*I*nternational

*V*irtual

*O*bservatory

*A*lliance

IVOA Parameterised Query Language

Version 0.1

IVOA Internal Working Draft 2009 February 12

This version:

PQL-0.1-20090212

Latest version:

Not yet issued

Previous version(s):

Authors:

TBD

Contributors:

TBD

Abstract

This document describes the Parameterised Query language (PQL). PQL has been developed as part of various Data Access Layer (DAL) services. This document formalises the syntax and meaning of PQL as a generic parameter-based query language for querying astronomical data services.

Status of This Document

This is a working draft internal to the DAL-WG.

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

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

Acknowledgements

"Ack here, if any"

Contents

1 Introduction

The Parameterised Query language (PQL) is a language used by the International Virtual Observatory Alliance (IVOA) to represent simple astronomy queries posted to VO services.

PQL is based on past use and parameter-based query interfaces in DAL services such as Simple Image Access (SIA), Simple Spectral Access (SSA), and Simple Cone Search (SCS). Parametric queries are simple to express and to implement for cases where the data model is sufficiently well defined and adequate for the data to be queried, hiding many of the details required to pose and evaluate the query. In this sense PQL provides a higher level of abstraction that richer and more detailed languages such as the Astronomical Data Query Language (ADQL).

2 Parameterised Query Language (PQL)

2.1 General Parameter Rules

2.1.1 Single-Valued Parameters

Parameters which are single-valued always specify equality. The value may not use any of the reserved characters listed above unless these are URL-encoded (REF?).

2.1.2 Multi-Valued Parameters

Parameters which are multi-valued (list valued, such as positions) use the comma (“,”) as the separator between successive items in the list. Embedded white space is not permitted. If a parameter value includes a space or comma, it **must** be escaped using the URL encoding rules (see section Error: Reference source not found and IETF RFC 2396 [5]).

In some lists, individual entries may be empty, and **should** be represented by the empty string. Thus, two successive commas indicate an empty item, as does a leading comma or a trailing comma. An empty list **should** be interpreted either as a list containing no items, or as a list containing a single empty item, depending upon the context.

2.1.3 Range-Valued Parameters

Parameters thats specify a range of values use the forward slash (“/”) character as the separator between elements of the range specification (as in the ISO 8601 date specification after which this convention is patterned). For example,a range consisting of all values from 5E-7 to 8E-7 inclusive would be:

```
Example: 5E-7/8E-7
```

If a third field is specified it is a step size for traversing the indicated range. If a parameter permits a step size the semantics of the step size are defined by the specific parameter.

An open range **may** be specified by omitting either range value. If the first value is omitted the range is open toward lower values. If the second value is omitted the range is open toward higher values. Omitting both values indicates an infinite range which accepts all values. An open range which accepts all values less than or equal to 5:

```
Example: /5
```

Range values can be used with parameters which specify numeric and date values only. (TBD: Why not all data types that are fully ordered? e.g. numbers, dates, and strings but not regions)

2.1.4 Qualifiers

If specified by the definition of a particular parameter, a single-valued parameter, range, or list **may** be qualified by appending the character ";" (semicolon) followed by a qualifier string. This could be used to specify an alternate coordinate system, e.g.

```
Example: 180.0,1.0;GALACTIC
```

could specify a position in galactic coordinates. In some cases, multiple semicolons may be used to delimit separate sub-lists or clauses within the parameter value.

2.1.5 Combinations

List and range syntax may be combined, e.g., to indicate a list of scalar or range-valued parameter values. Such a range list may be ordered or unordered, and may contain either numeric or string data. An ordered list is one which requires values to be processed in a specified order, and to ensure this the range list is sorted or ordered by the service as necessary before being used. It is the responsibility of the service to sort an ordered range list, hence the client may input ranges or range values in any order for an ordered range list and the result **must** be the same. The sequence in which items in an unordered list occur on the other hand is significant, as since there is no intrinsic ordering for the list which can be enforced by the service, items will be processed by the service in the order they are input by the client.

2.1.6 Symbolic Values

The value for any parameter can be a symbolic value rather than a literal value (constant). The @ character is used to denote a symbolic value:

```
Example: POS=@something
```

The meaning or interpretation of the symbolic value is defined by the service that accepts PQL.

2.1.7 Missing or null-valued parameters

If a parameter is not included in a query its value is unset; no value has been specified. If a parameter is given a null value, e.g., “*POS*=”, the parameter value has been set and the value is the null string. The interpretation of such an input is defined by the service that accepts PQL input and may or may not be an error.

2.1.8 Case of parameters

Parameter names **must not** be case sensitive, but parameter values **must** be so. In this document, parameter names are typically shown in uppercase for typographical clarity, not as a requirement.

2.1.9 Order and cardinality of parameters

Parameters in a request **may** be specified in any order.

When request parameters are duplicated with conflicting values, the response from the service is undefined. The service **may** reject the request or it **may** pick one value for for the parameter. Clients **should not** repeat parameters in a request.

2.2 Standard Parameters

A service which implements parametric queries on data must do so using the parameters defined in this section.

2.2.1 POS, SIZE

The POS and SIZE parameters provide an easy to use, optimized facility for performing spatial queries of astronomical data, as is used in the SIA and CSC protocols. Spatial queries are supported only for content which contain positional information; most astronomical data services serve content where this is true.

POS and SIZE define a circular search region in the specified coordinate system (default ICRS). A service which supports must support the POS and SIZE parameters, and implement them as a query constraint for tables containing records tagged with spatial positions. If POS and SIZE cannot be applied to the referenced table an error should be returned.

The coordinate values for POS are specified in list format (comma separated) with no embedded white space, as defined in section 2.1.2.

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

The POS parameter defaults to right-ascension and declination in decimal degrees in the ICRS coordinate system. A coordinate system reference frame may optionally be specified to indicate a spatial coordinate system other than ICRS. The reference frame is specified as a list format modifier, with the acceptable values as defined by Table 3 (standard reference frames) in STC [4].

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

The SIZE parameter specifies the diameter of the search region input in decimal degrees.

```
Example: SIZE=0.05
```

A valid query does not have to specify a SIZE parameter. If SIZE is omitted in a positional query, the service should supply a default value intended to find nearby objects which are candidates for a match to the given object position, taking into account the spatial resolution of the data.

2.2.2 REGION

The REGION parameter provides a more general spatial search than can be defined using POS and SIZE. The value of REGION **must** be a STC/S (REF) region specifier, e.g.

```
Example: REGION=Ellipse ICRS 148.9 69.1 2.0 4.0 32.7
```

In the example above the embedded spaces are shown for clarity, but in real use they **must** be URL encoded.

If POS,SIZE and REGION are all specified in the same query, they both apply. That is, REGION **must** be used as a mask to further qualify the circular region specified by POS and SIZE. This is most useful for multi-position queries (TBD), where a large table of possible search positions may include positions outside the desired search region. In this case REGION specifies the sub-region of the referenced table to be used. This allows large tables to be used in a multi-position query. In particular it permits a cross match of two data tables (e.g., two large astronomical catalogs) to be performed in a single operation, restricting the spatial portion of the cross match to the mask region.

[Note: PQL defines the use of symbolic values (above) and it would be up to a service like TAP to specify this kind of usage. -Ed.]

2.2.3 BAND

The BAND parameter specifies a constraint on the energy value or coverage of the astronomical content. This is generally applicable for queries to data services, where the data is described with some representative energy value or range of values.

The BAND parameter defines a single value or range of values of energy in the specified representation (TBD: default is wavelength in meters). For example, to find content at 550nm (e.g. including photons of that energy):

```
Example: BAND=5.5E-9
```

The value of the BAND parameter may be qualified to indicate an alternate energy representation. For example,

```
Example: BAND=2.0E9/3.0E9;FREQ
```

would specify an frequency range of 2.0 to 3.0 GHz. Allowed qualifiers are WAVE (wavelength in meters), FREQ (frequency in Hz), and ENERGY (energy in eV).

[NOTE: BAND was not in TAP 0.31, but has been included from SSA for completeness. This use of the qualifier described above has been extrapolated from the POS,SIZE material above; SSA

uses the qualifier to specify the rest frame - "source" or "observer" - rather than to specify an alternate representation as above. Both seem valuable and if there can be multiple qualifiers we could have both; otherwise we need to make decisions: compatibility vs symmetry. -Ed.]

2.2.4 TIME

The TIME parameter specifies a constraint on the time value or coverage of the astronomical content. This is generally applicable for queries to data services, where the data is described with some representative time value or range of values (the observation date, for example).

The TIME parameter defines a single value or range of values of time in the specified representation (TBD: default is ISO8601 for compatibility with SSA). For example, to find data collected in January 2009:

```
Example: TIME=2009-01-01T00:00:00/2009-01-31:23:59:59
```

The value of the BAND parameter may be qualified to indicate an alternate time representation. For example,

```
Example: TIME=53200.0/53210.0;MJD
```

would specify a range of dates using Modified Julian Date. Allowed qualifiers are ISO (ISO8601 date-time format) and MJD (Modified Julian Date).

[NOTE: As above, TIME has been included from SSA and the use of qualifiers has been extrapolated from POS,SIZE. SSA only allows ISO8601, while the qualifier use here allows clients to use Modified Julian Date as well. -Ed]

2.2.5 SELECT

The SELECT parameter specifies the fields to be returned by the query, specified either as a comma delimited list of field names, or optionally by specifying one of the reserved values *$STD* (to return only the standard or "primary" fields), or *$ALL* (to return all table fields).

```
Example: SELECT=ra,dec,flux
Example: SELECT=$ALL
```

By default only the $STD fields are returned. The "primary" fields are specified on a per-table basis, and define a subset of the most important table fields chosen by the service implementor. This is used to provide a more readable view of very wide tables. The service **must** permit *$STD* and *$ALL* to be input without error, but is not required to actually use them to adjust the view of the table. If no $STD view is defined for a table the service **should** ignore *$STD* and merely return all table fields.

[TBD: ALL and STD are symbolic values and could use the symbolic value mechanism in 2.1.6 instead of introducing another reserved symbol.]

The names of available fields can be specified by the service specification directly (e.g. for services like SIA and SSA with a data model) or obtained from service metadata (for generic services with no inherent astronomical data model).

2.2.6 FROM

The FROM parameter indicates the target of the query. Only a single value is allowed.

```
Example: FROM=hdfv2
```

It is up to the service specification to decide if there is a sensible default value; in services with a data model (e.g. SSA) there would be a valid default and FROM would not be required. In generic services with no data model (e.g. TAP, where targets means tables) there may be multiple targets to chose from and FROM would be required.

2.2.7 WHERE

The WHERE parameter is used to specify generic optional filtering constraint(s) to be applied to the query target and determine which records are returned. By default all trecords are returned.

The WHERE parameter **may** be combined with other query constraints such as POS and REGION to further refine the query.

The syntax of the WHERE parameter value is a simple sequence of equality or range constraints delimited by semicolons, with the field name and value elements of an individual constraint separated by a comma.

```
Example: WHERE=observer,*smith*;z,1.5/2.2
```

This specifies two table field constraints: the field “observer” **must** contain the case-insensitive substring “smith” (hence the wildcards), and the field “z” must be in the range 1.5 to 2.2 inclusive.

> *[NOTE: In the WHERE parameter, the semi-colon is used as the list delimiter to separate constraints, while in the range-list section above comma separates list items and semi-colon separates an item from a qualifier. That is, in the BNF below field-list uses semi-colon and the other lists use comma. Clearly the separators need to be different, but maybe re-using the qualifier separator is not such a good idea... just in case we want to allow (now or in future) the use of qualifiers in the WHERE parameter. -Ed.]*

The field names come from the service specification or service metadata as described in 2.2.5 .

The WHERE syntax has deliberately been kept simple as ADQL already provides a fully general expression evaluation capability, which should be used to support advanced query capabilities. Each constraint applies to a single table field; multiple constraints on the same table field are allowed. The constraints have an AND relationship, hence all must evaluate to true for a table row to satisfy the

WHERE condition. Individual constraints may be negated to construct more complex expressions.

The syntax chosen is intended to be easy to compose, easy and unambiguous for a service to parse and map to a SQL back end or otherwise evaluate (a conventional rule-based parser is not required). It was also chosen to be consistent with similar usage in other data access services, e.g., in the use of range-list syntax (2.1.2) for the WHERE expression. An effort has been made to define a minimal set of meta-characters so as to minimize the need for URL encoding – specifically not using reserved characters in the URI and URL query string syntax (e.g. =, &, ?, #). Most simple expressions should not require URL encoding, e.g., if typed interactively into a Web browser, allowing the simplest Web tools to be easily used for basic queries.

A partial BNF for the WHERE expression is as follows:

```
<where-expr>    ::= <field-list>

<field-list>    ::= <field-expr> [ ';' <field-list> ]

<field-expr>    ::= <field> ',' ['!'](<list> | "null")

<list>          ::= <numeric-list> | <string-list> | <date-list>

<numeric-list>  ::= <number> [ ',' <numeric-list> ]

<string-list>   ::= <string> [ ',' <string-list> ]

<date-list>     ::= <date> [ ',' <date-list> ]
```

- Where we have not attempted to detail the BNF for the numeric, string, and date tokens. Some additional notes follow.
- Each field expression defines a constraint on the named table field.
- Field expressions are of the form <field-name>','<value> (meaning field-name=value), where <value> is a single value, a range, or a list of single values or ranges all of the same type. Constraint expressions within the overall WHERE expression are combined with a logical AND operation. Values within a range-list are combined with a logical OR operation, i.e. a range or list for a specific field gives a list of acceptable values.
- A parameter value may optionally be prefixed with '!' (exclamation) to negate the sense of the entire clause.
- The special value "null" indicates a null-valued field. For example "flux,!null" is true only if field "flux" has a non-null value.
- A <date> conforms to ISO8601 date syntax, e.g., "2007-04-05T14:30".
- A <number> token is any legal integer or floating point number optionally preceded by '+' or '-'.
- A <string> token is any token which is not a number or date, or any sequence of characters which is quoted using single quotes.
- While accumulating a string token, anything quoted in single quotes is literally included in the string, otherwise (where case-insensitive context applies),

characters are converted to lower case for use in case-insensitive comparisons. Quoted characters are treated in a case sensitive fashion. Any metacharacter other than the quote character may be quoted to include it within a token. A single quote may be included within a string by quoting it (that is, three single quotes in sequence). Quotes used within a string token do not delimit the token.

- For string-valued fields the constraint is a case-insensitive simple pattern, with "*" matching zero or more characters. Absent any use of "*", the entire string must match. Hence "obj,m31" specifies that the value of field "obj" must match "m31" exactly, except for case. To force a case sensitive match the case sensitive characters must be quoted.
- For numeric or date values the constraint is either a single value or a range, using "/" as the range delimiter (range syntax is not supported for strings). Both open and closed ranges can be specified, e.g., "5/" specifies an open range equivalent to "greater than or equal to 5", whereas "5/9" means "5 to 9 inclusive".
- Spaces may be embedded to improve readability, but if so they must be URL encoded as "%20".

Field names or value expressions must be quoted if they contain any special characters (e.g., semicolon, comma, forward slash, asterisk). The single quote is used to avoid conflict with double quote which is often used to quote the entire URL string.

As a more complex example of WHERE usage consider the following somewhat contrived expression (with extra spaces for readability here):

```
WHERE=vmag,4.5/5.5;     imag,4.5/;     bmag,/5.5;     flag,4,5,6;
jmag,4.5/5.5,/3.0,9.0/;   name,*Lon*;   kmag,4.5/5.5;   flux,null;
last,1
```

The equivalent SQL WHERE clause would be the following:

```
vmag between 4.5 and 5.5
and imag >= 4.5
and bmag <= 5.5
and (flag = 4 or flag = 5 or flag = 6)
and (jmag between 4.5 and 5.5 or jmag <= 3.0 or jmag >= 9.0)
and name like '%Lon%'
and kmag between 4.5 and 5.5
and flux is null
and last = 1
```

Note the special treatment of the jmag constraint; the list of ranges are combined with the OR operator while the jmag constraint itself is combined with the others with the AND operator.

2.3 Numeric and boolean values

Integer numbers **must** be represented in a manner consistent with the specification for integers in *XML Schema Datatypes* [10]. This document indicates explicitly where an integer value is mandatory. Real numbers **must** be represented in a manner consistent with the specification for double-precision numbers in *XML Schema Datatypes*. This representation allows for integer, decimal and exponential notations. A real value is allowed in all numeric fields defined by this document unless the value is explicitly restricted to integer.

Sexagesimal formatting is generally not permitted other than in ISO 8601 formatted time strings.

Positive, negative and zero values are allowed unless explicitly restricted by a service specification making use of PQL.

Boolean values must be represented in a manner consistent with the specification for Boolean in XML Schema Datatypes. The values “0” and “false” are equivalent. The values “1” and “true” are equivalent. Absence of an optional value is equivalent to logical false.

3 Use with HTTP (informative)

An HTTP service which accepts PQL as input is constrained by the general rules for use of HTTP, which are contained in IETF RFC documents. This section collates some of issues in using PQL with such services. For authoritative specifications, please refer to the original RFCs.

The PQL parameters described in this document may be mapped directly to HTTP request parameters in the query string portion of the URL (HTTP GET) or included in the request (HTTP POST). As noted above, it may not be necessary to URL encode the parameter values in all cases, but it is generally good practice to do so.

3.1.1 Reserved characters in HTTP GET URLs

The URL specification (IETF RFC 2396 [5]) reserves particular characters as significant and requires that these be escaped when they might conflict with their defined usage. This document explicitly reserves several of those characters for use in the query portion of TAP requests. When the characters “?”, “&”, “=”, “,” (comma), “/”, and “;” appear in one of the roles defined in Table 1, they **must** appear literally in the URL. When those characters appear elsewhere (for example, in the value of a parameter), they should be encoded as defined in IETF RFC 2396. The server **must** be prepared to decode any character escaped in this manner.

Table 1 — Reserved characters in HTTP URLs

Character	Reserved usage
?	Separator indicating start of the URL query string
&	Separator between parameters in the query string

=	Separator between name and value of a parameter
#	Separator indicating start of a URL fragment (anchor?)
, / ;	Separator between individual values in range or list parameters

For example, while PQL does not specify any use for the fragment or anchor (#) separator, any parameter value contains this character must be URL encoded to be legally included in a URL.

4 References

[1] I. Ortiz, J. Lusted, P. Dowler, A. Szalay, Y. Shirasaki, M. Nieto- Santisteban, M. Ohishi, W. O'Mullane, P. Osuna, VOQL-TEG & VOQL-WG, *IVOA Astronomical Data Query Language version 2,* IVOA recommendation 30th October 2008. http://www.ivoa.net/Documents/REC/ADQL/ADQL-20081030.pdf

[2] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels,* IETF RFC 2119. http://www.ietf.org/rfc/rfc2119.txt

[3] A. Rots, *Space-Time Coordinate Metadata for the Virtual ObservatoryVersion 1.33*, IVOA Recommendation 30 October 2007. http://www.ivoa.net/Documents/REC/DM/STC-20071030.html

[4] T. Berner-Lee, R. Fielding L. Masinter, *Uniform Resource Identifiers (URI): Generic Syntax,* IETF RFC 2396. http://www.ietf.org/rfc/rfc2396.txt

[5] P. Biron & A. Malhotra, *XML Schema Part 2: Datatypes Second Edition,* W3C Recommendation 28 October 2004. http://www.w3.org/TR/xmlschema-2/

[6] R. Fielding, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, IETF RFC 2616. http://www.rfc-editor.org/rfc/rfc2616.txt

TODO: add references to previous DAL services (e.g. SSA)