns the index (npix) of the HEALPix cell containing the spherical point given by longitude `long` (typically, right ascension) and latitude `lat` (typically, declination) at order `hpxOrder` in NESTED numbering.

Parameters

- *hpxOrder* (`INTEGER`) – the HEALPix order to use.
- *long* (`REAL`) – longitude of the spherical point to compute the index for, in degrees.

- *lat* (`REAL`) – latitude of the spherical point to compute the index for, in degrees.

Return type `BIGINT`

Source This document, version 1.0

Examples (non-normative)

```
ivo_healpix_index(0, 1, 1)
    → 4
ivo_healpix_index(17, 1, 1)
    → 81609757711
ivo_healpix_index(17, 359, -1)
    → 73009064944
```

2.1.2 ivo_healpix_index(hpxOrder, point)

Returns the index (npix) of the HEALPix cell containing the spherical point given by `point` at order `hpxOrder` in NESTED numbering.

Parameters

- *hpxOrder* (`INTEGER`) – the HEALPix order to use.
- *point* (`POINT`) – the position to compute the index for.

Return type `BIGINT`

Source This document, version 1.0

Examples (non-normative)

```
ivo_healpix_index(0, POINT(1, 1))
    → 4
ivo_healpix_index(17, POINT('ICRS', 1, 1))
    → 81609757711
ivo_healpix_index(17, CENTROID(CIRCLE(359, -1, 3)))
    → 73009064944
```

2.1.3 ivo_healpix_center(hpxOrder, hpxIndex)

Returns a POINT corresponding to the center of the HEALPix cell with index (npix) `hpxIndex` at order `hpxOrder` in NESTED numbering.

Parameters

- *hpxOrder* (`INTEGER`) – the HEALPix order to use.

- *hpxIndex* (`INTEGER`) – the index for which to return the center position.

Return type `POINT`

Source This document, version 1.0

Examples (non-normative)

`ivo_healpix_center(0, 6)`
  → `[180., 0.]`
as a 2-array with ***xtype***=point.

`ivo_healpix_center(17, ivo_healpix_index(17, 23.4, 56.7))`
  → `[23.399843462947466, 56.70016214363192]`

`ivo_healpix_center(17, 23.2)`
  → undefined
Implementations are advised to fail when floating point numbers are passed as indices, but may choose to do some rounding.

2.2 Astrometry

2.2.1 ivo_epoch_prop_pos(ra, dec, parallax, pmra, pmdec, radial_velocity, ref_epoch, out_epoch)

Returns an ADQL POINT giving the position at `out_epoch` for an object with the six parameters at `ref_epoch`. Essentially, it will apply the proper motion and the radial velocity under the assumption of linear motion. Despite the name of the positional parameters, this is not restricted to equatorial systems, as long as positions and proper motions are expressed in the same reference frame.

Implementations must assume linear motions of both the star and the sun (i.e., not include secular aberration). No relativistic corrections at low parallaxes must be applied (which are very likely spurious for most applications of this UDF). The recommended formalism is Lindegren's "rigorous treatment" (ESA, 1997).

This way, only the difference between ref_epoch and out_epoch is relevant to the function's result. The signature was chosen for compatibility with established UDFs at major TAP service providers.

Parameters

- *ra* (`REAL`) – the object's longitude at *ref_epoch*, in degrees. NULL values here are an error.
- *dec* (`REAL`) – the object's latitude at *ref_epoch*, in degrees. NULL values here are an error.
- *parallax* (`REAL`) – the parallax at *ref_epoch* in mas. A NULL here is to be interpreted as "object is infinitely remote".

- *pmra* (`REAL`) – the tangential plane proper motion in longitude at *ref_ epoch*, in mas/yr. "Tangential plane" here means it is the temporal derivative of the longitude multiplied by the cosine of the latitude, which is what is given as proper motion in RA in almost all modern catalogues. NULLs must be interpreted as 0.
- *pmdec* (`REAL`) – the object's proper motion in latitude at *ref_ epoch*, in mas/yr. NULLs must be interpreted as 0.
- *radial_ velocity* (`REAL`) – the object's radial velocity at *ref_ epoch*, in km/s. NULLs must be interpreted as 0.
- *ref_ epoch* (`REAL`) – the epoch for which the six previous parameters are given, in Julian years. NULL values here are an error.
- *out_ epoch* (`REAL`) – the epoch for which to compute the new position. NULL values here are an error.

Return type `POINT`

Source This document, version 1.1

Examples (non-normative)

```
ivo_epoch_prop_pos(7.606083572, 11.79044105, 125,
  300, -428.8, 52.51, 2016.0, 1992.25)
```
→ [7.604061404627978, 11.7932703828279]

where [] denote an array and we use DALI representation.

```
ivo_epoch_prop_pos(7.606083572, 11.79044105, 125,
  300, -428.8, 52.51, 2016.0, 1875.0)
```
→ [7.594068213694308, 11.807251375203935]

```
ivo_epoch_prop_pos(7.606083572, 11.79044105, 125,
  300, -428.8, NULL, 2016.0, 1875.0)
```
→ [7.594079587029898, 11.807235464326332]

```
ivo_epoch_prop_pos(7.606083572, 11.79044105, 125,
  300, NULL, NULL, 2016.0, 1875.0)
```
→ [7.594080321498365, 11.790440798509207]

```
ivo_epoch_prop_pos(7.606083572, 11.79044105, NULL,
  300, -428.8, 52.51, 2016.0, 1875.0)
```
→ [7.594079587020846, 11.807235464339055]

This last value is approximative in the sense that the concrete "small" parallax has not been fixed normatively.

```
ivo_epoch_prop_pos(NULL, 11.79044105, 125,
  300, NULL, NULL, 2016.0, 1875.0)
```

→ an error.

```
ivo_epoch_prop_pos(7.606083572, 11.79044105, 125,
  300, -428.8, 52.51, NULL, 1875.0)
```

→ an error.

2.2.2 ivo_geom_transform(from_sys, to_sys, geo)

Transforms ADQL geometries (i.e., at least values of type `POINT`, `CIRCLE`, or `POLYGON`) between various reference systems. The function will return a geometry of the same type as the `geo` argument.

As specified here, `from_sys` and `to_sys` must be literal strings (i.e., they cannot be computed through expressions or be taken from database columns), although implementors are free to accept more general string expressions as an extension. The identifiers of the reference frames must be taken from the IVOA refframe vocabulary[1], where additional reference frames can be added as needed as described in Demleitner and Gray et al. (2023). Services publishing sky data are advised to implement ICRS and GALACTIC at the very least.

Implementations should list the reference frames supported in their local descriptions of this function.

Reference frame identifiers are case-sensitive.

For reproducability, all transforms should be implemented as simple rotations. This is only a rough approximation to the actual relationships between reference systems, and applications requiring high levels of precision in references frame transforms have to perform these outside of the database.

For equinox-dependent frames, we for now define default equinoxes:

- FK5: J2000.0
- FK4: J1950.0
- ECLIPTIC: J2000.0

A later version of this document may define a four-argument form of ivo_transform to support passing equinoxes or other metadata influencing the transformation.

Parameters

- *from_sys* (`string literal`) – the refframe identifier of the system `geo` is in.
- *to_sys* (`string literal`) – the refframe identifier of the system the return value is in.

[1] http://www.ivoa.net/rdf/refframe

- *geo* (`GEOMETRY`) – a `POINT`, `CIRCLE`, or `POLYGON` (accepting and transforming additional types is permitted) to transform from `from_sys` to `to_sys`.

Return type The function returns a value of the same type as `geo`

Source This document, version 1.2

Examples (non-normative)

```
ivo_geom_transform('ICRS', 'GALACTIC', POINT(120, -23))
    → [241.121831182, 3.62785788087]
```
in limited-precision DALI representation, where [] denote an array.

```
ivo_geom_transform('FK4', 'GALACTIC', CIRCLE(189.70303055, 22.96240006, 3))

    → [275.00025405, 84.99996154, 3].
ivo_geom_transform('FK4', 'GALACTIC', CIRCLE(189.70303055, 22.96240006, 3))

    → [275.00025405, 84.99996154, 3].
ivo_geom_transform('GALACTIC', 'FK5', POLYGON(5, 3, 4, 4, 4.5, 2))

    → [266.44721, -23.107310, 264.95088, -23.434037, 267.09936, -24.051919].
```

2.3 Dates and Times

2.3.1 ivo_to_jd(d)

Converts a database timestamp to a Julian Date as a floating point number. This is naive; no corrections for timezones, let alone time scales or the like are done. Users can thus not expect this to be good to second-precision unless they are careful in the construction of the timestamp.

Parameters

- *d* (`TIMESTAMP literal`) – a SQL timestamp.

Return type `REAL`

Source This document, version 1.2

Examples (non-normative)

```
ivo_to_jd(CAST('2000-01-02T12:00:00' AS TIMESTAMP))
    → 2451546.0
```

```
ivo_to_jd('1910-12-31T06:00:00')
    → 2419036.75
```
While not strictly necessary for compliance, implementations SHOULD accept reasonable

strings – primarily, DALI-style ISO dates – as arguments, in particular when they do not implement the optional CAST construct.

```
ivo_to_jd(NULL)
```
→ NULL

2.3.2 ivo_to_mjd(d)

Converts a database timestamp to a Modified Julian Date as a floating point number. This is naive; no corrections for timezones, let alone time scales or the like are done. Users can thus not expect this to be good to second-level precision unless they are careful in the construction of the timestamp.

Parameters

- *d* (`TIMESTAMP literal`) – a SQL timestamp.

Return type `REAL`

Source This document, version 1.1

Examples (non-normative)

```
ivo_to_mjd(CAST('2000-01-02T12:00:00' AS TIMESTAMP))
```
→ `51545.5`

```
ivo_to_mjd('1910-12-31T06:00:00')
```
→ `19036.25`

While not strictly necessary for compliance, implementations SHOULD accept reasonable strings – primarily, DALI-style ISO dates – as arguments, in particular when they do not implement the optional CAST construct.

```
ivo_to_mjd(NULL)
```
→ NULL

2.4 Text-Related

2.4.1 ivo_string_agg(expression, delimiter)

An aggregate function returning all values of `expression` concatenated with `delimiter`.

Parameters

- *expression* (`TEXT`) – a SQL expression giving the strings to concatenate. The expression may be NULL, in which case the row does not leave a trace in the result string.
- *delimiter* (`TEXT`) – a string used to concatenate the values of `expression` in each group.

Return type `TEXT`

Source RegTAP 1.0

Examples (non-normative)

```
SELECT ivo_string_agg(table_name, '␣<//>␣')|
FROM TAP_SCHEMA.tables
WHERE table_name ILIKE 'tap_schema.%'
```

→ `'tap_schema.columns <//> tap_schema.schemas <//> ta...'`

Of course, the actual value depends on the contents of the TAP schema on the service used to run the query.

2.4.2 ivo_nocasematch(value, pattern)

Evaluates `value ILIKE <pattern>` pattern, i.e., pattern is defined as for the SQL LIKE operator, but the match is performed case-insensitively. Returns 1 if the pattern matches, 0 otherwise.

Databases processing non-ASCII should perform case folding according to algorithm R2 in section 3.13, "Default Case Algorithms" of the Unicode Standard (The Unicode Consortium, 2012).

Please note that in ADQL versions higher than ADQL 2.1, the ILIKE operator should be used instead.

Parameters

- *value* (`TEXT`) – a string-valued SQL expression.
- *pattern* (`TEXT`) – a SQL pattern for LIKE evaluation (i.e., underscore is any character, percent zero or more arbitrary characters).

Return type `INTEGER`

Source RegTAP 1.0

Examples (non-normative)

`ivo_nocasematch('abcabc', 'ab%')`
→ 1

`ivo_nocasematch('ABcabc', 'ab__bc')`
→ 1

`ivo_nocasematch('abcabc', '%BC')`
→ 1

`ivo_nocasematch('abcabc', 'ABCABC')`
→ 1

`ivo_nocasematch('abcabc', 'abcab')`
→ 0

2.4.3 ivo_hasword(haystack, needle)

Returns 1 if all tokens from the string `needle` are contained (in some sense) in the string `haystack`, 0 otherwise. This is intended to support somewhat "Google-like", soft string matches. This specification does not precisely specify what "token" exactly means and whether stemming or any other normalisation should be performed, except that matching must be case-insensitive within ASCII, and that the minimal token definition is a continuous run of ASCII characters. Implementors are encouraged to attempt a reasonable approximation to what Web search engines do.

Parameters

- *needle* (`TEXT`) – a string to locate in `haystack`.
- *haystack* (`TEXT`) – text to match `needle` in.

Return type `INTEGER`.

Source RegTAP 1.0

Examples (non-normative)

```
ivo_hasword('Miller and Urey have', 'miller')
```
→ 1

```
ivo_hasword('shown that a primordial', 'show')
```
→ 1

This could also be 0 on a platform that does not perform English-language stemming.

```
ivo_hasword('shown that a primordial', 'soup')
```
→ 0

```
ivo_hasword('shown that a primordial', 'shown primordial')
```
→ 1

All tokens from needle are in haystack.

```
ivo_hasword('shown that a primordial', 'shown soup')
```
→ 0

One token from needle is missing in haystack.

2.4.4 ivo_hashlist_has(hashlist, item)

The `hashlist` argument is a list of words not containing the hash sign (#), concatenated by hash signs; the `item` argument is a string not containing a hash sign. The function returns 1 if, compared case-insensitively, the second argument is in the list of words encoded in the first argument, 0 otherwise. In case the second argument does contain a hash sign, the function must return 0.

Parameters

- *hashlist* (`TEXT`) – a string containing hash-separated terms.
- *item* (`TEXT`) – a string containing a single term not containing a hash.

Return type `INTEGER`

Source RegTAP 1.0

Examples (non-normative)

`ivo_hashlist_has('red#green#blue', 'red')`
→ 1

`ivo_hashlist_has('Red#green#blue', 'red')`
→ 1

`ivo_hashlist_has('red#green#blue', 're')`
→ 0

`ivo_hashlist_has('red#green#blue', 'red#green')`
→ 0

2.5 Statistics

2.5.1 ivo_histogram(val, lower, upper, nbins)

This aggregate function returns a histogram of val with `nbins` + 2 elements. Assuming 0-based arrays, `results[0]` contains the number of underflows (i.e., `val` < `lower`), `result[nbins+1]` the number of overflows. Elements 1 . . . `nbins` are the counts in `nbins` bins of width (`upper` − `lower`)/`nbins`. Clients will have to convert back to physical units using some external communication, as there currently is no (meta-) data on the lower limit or the bin size in the TAP response.

When val is NULL, it counts as an underflow.

Parameters

- *val* (`REAL`) – the value to bin. This can be any expression suitable in aggregate functions.
- *lower* (`REAL`) – the lower limit of the histogram (anything smaller will end up in bin 0).
- *upper* (`REAL`) – the upper limit of the histogram (anything larger will end up in bin nbins+1).
- *nbins* (`INTEGER`) – the number of “natural” bins in the histogram. The returned array will have two additional cells for under- and overflows.

Return type `INTEGER[]`

Source This document, version 1.1

Examples (non-normative)

```
SELECT ivo_histogram(parallax, 0.5, 3.5, 8)
FROM gaiadr3.nss_vim_fl
```

→ 6 245 287 162 92 33 22 10 7 6

on the TAP service at https://gaia.ari.uni-heidelberg.de/tap.

```
SELECT ivo_histogram(LOG10(n_star+1), 3, 5, 5) FROM mwsc.main
```

→ 781 41 329 1239 1107 260 27

on the TAP service at http://dc.g-vo.org/tap.

2.5.2 ivo_normal_random(mu, sigma)

Returns a random number drawn from a normal distribution

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{1}$$

Implementation note: Few databases at this point have built-in facilities for drawing random numbers from non-uniform distributions. In practice, an implementation of the type

```
(((random()+random()+random()+random()+random()
  +random()+random()+random()+random()+random()-5
)*sigma)+mu)
```

has been giving sufficiently good results and adequate runtime behaviour. Of course, better approximations are preferred.

Parameters

- *mu* (`REAL`) – the μ in eq. (1)
- *sigma* (`REAL`) – the σ in eq. (1)

Return Type REAL

Source This document, version 1.2

Examples (non-normative)

```
ivo_normal_random(10, 5), ivo_normal_random(10, 5)
```

→ 3.4275427, 7.839408

(or, really, almost anything else).

2.6 Convenience Functions

2.6.1 ivo_interval_overlaps(a1, b1, a2, b2)

The function returns 1 if the interval $[a1 \ldots b1]$ overlaps with the interval $[a2 \ldots b2]$. For the purposes of this function, the case $a1 = b2$ or $a2 = b1$ is treated as overlap. The function returns 0 for non-overlapping intervals.

Parameters

- *a1* (`NUMERIC`) – the lower bound of the first interval.
- *b1* (`NUMERIC`) – the upper bound of the first interval.
- *a2* (`NUMERIC`) – the lower bound of the second interval.
- *b2* (`NUMERIC`) – the upper bound of the second interval.

Return type `INTEGER`

Source Originally proposed by the roadmap for STC discovery (Demleitner, 2018); standardised in this document, version 1.0.

Examples (non-normative)

`ivo_interval_overlaps(-3, -2, 2, 3)`
  $\rightarrow 0$

`ivo_interval_overlaps(-2, 55.3, 2, 3)`
  $\rightarrow 1$

`ivo_interval_overlaps(2, 3, -2, 55.3)`
  $\rightarrow 1$

`ivo_interval_overlaps(-2, 55.3, 3, 2)`
  $\rightarrow 0$

In accordance with common RDBS semantics, intervals with lower bound > upper bound never overlap with anything.

`ivo_interval_overlaps(-2, 2, 2, 3)`
  $\rightarrow 1$

Relying on the special case of identical first upper vs. second lower bound is probably not a good idea for floating point numbers that do not have a short and finite binary representation.

2.6.2 ivo_simbadpoint(identifier)

Queries Simbad (Wenger and Ochsenbein et al., 2000) for a position of `identifier` and returns a POINT for that position. This is mainly a convenience function, as `identifier` must be a literal (as opposed to a column reference or expression). For mass resolution of identifiers, Simbad's own TAP service should be used.

Parameters

- *identifier* (`string literal`) – A Simbad-resolvable identifier.

Return type `POINT`

Source This document, version 1.2

Examples (non-normative)

`ivo_simbadpoint('GJ 699')`
→ `[269.452076958619, 4.69336496657667]`
(where Simbad returns estimated current positions and hence return values will change over time – significantly so for the present example, Barnard's star).

`ivo_simbadpoint('an invalid identifier')`
→ an error
The error message should ideally state that the identifier passed in (which ought to be included in the message) cannot be resolved by Simbad.

2.7 Celestial Conversion Functions

2.7.1 ivo_hms_to_degrees(hms)

This function converts hms (hours, minutes, seconds) to degrees format. This is useful for right ascension (RA) conversion, since hms is fairly standard method of presenting RA coordinates.

Parameters

- *hms* (`TEXT`) – hms to be converted. The following syntaxes must be accepted: 'h:m:s', 'hhmmss' and 'hHmMsS'. Hours and minutes should be integer and seconds can be decimal. The function returns NULL on unparsable input.

Return type `REAL`

Examples (non-normative)

`ivo_hms_to_degrees('3:34:45.4')`
→ 53.6891667

`ivo_hms_to_degrees('3h34m45.4s')`
→ 53.6891667

`ivo_hms_to_degrees('3H34M45.4S')`
→ 53.6891667

2.7.2 ivo_dms_to_degrees(dms)

This function converts dms (degrees, minutes, seconds) to degrees format. This is useful for declination conversion (DEC) conversion, since dms is fairly standard method of presenting DEC coordinates. The function returns NULL on unparsable input.

Parameters

- *dms* (`TEXT`) – dms to be converted. The following syntaxes must be accepted: 'd:m:s', 'ddmmss' and 'dDmMsS'. Degrees and minutes should be integer and seconds can be decimal. A preceding + or - sign is allowed.

Return type `REAL`

Examples (non-normative)

`ivo_dms_to_degrees('42:9:6.768')`
→ 42.15188

`ivo_dms_to_degrees('42d9m6.768s')`
→ 42.15188

`ivo_dms_to_degrees('42D9M6.768S')`
→ 42.15188

`ivo_dms_to_degrees('-00:15:00')`
→ -0.25

`ivo_dms_to_degrees('+00:15:00')`
→ 0.25

`ivo_dms_to_degrees('-0d15m0s')`
→ -0.25

`ivo_dms_to_degrees('+0d15m0s')`
→ 0.25

A Changes from Previous Versions

A.1 Changes from EN-1.1

- Added `ivo_geom_transform` as astrometry function (gavo, esa)
- Added `ivo_simbadpoint` as convenience function (gavo, esa)
- Added `ivo_normal_random` as a statistics function (gavo, esa)
- Added `ivo_to_jd` and `ivo_to_mjd` as a datetime functions (gavo, esa, cefca-oaj)

- Added `ivo_hms_to_degrees` and `ivo_dms_to_degrees` as a celestial conversion functions (cefca-oaj)

A.2 Changes from EN-1.0

- Added `ivo_epoch_prop_pos` (gavo, esa)
- Added `ivo_histogram` (gavo, ari)
- Removed the list of third-party UDFs; its proper maintenance would have become too tedious.

A.3 Changes from PEN-20200806

Editorial changes only.

A.4 Changes from PEN-20190925

- Added Examples
- Removed `ivo_apply_pm`, since the right specification needs a bit more thought.
- Removed `ivo_interval_has`; this is still experimental, not the least because interval in ADQL is somewhat underspecified.

References

Bradner, S. (1997), 'Key words for use in RFCs to indicate requirement levels', RFC 2119. http://www.ietf.org/rfc/rfc2119.txt.

Demleitner, M. (2018), 'A roadmap for space-time discovery in the VO registry', IVOA Note. http://ivoa.net/documents/Notes/Regstc/20180208/NOTE-regstcnote-1.0-20180115.html.

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., Harrison, P., Molinaro, M., Greene, G., Dower, T. and Perdikeas, M. (2019), 'IVOA Registry Relational Schema Version 1.1', IVOA Recommendation 11 October 2019. doi:10.5479/ADS/bib/2019ivoa.spec.1011D, https://ui.adsabs.harvard.edu/abs/2019ivoa.spec.1011D.

Dowler, P., Evans, J., Arviset, C., Gaudet, S. and Technical Coordination Group (2021), 'IVOA Architecture Version 2.0', IVOA Endorsed Note 01 November 2021. doi:10.5479/ADS/bib/2021ivoa.spec.1101D, https://ui.adsabs.harvard.edu/abs/2021ivoa.spec.1101D.

ESA (1997), 'The HIPPARCOS and TYCHO catalogues. Astrometric and photometric star catalogues derived from the ESA HIPPARCOS Space Astrometry Mission'. https://ui.adsabs.harvard.edu/abs/1997ESASP1200.....E.

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.

Genova, F., Arviset, C., Demleitner, M., Glendenning, B., Molinaro, M., Hanisch, R. J. and Rino, B. (2017), 'IVOA Document Standards Version 2.0', IVOA Recommendation 17 May 2017. doi:10.5479/ADS/bib/2017ivoa.spec.0517G, https://ui.adsabs.harvard.edu/abs/2017ivoa.spec.0517G.

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.

Taylor, M. B., Mantelet, G. and Demleitner, M. (2016), 'All of the Sky: HEALPix Density Maps of Gaia-scale Datasets from the Database to the Desktop', *ArXiv e-prints* , arXiv:1611.09190. http://ads.ari.uni-heidelberg.de/abs/2016arXiv161109190T.

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

Wenger, M., Ochsenbein, F., Egret, D., Dubois, P., Bonnarel, F., Borde, S., Genova, F., Jasniewicz, G., Laloë, S., Lesteven, S. and Monier, R. (2000), 'The SIMBAD astronomical database. The CDS reference database for astronomical objects', A&AS **143**, 9–22, arXiv:astro-ph/0002110. doi:10.1051/aas:2000332, https://ui.adsabs.harvard.edu/abs/2000A&AS..143....9W.