*I*nternational

*V*irtual

*O*bservatory

*A*lliance

IVOA Astronomical Data Query Language Version 1.02

IVOA Working Draft 24 September 2005

This version:

1.02: http://www.ivoa.net/Documents/WD/ADQL/ADQL-20050924.doc

Latest version:

http://www.ivoa.net/Documents/latest/ADQL.html

Previous versions:

none

Working Group:

http://www.ivoa.net/twiki/bin/view/IVOA/IvoaVOQL

Editors:

Maria A. Nieto-Santisteban, Masatoshi Ohishi, William O'Mullane, Yuji Shirasaki, and Alexander Szalay

Authors:

IVOA VOQL Working group

Abstract

This document describes the Astronomical Data Query Language (ADQL) and its two representations as String (ADQL/s) and XML (ADQL/x). ADQL has been developed based on SQL 92. This document describes the subset of the SQL 92 grammar

supported by ADQL. Special extensions to SQL 92 have been defined in order to support astronomy specific operations such as Region and XMATCH.

Status of this document

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

Acknowledgments

This working draft has been developed based on discussions at various IVOA meetings and continuing emails on the mailing list. The editors express their appreciation for many valuable contributions by Naoki Yasuda, Clive Page, Bob Mann, Martin Hill, and many others.

Contents

1 Introduction

The Astronomical Data Query Language (ADQL) is the language used by the International Virtual Observatory Alliance (IVOA) to represent astronomy queries posted to VO data services. IVOA has developed several standardized protocols to access astronomical data, e.g., SIAP, and SSAP for image and spectral data respectively, and the SkyNode Interface protocol to access catalogs. Different VO data services have different needs in terms of query complexity. For example, SIAP and SSAP can be satisfied using a single table. However, SkyNodes usually include more than one catalog table which makes necessary richer language expressivity. ADQL 1.0 has been designed in a layered hierarchy so data services implement the complexity level that meets their needs. In this way, clients know what query types the data services will accept.

ADQL 1.0 is based on the Structured Query Language (SQL), specifically on SQL 92. The VO has a number of tabular data sets and many of them are stored in Relational Databases (RDBs), making SQL a convenient access language. ADQL 1.0 focuses on a subset of the SELECT statement, adding a few extensions to define astronomy operations like REGIONS and XMATCH.

SkyNode services (or just nodes) accept queries in ADQL. The mechanism of passing a query to a node is described in the SkyNode Interface specification **Error! Reference source not found.**, developed by the IVOA VOQL WG as well. SkyNodes are defined and implemented as XML Web services. It should be noted that the SkyNode Interface is also related to IVOA Data Access Layer WG.

To access some current SkyNode implementations, visit, e.g. OpenSkyQuery.net. The OpenSkyQuery portal is an example of how astronomers can use ADQL to query a federation of astronomical databases which have been published as SkyNodes.

2 Astronomical Data Query Language (ADQL)

ADQL is based on a subset of SQL which has been extended to support queries which are specific to astronomy. The ADQL syntax specification is made of a core and several extensions. All SkyNodes **MUST** conform to the core specification. ADQL has two representations:

- **ADQL/s** : A string form based on the SELECT statement of the SQL 92 standard **Error! Reference source not found.** that conforms to the ADQL grammar (see appendix). Some non standard SQL extensions have been added to support astronomy queries.
- **ADQL/x** : An XML document conforming to the ADQL schema [3]. The XML document is the mechanism used to pass a query to the SkyNode Web service Interface.

ADQL/s and **ADQL/x** are translatable to each other without loss of information. [Translation Services & Translation Styles sheets]

2.1 Restrictions on SQL 92

The formal notation for syntax of computing languages is often expressed in the "Backus Naur Form" BNF[1]. Appendix to this document provides the BNF definition of **ADQL/s**.

In essence this is any valid SELECT SQL statement. However ADQL has restrictions described below.

2.1.1 Built-in Functions

In ADQL built-in functions which are defined on the server system may be called. These would include, e.g., a function to provide great circle distance, converter such as from sexagesimal to decimal, and unit converters. The SkyNode Interface specification also defines a method by which all functions available on the server may be discovered. If a user knows that certain functions exist in the target system (SkyNode etc.), the user may use such functions in ADQL. An example of a function would be (in **ADQL/s**):

```
Select HEALPIXID(a.ra, a.dec), a.ra, a.dec from photobjall a
```

A concise set of common built-in functions that represent the necessary astronomical functionality, together with their standard function names, will be defined in later versions of the ADQL specification.

2.1.2 INTO clause

INTO is supported for future interoperability with VOSpace. The VOSpace specification is under development within the Grid and Web Services WG of the IVOA. In SQL we may use '`SELECT INTO`' to create a new table or '`INSERT INTO`' to add data into an existing table. In ADQL this will probably be a VOSpace endpoint wherein the file/table will be created or appended to. How that is specified is not part of ADQL. ADQL simply supports syntax to allow to specification of a destination, e.g.:

```
Select g.* into VOS:/JHU/gal from galaxy g where g.redshift > 3.5
```

[1] http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html#Johnson75

2.1.3 Comments

Comments will only be supported using the /* */ syntax to delimit comments. Comments are only supported before or after the main query – they may not be interspersed with the actual query.

2.2 Extensions to SQL 92

This specification adds requirements on top of SQL92. ADQL SHALL support the extension described below.

These extensions to SQL92 are given with examples in **ADQL/s**, but of course **ADQL/x** can express any string from **ADQL/s**.

2.2.1 Aliases

All table names in ADQL MUST have an alias. Aliasing tables is a part of standard SQL, but we are enforcing this in **ADQL/s**.

This means queries in **ADQL/s** must take the form

```
Select * from table t
```

This makes substitution of table names much easier as it must be done in only one place to change the alias.

2.2.2 Archive Qualification

ADQL allows for an archive to be specified in front of the table name. The archive's SHORTNAME (registration name) is pre appended to the table name with the ':' separator. E.g. TWOMASS:PhotoPrimary refers to the PhotoPrimary table of the TWOMASS SkyNode.

2.2.3 Regions

ADQL adds a keyword `REGION` to be used in the `WHERE` clause to specify search constraints. The REGION specification is supported as defined by the IVOA Data Model WG [3]. See subsection 2.4 for its detailed specification. The default coordinate system and units have been specified to simplify ADQL and the SkyNode implementation.

2.2.4 Mathematical Funtions

JDBC [5] mathematical functions shall be allowed in ADQL as follows:

Trigonometric functions:

acos(x), **asin(x)**, **atan(x)**, **atan2(x, y)** where x and y are numeric, and

cos(x), **cot(x)**, **sin(x)**, **tan(x)** where x is expressed in radians

Math functions:

abs(x), **ceiling(x)**, **degrees(x)**, **exp(x)**, **floor(x)**, **log(x)**, **log10(x)**, **mod(x, y)**,

pi(), **power(x, y)**, **radians(x)**, **sqrt(x)**, **rand()**, **round(x, n)**, **truncate(x, n)**

where x and y are numeric and n is an integer.

2.2.5 XMATCH

ADQL includes a family of `XMATCH` keywords which mean cross-match between two or more astronomical catalogues. The semantic meaning of `XMATCH` is defined more precisely in the SkyNode Interface specification. This document only specifies the syntax. The `XMATCH` keyword appears in the `WHERE` clause and looks like a function. At the moment there is only one `XMATCH` function accepted. As new functions are accepted they will be included in this specification.The `XMATCH` has three parameters; first two parameters are table names to be cross-matched, the third parameter is the sigma value for the chi-square match.

Here is an example in **ADQL/s**:

```
SELECT o.objId, o.ra, o.r, o.type, t.objId
  FROM SDSS:PhotoPrimary o,
       TWOMASS:PhotoPrimary t
  WHERE XMATCH(o,t,3.5)
        AND Region('Circle J2000 181.3 -0.76 6.5')
        AND o.type=3
```

2.2.6 XPATH for Columns

To support XQuery as well as SQL, and since some of our data formats are described as XSD, it will be possible to express selections and selection criteria as a simple XPath. Square brackets ([,]) and standard operators such as parent are NOT supported. An example of a valid query of this form would be

```
Select /Resource/Contact/Name from Resource where /Resource/Type
like 'catalog'
```

2.2.7 Returning subset of records – TOP

ADQL supports the `TOP` syntax to return only the first N records from a query, e.g.,

```
Select top 10 g.* from galaxy g
```

The semantics of this may vary on different database management systems. In ADQL the assumption is that `TOP` returns the first N records satisfying the criteria specified in the query.

2.2.8 Units

ADQL allows units for all constant values specified in the query. These are **optional**. ADQL does not specify what the units mean, and it simply allows for them syntactically specified, e.g:

```
Select g.* from galaxy g where g.gmag > 100 Jansky
```

2.2.9 Table Names with special chars

ADQL supports the use of '**[ ]**' to enclose literal names which may otherwise cause parse errors. For example if a table name starts with a number the parser could not deal with this but the following is valid:

```
Select a.* from [2df] a
```

This is also true for table names with spaces in or tables whose names are reserved words. Many database systems also support this syntax.

2.3 Version information

ADQL/x documents SHALL contain a version identifier for the version of ADQL. This will start as 1.0. The version number is a dot separated string of numbers. The version number is included in the document solely so the receiving node may decide if it wishes to deal with the document or to return an exception. This is assumed to only come into use at some later stage when there may be a major version change causing some possible incompatibility between versions. We should strive for backward compatibility i.e. only adding new features not deprecating the old.

2.4 Regions

- **ADQL/s** SHALL support the `Region` keyword. This will be followed by a single quoted string specifying a region in a simple manner similar to the current SDSS coverage specification in [6]. This would look something like:

```
Region('CIRCLE J2000 19.5 –36.7 0.02')
```

This is a one way operation. If an **ADQL/s** string is converted to **ADQL/x** this `Region` string will be converted to XML. If the resulting **ADQL/x** is converted back to **ADQL/s** the `Region` should remain as inlined XML using the `RegionXML` keyword.
There may be a comment section added to the region.xsd. In this comment section the original string should be kept. The comment section will be used for display purposes in certain areas, and should contain a summary description (in English) of the region.
Other constructs mentioned in [6] are `RECT`, `POLY`, and `CHULL` are also supported.

As implied above it is possible to inline a region specification as in **ADQL/s** using the RegionXML keyword, e.g., (not a valid region specification)

```
RegionXML    (‘<circle><coordsys>ICRS</coordsys><ra>19.5</ra><dec>-
36.7</dec><radius>0.02</readius></circle>’)
```

It is also possible to refer to a region specification as a URL in **ADQL/s** using the RegionURL keyword, e.g.

```
RegionURL (‘http://aserver.edu/aregion.xml’)
```

3 ADQL example

An **ADQL/s** might be as follows:

```
SELECT a.objid, a.ra, a.dec
FROM SDSSDR2:Photoprimary a
WHERE Region('CIRCLE J2000 181.3 -0.76 6.5')
```

This would be represented in **ADQL/x** as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<Select xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.ivoa.net/xml/ADQL/v1.0">
  <SelectionList>
    <Item xsi:type="columnReferenceType" Table="a" Name="objid" />
    <Item xsi:type="columnReferenceType" Table="a" Name="ra" />
    <Item xsi:type="columnReferenceType" Table="a" Name="dec" />
  </SelectionList>
  <From>
```

```
    <Table xsi:type="archiveTableType" Archive="SDSSDR2"
Name="Photoprimary" Alias="a" />
  </From>
  <Where>
    <Condition xsi:type="regionSearchType">
      <Region xmlns:q1="http://www.ivoa.net/xml/STC/STCregion/v1.10"
xsi:type="q1:circleType" unit="deg">
        <q1:Center>181.3 -0.76</q1:Center>
        <q1:Radius>6.5</q1:Radius>
      </Region>
    </Condition>
  </Where>
</Select>
```

4 ADQL XSD

The XML schema for ADQL is found at http://www.ivoa.net/xml/ADQL/ADQL-v1.0.xsd.

5 Changes from previous versions

- None. This is the first release.

6 References

[1] IVOA SkyNode Interface
http://www.ivoa.net/Documents/latest/SNI.html

[2] ISO/IEC 9075:1992(E) Information technology – Database languages - SQL

[3] ADQL XML schema.
http://www.ivoa.net/xml/ADQL/ADQL-v1.0.xsd

[4] Space-Time Coordinates for the Virtual Observatory Version 1.10 http://www.ivoa.net/xml/STC/STCregion/v1.10

[5] Java Database Connectivity Specification 3.0; download from http://java.sun.com/products/jdbc/index.jsp

[6] SQLServer2000 HTM Interface specification; Alex Szalay, George Fekete, Jim Gray; July 2003 ; http://skyservice.pha.jhu.edu/develop/vo/adql/htmdll_2_0.doc

Appendix A ADQL Grammar

A- 1 Core Query Syntax

A-1-1 Construct

```
SELECT selection_list
   FROM table_name  [AS] alias
   [ WHERE condition ]
```

A-1-1-1 Select list

```
selection_list := { * | [table_alias.]* |
 {[table_alias.]column_name  [[AS] alias]}[,...] }
```

- `*` represents all the columns.
- `*` may be qualified by a table alias name.
- A column name may be qualified by a table alias name. Table name is not used for qualifying the column, as alias to the table is mandatory.
- Algebraic expression is not supported in this CORE spec.

A-1-1-2 FROM Clause

- Only one table may be specified in the from clause.
- An alias name must be given to the table.

A-1-1-3 WHERE Clause

- Boolean value expression that conforms to CORE syntax is specified.
- If boolean value expression that is not supported is specified, it should be evaluated as true rather than throwing an exception.
- Only one region search condition may be specified at most.

A-2-2 Specification number

`QL-SL-C01 [Core]` All the SkyNodes must support the Core construct.

A-2-3 XML representations

- Coming later…

A- 2 Full Query Syntax

A-1-1 Construct

```
SELECT[ ALL | DISTINCT ] [ INTO table_name ]
     [ TOP number ] [ OFFSET number ]
     selection_list
     FROM from_item [, ...]
     [ WHERE condition ]
     [ GROUP BY expression [, ...] ]
     [ HAVING condition [, ...] ]
```

```
    [ ORDER BY expression
       [ ASC | DESC | USING operator ] [, ...] ]
```

A-1-1-1 Select list

```
selection_list ::= * | {[table_alias].* |
  UCD [table_alias.]ucd | UTYPE [table_alias.]utype |
  [table_alias.]value_expression [[AS] select_alias]}[, ...]
```

- UCD is a keyword which is followed by a ucd name or matching pattern. This syntax will be used for selecting columns based on the ucds.
- UTYPE is a keyword which is followed by a utype name or matching pattern. This syntax will be used for selecting columns based on the utypes.
- value_expression is one of the followings:
    - a column_name
    - a function
    - a constant value
    - a numerical formula of them

A-1-1-2 FROM Clause

```
from_item ::= aliased_table_name [, aliased_table_name ]…
     [join_type from_item
       {ON comparison_pred | USING (column_name [,…])}]
     [ NATURAL join_type from_item [AS] alias]
  [ ( sub_query ) [AS] alias ]

  aliased_table_name ::=
    {[resource_id.] table_name | #upload}
     [AS] alias
```

- resource_id is a service identifier which is expressed as:

  `resource_id ::= authority_name:resource_path`

  e.g. ivo://archive.stsci.edu/hdfn/SKYNODE

  → archive.stsci.edu:[hdfn/SKYNODE]

- The exteranl votable is specified by the "#upload" keyword.
- The join_type is one of the followings:

– `CROSS JOIN` : which is identical to write two tables separated by a comma.
– `INNER JOIN` : "INNTER" may be omitted. *Explanation of this join.*
– `LEFT OUTER JOIN`: "OUTER" may be omitted . .*Explanation of this join*
– `RIGHT OUTER JOIN`: "OUTER" may be omitted . *Explanation of this join*
– `FULL OUTER JOIN`: "OUTER" may be omitted . .*Explanation of this join*

- The join condition that can be specified is one of the followings:

- `NATURAL :` join is performed by comparing rows of identical name in the two joined tables. Only one of the columns appears in the output list if it is not explicitly specified.
- `ON :` specifies the join condition, which is a comparison between a pair of rows from the two tables. Both of the columns appear in the output list, if it is not explicitly specified.
- `USING : JOIN USING (a,b,c)` is equivalent to `JOIN ON (t1.a=t2.a AND t1.b=t2.b AND t1.c=t2.c)` with the exception that only one of the identical column is included in the output list, if it is not explicitly specified.

• Subquery is a select statement.

A-1-1-3 WHERE Clause

• Boolean value expression supported on the node.

• If boolean value expression that is not supported is specified, it is recommended to evaluate it true rather than to throw an exception.

A-1-1-4 INTO, TOP, OFFSET, ALL, DISTINCT

• INTO: specifies the location of VOSpace where the query result is stored.

• TOP: returns only the first n rows from the offset position

• OFFSET: skip the first n rows

• The recurrence of the query result is not guaranteed by TOP and OFFSET selection. It is recommended to use them with the ORDER BY clause, which is the only way to guarantee the recurrence of the query result under the condition that contents of the table are not changed.

• ALL or DISTINCT: ALL selects all the rows. It is default. DISTINCT rejects duplicated rows from the query result.

A-1-1-4 GROUP BY, HAVING, ORDER BY

• GROUP BY : is …

• HAVING : is…

• ORDER BY : is…

A-1-2 Specification number

`QL-SL-E01 [Ext]` A SkyNode may support UCD selection syntax

`QL-SL-E02 [Ext]` A SkyNode may support UTYPE selection syntax

`QL-SL-E03 [Ext]` A SkyNode may support value expression in the selection list.

`QL-SL-E04 [Ext]` A SkyNode may support CROSS JOIN with one external table using the #upload keyword. A SkyNode which implement the xmatch service must support this syntax as well as xmatch function.

`QL-SL-E05 [Ext]` A SkyNode may support all the JOIN sysntax. In this case all the join types and the join condition types must be supported.

`QL-SL-E06 [Ext]` A SkyNode may support a subquery table

`QL-SL-E07 [Ext]` A SkyNode may support INTO syntax.

`QL-SL-E08 [Ext]` A SkyNode may support TOP syntax.

`QL-SL-E09 [Ext]` A SkyNode may support OFFSET syntax.

`QL-SL-E10 [Ext]` A SkyNode may support ALL and DISTINCT syntax.

`QL-SL-E11 [Ext]` A SkyNode may support GROUP BY syntax.

`QL-SL-E12 [Ext]` A SkyNode may support HAVING syntax.

`QL-SL-E13 [Ext]` A SkyNode may support ORDER BY syntax.

A-1-3 XML representations

A- 3 Keyword, Identifier and delimited identifier

A-3-1 Keyword

- ADQL Keywords:

SELECT, INTO, TOP, OFFSET, AS, FROM, WHERE, GROUP, BY, HAVING, ORDER, ASC, DESC, USING, BETWEEN, AND, OR, NOT LIKE, ... (not complete yet)

- keyword is case insensitive.

A-3-2 Identifier

- Identifier, such as a column name or a table name, must begin with a letter {a-z} or an underscore {_}. Subsequent characters in an identifier can be letters, underscores or digits {0-9}.
- Identifier that matches the keywords is not allowed.
- Identifier is case insensitive.

A-3-3 Delimited identifier

- Delimited identifier may be used to allow for the use of keywords or special characters in naming the column and table. Delimited identifier is enclosed by “[“ and ”]”.
- Delimited identifier is case sensitive.
- The way of writing ”[” and ”]” within a delimited identifier is to write two adjacent brackets. e.g. [O/Fe] --> [[[O/Fe]]].
- Use of the delimited identifier is not encouraged and should be avoided.

A-3-4 Specification number

`QL-KI-E01 [Ext]` All the SkyNodes must support keyword, identifier, delimited identifier specification.

A-4 Data types

A-4-1 Numeric type

A-4-1-1 Integer and Floating-Point types

bit	*
unsignedByte	1 byte
short	2 byte
int	4 byte
long	8 byte
float	4 byte
double	8 byte
floatComplex	8 byte
doubleComplex	16 byte

A-4-1-2 Literal expression

```
<digits>
<digits>.[<digits>][e[+-]<digits>]
[<digits>].<digits>[e[+-]<digits>]
<digits>e[+-]<digits>
```

where `<digits>` is one or more decimal digits (0 through 9).

e.g. 42, 3.5, 4., .001, 5e2, 1.925e-3

A-4-1-3 Functions, operators, and predicates for numeric value expression

- comparison operator: comparison must be made between numeric data types.

  <
  >
  <=
  >=
  =
  <> or !=
- BETWEEN predicate

```
a BETWEEN x AND y ( is equivalent to a>=x AND a <=y )
a NOT BETWEEN x AND y ( is equivalent to a<x OR a>y )
```

- IN predicate

```
A IN (n1, n2, …)
A NOT IN (n1, n2, …)
```

- NULL comparison predicate

```
a IS NULL
a IS NOT NULL
```

- mathematical operator

+	addition
-	subtraction
*	multiplication

`/`	division
`%`	modulo
`^`	exponentiatio

- mathematic function

```
abs(x)
exp(x)
ln(x)
log(x)
pi()
sqrt(x)
acos(x)
asin(x)
atan(x)
atan2(x, y)
cos(x)
cot(x)
sin(x)
tan(x)
```

- general function

`distance(coord1a, coord2a, coord1b, coord2b, 'frame')`

A-4-1-4 Specification number

`QL-NT-C01 [Core]` A SkyNode must support unsignedByte, short, int, long, float and double data types, and related operators, predicates and functions that conform Core specification.

`QL-NT-C02 [Core]` A SkyNode must support numeric comparison operator "<", ">", "<=", ">=", "=", "<>" and "!=".

`QL-NT-C03 [Core]` A SkyNode must support BETWEEN and NOT BETWEEN predicate for numeric data types.

`QL-NT-C03 [Core]` A SkyNode must support IN and NOT IN predicate for numeric data types.

`QL-NT-C04 [Core]` A SkyNode must support mathematical operators "+", "-", "*" and "/".

`QL-NT-E01 [Ext]` A Skynode may support null comparison predicate.

`QL-NT-E02 [Ext]` A Skynode may support mathematical operator "%" and "^".

`QL-NT-E03 [Ext]` A SkyNode may support all of the mathematical functions.

`QL-NT-E04 [Ext]` A SkyNode may support any functions.

A-4-1-5 XML representation

A-4-2 Character type

A-4-1-1 character types

`char`	1 byte

`char[n]`	n byte string
`char*`	string with variable unlimited length
`unicodeChar`	2 byte

A-4-1-2 Literal expression

`'{ non_single_quate_character | doubled_single_quates }…'`

A single quote can be specified in a string constant by writing two adjacent single quotes,

A-4-1-3 Functions, operators, and predicates for a character array data type

- comparison operator: comparison must be made on character data types

  <
  >
  <=
  >=
  =
  <> or !=

- string connection operator

  || connection of string

- BETWEEN predicate
- IN predicate
- LIKE predicate:

  "_" matches to one character, "%" matches to an arbitrary number of characters.

  ```
  S like 'APP_E'
  S like 'GRB%'
  ```

- NULL predicate
- string function

  ```
  substring()
  length()
  lower()
  upper()
  ```

A-4-1-4 Specification number

QL-CT-C01 [Core] A SkyNode must support char type and an array type of character.

QL-CT-C02 [Core] A SkyNode must support string comparison operators "=", "<>" and "!=".

QL-CT-E01 [Ext] A SkyNode may support string comparison operators "<", ">", "<=", ">=".

QL-CT-E02 [Ext] A SkyNode may support string connection operators "||"

`QL-CT-E03 [Ext]` A SkyNode may support string BETWEEN predicate.

`QL-CT-E04 [Ext]` A SkyNode may support string IN predicate.

`QL-CT-E05 [Ext]` A SkyNode may support LIKE predicate.

`QL-CT-E06 [Ext]` A SkyNode may support string NULL comparison predicate.

`QL-CT-E07 [Ext]` A SkyNode may support string function, substring(), length(), lower() and upprt()

A-4-1-4 XML representation

A-4-3 Date/Time type

A-4-1-1 data types

	Low Value	High Value	Resolution
Timestamp	TBD	TBD	<1s
date	TBD	TBD	1 day
time	00:00:00.00	23:59:59.999	<1s
datetime interval	TBD	TBD	<1s

A-4-1-2 Literal expression

```
[timestamp|date|time|datetime interval] 'expression'
```

standard expressions:

```
'2005-10-24'      ISO 8601
'20051024'  ISO 8601
'10:20:08.25'     ISO 8601
'10:20:08'  ISO 8601
'10:20'           ISO 8601
'102008'          ISO 8601
'2005-10-20 04:30:21+9'
'1 day 12 hours 59 min 10 sec'
```

extended expression

```
'2005-Oct-24'
'Oct-24-2005'
'24-Oct-2005'
'October 24, 2005'
'2005.100' year and day of year
'J2451187' Julian day
```

A-4-1-3 Functions, operators, and predicates

- comparison operator: comparison must be made on same data types.

  <
  >
  <=
  >=
  =
  <> or !=

- BETWEEN predicate
- IN predicate
- NULL predicate
- mathematical operator

+	addition	add datetime interval to date, time, or time stamp
−	subtraction	subtract datetime interval from date, time, or time stamp

A-4-1-4 Specification number

`QL-DT-C01 [Core]` A SkyNode must support date/time data types.

`QL-DT-C01 [Core]` A SkyNode must support date/time standard expression

`QL-DT-C01 [Core]` A SkyNode must support date/time comparison operator “<”, “>” “<=”, “>=”, “=”, “<>” and “!=”.

`QL-DT-C01 [Core]` A SkyNode must support date/time BETWEEN predicate.

`QL-DT-C01 [Core]` A SkyNode must support date/time IN predicate.

`QL-DT-E01 [Ext]` A SkyNode may support date/time extended expression.

`QL-DT-E02 [Ext]` A SkyNode may support date/time NULL predicate.

`QL-DT-E03 [Ext]` A SkyNode may support date/time mathematical operatior “+” and “-“.

A-4-1-5 XML representation

A-4-5 Boolean type

A-4-1-1 data type

```
  Boolean                   1 byte
```

A-4-1-1 Literal expression

Standard expressions:

```
TRUE
FALSE
```

Extended expressions:

```
't', 'true', 'y' , 'yes' , '1'
'f' 'false', 'n', 'no', '0'
```

A-4-1-2 Functions, operators, and predicates

- Logical operators

  AND
  OR
  NOT

- Boolean value functions

Region()
Xmatch_chi2()
Xmatch_distance()

- Boolean value predicates

  Comparison predicate

  BETWEEN predicate

  IN predicate

  LIKE predicate

  NULL predicate

A-4-1-3 Specification number

`QL-BT-C01 [Core]` A SkyNode must support Boolean data type and standard Boolean expression.

`QL-BT-C02 [Core]` A SkyNode must support logical operators “AND”, “OR” and “NOT”.

`QL-BT-E01 [Ext]` A SkyNode may support boolean extended expression.

A-4-1-3 XML representation

A-4-6 Array type of numeric type

A-4-1-1 data type

`Int[n]`

`Double[n]`

…

A-4-1-1 Literal expression

A-4-1-2 Functions, operators, and predicates

A-4-1-3 Specification number

A-4-1-4 XML representation

A-4-7 Space coordinate type

A-4-1-1 Data type

`point`

`circle`

`box`

A-4-1-1 Literal expression

```
Space 'Position [frame] coord1 coord2'

Space 'Circle [frame] coord1 coord2 radius [unit]'

Space 'Box [frame] coord1 coord2 size1 [unit] size2
[unit]'
```

- Frame is one of {ICRS, FK5, FK4, J2000, B1950, ECLIPTIC, GALACTIC}
- *frame* may be omitted if it is compared with a space value expression where frame is defined.
- coord1 and coord2 is spherical coordinate anlges (Ra,dec) or (long, lat) in degree or sexagecimal.
- Standard sexagecimal expreesion is:

```
hh:mm[:ss.ms]  {+|-}dd[:mm:ss.ms]
```

- Unit is a unit of region size and one of { deg | arcmin | arcsec }.

A-4-1-2 Functions, operators, and predicates

- Operators

```
Within          e.g. point within region
Covers          e.g. region1 covers region2
Overlaps        e.g. region1 overlaps region2
```

- Space coordinate value function & function of space coordinate values.

```
Point(coord1, coord2 [, frame])
Circle(point, radius [unit])
Box(center, size1 [unit], size2 [unit])
Region('space coordinate value expression')
Distance(p1, p2)
```

A-4-1-3 Specification number

`QL-SC-E01 [Ext]` A SkyNode may support Space coordinate data type.

`QL-SC-E01 [Ext]` A SkyNode may support Space coordinate operator within, covers, and overlaps.

A-4-1-4 XML representation