CQL 1.2: The Contextual Query Language Version 1.0

Committee Draft 01

30 June 2008

Specification URIs:
This Version:
http://docs.oasis-open.org/search-ws/june08releases/cql-1-2-V1.0-cd-01.doc (Authoritative)
http://docs.oasis-open.org/search-ws/june08releases/cql-1-2-V1.0-cd-01.pdf
http://docs.oasis-open.org/search-ws/june08releases/cql-1-2-V1.0-cd-01.html

Latest Version:
http://docs.oasis-open.org/search-ws/v1.0/cql-1-2-V1.0.doc
http://docs.oasis-open.org/search-ws/v1.0/cql-1-2-V1.0.pdf
http://docs.oasis-open.org/search-ws/v1.0/cql-1-2-v1.0.html

Technical Committee:
OASIS Search Web Services TC

Chair(s):
Ray Denenberg <rden@loc.gov>
Matthew Dovey <m.dovey@jisc.ac.uk>

Editor(s):
Ray Denenberg rden@loc.gov
Larry Dixson ldix@loc.gov
Matthew Dovey m.dovey@jisc.ac.uk
Janifer Gatenby Janifer.Gatenby@oclc.org
Ralph LeVan levan@oclc.org
Ashley Sanders a.sanders@MANCHESTER.AC.UK
Rob Sanderson azaroth@liverpool.ac.uk

Related work:
This specification is related to:

- Contextual Query Language (CQL)

Abstract:

CQL is a formal language for representing queries to information retrieval systems. The design objective is that queries be human readable and writable, and that the language be intuitive while maintaining the expressiveness of more complex languages.

Status:
This document was last revised or approved by the OASIS Search Web Services TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/search-ws

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/search-ws/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/search-ws/.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

1 Introduction 5
1.1 Terminology 5
1.2 Normative References 5
2 Query Syntax Description 6
2.1 Search Clause 6
2.1.1 Search Term 6
2.1.2 Index Name 7
2.1.3 Relation 7
2.2 Boolean Operators 8
2.2.1 Boolean Modifiers 8
2.2.2 Proximity Modifiers 8
2.3 Sorting 9
2.4 Prefix Assignment 9
2.5 Case Sensitivity 9
3 BNF 10
4 Context Sets 12
5 The CQL Context Set 13
5.1 Indexes 13
5.2 Relations 14
5.2.1 Implicit Relations 14
5.2.2 Defined Relations 15
5.2.3 Relation Modifiers 16
5.3 Booleans 19
5.3.1 Boolean Modifiers 19
Note about Proximity Units 20
6 The Sort Context Set 21
A. Diagnostics 23
B. Acknowledgements 27

1 Introduction

CQL, the *Contextual Query Language*, is a formal language for representing queries to information retrieval systems such as web indexes, bibliographic catalogs and museum collection information. The design objective is that queries be human readable and writable, and that the language be intuitive while maintaining the expressiveness of more complex languages.

Traditionally, query languages have fallen into two camps: Powerful, expressive languages, not easily readable nor writable by non-experts (e.g. SQL, XQuery);or simple and intuitive languages not powerful enough to express complex concepts (e.g. CCL and google). CQL combines simplicity and intuitiveness of expression for simple, every day queries, with the richness of more expressive languages when necessary to accommodate complex concepts.

1.1 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in **[RFC2119]**. When these words are not capitalized in this document, they are meant in their natural language sense.

1.2 Normative References

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

2 Query Syntax Description

A CQL query consists of either a single search clause [example a], or multiple search clauses connected by boolean operators [example b]. It may have a sort specification at the end, following the 'sortBy' keyword [example c]. In addition it may include a prefix [‘dc’ in example d] assigning a context for the search index, and even an assignment for a context prefix, that binds the short names to a context set identifier [‘> dc = "info:srw/context-sets/1/dc-v1.1"‘ in example e].

Examples:

a. title = fish

b. title = fish and creator = sanderson

c. title = fish sortBy date/ascending

d. dc.title = fish

e. > dc = "info:srw/context-sets/1/dc-v1.1" dc.title = fish

2.1 Search Clause

A search clause consists of either an index, relation and a search term [example a], or a search term by itself [example b]. Examples:

a. title = fish

b. fish

If the clause consists of just a term the index and relation are implied: the index is treated as 'cql.serverChoice', where ‘cql’ is the context and ‘serverChoice’ is the index (an index defined within the ‘cql’ context set) and the relation is treated as '=' [example c]. (Therefore example b and c are semantically equivalent.)

c. cql.serverChoice = fish

2.1.1 Search Term

Search terms MAY be enclosed in double quotes [example a], though need not be [example b]. Search terms MUST be enclosed in double quotes if they contain any of the following characters: < > = / ( ) and whitespace [example c]. The search term MUST be present in a search clause but it may be an empty string [example d]. The empty search term has no defined semantics.

Examples:

a. "fish"

b. fish

c. "squirrels fish"

d. “”

2.1.2 Index Name

An index name always includes a base name [example a] and may also include a prefix [example b], which determines the context set of which the index is a part. The base name and the prefix are separated by a dot character ('.'). If multiple '.' characters are present, then the first should be treated as the prefix/base name delimiter [example c]. If the prefix is not supplied, it is determined by the server [example a].

Examples:

a. title any Afish dog@ *[no prefix'. Prefix determined by the server.]*

b. dc.title any Afish dog@ *[prefix is 'dc']*

c. ac.bc.title any Afish dog@ *[prefix is 'ac']*

2.1.3 Relation

The relation in a search clause specifies the relationship between the index and search term. As for an index, It too always includes a base name [example a] and may also include a prefix providing a context for the relation [example b]. If a relation does not have a prefix, the context set is 'cql'. If no relation is supplied in a search clause, then = is assumed, which means that the relation is determined by the server. (As is noted above, if the relation is omitted then the index MUST also be omitted; the relation is assumed to be A=@ and the index is assumed to be cql.serverChoice; thus the server chooses both the index and the relation.)

Examples:

a. dc.title any "fish frog"
Find records where the title (as defined by the Adc@ context set) contains at least one of the words :fish@, Afrog@

b. dc.title cql.any "fish frog"
This query has the same meaning as the previous, since the default context set for the relation is Acql@.

c. dc.title cql.all "fish frog"
Find records where the title contains all of the words :fish@, Afrog@

2.1.3.1 Relation Modifiers

Relations may be modified by one or more relation modifiers. Relation modifiers always include a base name, and may include a prefix for a context set [example a] as above. If a prefix is not supplied, the context set is 'cql'. Relation modifiers are separated from each other and from the relation by forward slash characters('/'). Whitespace may be present on either side of a '/' character [example b], but the relation-plus-modifiers group may not end in a '/'. Relation modifiers may also have a comparison symbol and a value. The comparison symbol is '=' (equal), '<' (less than), '<=' (less than or equal), '>' (greater than), '>=' (greater than or equal), or '<>' (not equal). The value must conform to the same rules for quoting as search terms, above [example c].

Examples:

a. dc.title any/relevant fish
The relation modifier Arelevant@ means the server should use a relevancy algorithm for determining matches and the order of the result set. When the relevant modifier is used, the actual relation is often not significant.

b. dc.title any / relevant fish
This example is equivalent to example (a).

c. title any/rel.algorithm=cori fish
This example is distinguished from example (a) in which the modifier Arelevant@ is from the CQL context set. In this case the modifier is Aalgorithm=cori@, from the rel context set, in essence meaning use the relevance algorithm Acori@. A description of this context set is available at http://srw.cheshire3.org/contextSets/rel/

2.2 Boolean Operators

Search clauses may be linked by boolean operators. These are: **and**, **or**, **not** and **prox**. Note that **not** is semantically 'and-not' (it is not intended as a unary operator). Boolean operators all have the same precedence; they are evaluated left-to-right. Parentheses may be used to override left-to-right evaluation [example e].

Examples:

a. dc.title = “monkey house” **and** dc.creator = vonnegut

b. dc.title = fish **or** dc.creator = sanderson

c. dc.title = “monkey house” **not** dc.creator = vonnegut

d. cat **prox**/unit=word/distance>2/ordered hat
Find 'cat' where it appears more than two words before 'hat' (see 3.3.1.)

e. dc.title = fish or (dc.creator = sanderson and dc.identifier = "id:1234567")

2.2.1 Boolean Modifiers

Booleans may be modified by one or more boolean modifiers, separated as per relation modifiers with '/' characters. Again, boolean modifiers consist of a base name and may include a prefix determining the modifier's context set [example a]. If not supplied, then the context set is 'cql'. As per relation modifiers, they may also have a comparison symbol and a value [example b].

Examples:

a. dc.title = fish or/rel.combine=sum dc.creator any sanderson

b. dc.title = monkey prox/unit=word/distance>1 dc.title = house
Find records where both Amonkey@ and Ahouse@ are in the title, separated by at least one intervening word.

2.2.2 Proximity Modifiers

Basic proximity modifiers are defined in the CQL context set. Proximity units 'word', 'sentence', 'paragraph', and 'element' are defined there and may also be defined in other context sets. Within the CQL set they are explicitly undefined. When defined in another context set they may be assigned specific meaning.

Thus compare "prox/unit=word" with "prox/xyz.unit=word". In the first, 'unit' is a prox modifier from the CQL set, and as such its values are undefined, so 'word' is subject to interpretation by the server. In the second, 'unit' is a prox modifier defined by the xyz context set, which may assign the unit 'word' a specific meaning.

The context set xyz may define additional units, for example, 'street':

prox/xyz.unit="street"

This approach, 'prox/xyz.unit="street"', is chosen rather than 'Prox/unit=xyz.street' for the following reason. In the first case, 'unit' is a modifier defined in the xyz context set, and 'street' is a value defined for that modifier. In the second, 'unit' is a modifier from the cql context set, with a value defined in a different set. so its value would have to be one that is defined in the cql context set. This approach is chosen to avoid pairing a modifier from one set with a value from another, which can lead to unpredictable results.

2.3 Sorting

Queries may include explicit information on how to sort the result set generated by the search.

The sort specification is included at the end, and is separated by a 'sortBy' keyword. The specification consists of an ordered list of indexes, potentially with modifiers, to use as keys on which to sort the result set. If multiple keys are given, then the second and subsequent keys should be used to determine the order of items that would otherwise sort together. Each index used as a sort key has the same semantics as when it is used to search.

Modifiers may be attached to the index in the same way as to booleans and relations in the main part of the query. These modifiers may be part of any context set, including the CQL context set and the Sort context set. This is the only time when a modifier may be attached to an index. If a modifier may be used in this way it should be stated in the description of its semantics. As many types of search also require specification of term order (for example the <, > and within relations), these modifiers are often specified as relation modifiers.

Examples:

a. "cat" sortBy dc.title

b. "dinosaur" sortBy dc.date/sort.descending dc.title/sort.ascending

2.4 Prefix Assignment

Note: The use of Prefix Maps is uncommon.

A Prefix Map may be used to assign context set names to specific identifiers in order to be sure that the server maps them in a desired fashion. It may occur at any place in the query and applies to anything below the map in the query tree. A prefix assignment is specified by: '>' shortname '=' identifier [example a]. The shortname and '=' sign may be omitted, in which case it sets a default context set for indexes [example b].

Examples:

a. > dc = "info:units/direct-current" dc.voltage > 12
While Adc@ is almost always used as the prefix for the Dublin Core context set, this example illustrates that this is not always so, as in this case it is used for the (hypothetical) Adirect current@ context set.

b. > "info:units/direct-current" voltage > 12
This query has the same meaning as example a.

2.5 Case Sensitivity

All parts of CQL are case insensitive apart from user supplied search terms, values for modifiers and prefix map identifiers, which may or may not be case sensitive. If any case insensitive part of CQL is specified with mixed upper and lower case, it is for aesthetic purposes only.

3 BNF

Following is the Backus Naur Form (BNF) definition for CQL. ("::=" represents "is defined as".)

sortedQuery	::=	prefixAssignment sortedQuery | scopedClause ['sortby' sortSpec]
sortSpec	::=	sortSpec singleSpec | singleSpec
singleSpec	::=	index [modifierList]
cqlQuery	::=	prefixAssignment cqlQuery | scopedClause
prefixAssignment	::=	'>' prefix '=' uri | '>' uri
scopedClause	::=	scopedClause booleanGroup searchClause | searchClause
booleanGroup	::=	boolean [modifierList]
boolean	::=	'and' | 'or' | 'not' | 'prox'
searchClause	::=	'(' cqlQuery ')' | index relation searchTerm | searchTerm
relation	::=	comparitor [modifierList]
comparitor	::=	comparitorSymbol | namedComparitor
comparitorSymbol	::=	'=' | '>' | '<' | '>=' | '<=' | '<>' | '=='
namedComparitor	::=	identifier
modifierList	::=	modifierList modifier | modifier
modifier	::=	'/' modifierName [comparitorSymbol modifierValue]
prefix, uri, modifierName, modifierValue, searchTerm, index	::=	term
term	::=	identifier | 'and' | 'or' | 'not' | 'prox' | 'sortby'
identifier	::=	charString1 | charString2
charString1	:=	*Any sequence of characters that does not include any of the following:* *whitespace* *((open parenthesis)*

) (close parenthesis)

=

<

>

"" (double quote)

/

If the final sequence is a reserved word, that token is returned instead. Note that '.' (period) may be included, and a sequence of digits is also permitted. Reserved words are 'and', 'or', 'not', and 'prox' (case insensitive). When a reserved word is used in a search term, case is preserved.

charString2 := *Double quotes enclosing a sequence of any characters except double quote (unless preceded by backslash (\)). Backslash escapes the character following it. The resultant value includes all backslash characters except those releasing a double quote (this allows other systems to interpret the backslash character). The surrounding double quotes are not included.*

4 Context Sets

The "Contextual Query Language" is founded on the concept of searching by semantics or context (hence the name), rather than by syntax. The same search may be performed in a different way on very different underlying data structures in different servers, but both servers should understand the intent behind the query. In order for multiple communities to define their own semantics, CQL uses context sets to ensure cross-domain interoperability.

Context sets permit CQL users to create their own indexes, relations, relation modifiers and boolean modifiers without risk of choosing a name that someone else has chosen. All of these aspects of CQL must come from a context set, however there are rules for determining the prevailing default if one is not supplied. Context sets allow CQL to be used by communities in ways that the designers have not foreseen, while still maintaining the same rules for parsing which allow interoperability.

A contexts set may define:

- indexes
- relations
- relation modifiers
- boolean modifiers
- index modifiers, but only for use within a sort clause. See Sort Context Set.

When defining a new context set, it is necessary to provide a description of the semantics of each item within it. While context sets may contain indexes, relations, relation modifiers and boolean modifiers (and index modifiers for use in sort clauses), there is no requirement that all should be present; in fact most context sets define indexes only.

Each context set has a unique identifier, a URI. When indicating the context set in a query, a short form is used. The short name must be bound to the URI, and this binding may be sent as a mapping within the query itself, or be published by the recipient of the query in some protocol dependent fashion. The short name 'cql' is reserved for the CQL context set, but authors may wish to recommend a short name for use with their set.

An index, relation, or modifier qualified by a context is represented in the form *prefix*.*value*, (i.e. the prefix and value, separated by period) where *prefix* is a short name for a unique context set identifier.

5 The CQL Context Set

The CQL context set defines a set of indexes, relations and relation modifiers. The indexes supplied are 'utility' indexes which are generally useful across all applications of the language. These utility indexes are for instances when CQL is required to express a concept not directly related to the records, or for indexes applicable in practically every context.

The short name for this context set should always be ‘cql’, which is reserved for this context set. This is the only context set with a reserved name. Other context sets may recommend a short name to be used, but do not reserve that name.

The identifier for this context set is: info:srw/cql-context-set/1/cql-v1.2

5.1 Indexes

- **resultSetId**
  A search clause may be a result set id. This is a special case, where the index and relation are expressed as "cql.resultSetId =" and the term is the result set id returned by the server in the 'resultSetId' parameter of the searchRetrieve response. It may be used by itself in a query to refer to an existing result set from which records are desired. It may also be used in conjunction with other resultSetId clauses or other indexes, combined by boolean operators. The semantics of resultSetId with any relation other than ‘=’ is undefined.
  Example:
  cql.resultSetId = "5940824f-a2ae-41d0-99af-9a20bc4047b1" and dc.contributor=”Willie Mo”
  Match the result set with the given identifier.
- **allRecords**
  This is a special index which matches every record available. Every record is matched no matter what values are provided for the relation and term, but the recommended syntax is:
  cql.allRecords = 1.
  Example:
  cql.allRecords = 1 NOT dc.title = fish
  Search for all records that do not match 'fish' as a word in title.
- **allIndexes**
  The 'allIndexes' index will result in a search equivalent to searching all of the indexes (in all of the context sets) that the server has access to.
  Example:
  cql.allIndexes = fish
  If the server had three indexes - title, creator and date - then this would be the same as title = fish or creator = fish or date = fish
- **anywhere**
  Equivalent to ‘allIndexes’. Retained for historical purposes and expected to be deprecated in the future.
- **anyIndexes**
  The 'anyIndexes' index allows the server to determine how to search for the given term. The server may choose one or more indexes to search, which may or may not be generally available via CQL. It may choose a different index to search based on the term.
  This is the default when the index and relation is omitted from a search clause. The relation used when the index is omitted is '='.
  Examples:

cql.anyIndexes = fish
Search in any one or more indexes for the term fish

- **serverChoice**
  Equivalent to ‘anyIndexes’. Retained for historical purposes and expected to be deprecated in the future.
- **keywords**
  The keywords index is an index of terms determined by the server to be generally descriptive or meaningful to search on. It might include the full text of a document, descriptive metadata fields, or anything else generally useful to search as an initial entry point to the data. It might be a combination of other indexes. The server determines exactly what makes up this index, however the choice must be consistent, unlike anyIndexes above, when the choice can be different for different searches.
  Example:
  cql.keywords any/relevant "code computer calculator programming"
  Search the keywords index for the given term

5.2 Relations

5.2.1 Implicit Relations

These relations are defined in the grammar of CQL. The cql context set defines their meaning, when they are used within this context set (other context sets may assign different meanings).

- =
  This is the default relation, and the server can choose any appropriate relation or means of comparing the query term with the terms from the data being searched. If the term is numeric, the most commonly chosen relation is '==='. For a string term, either 'adj' or '==' as appropriate for the index and term.
  Examples:
  - animal.numberOfLegs = 4
    The recommended server choice for this example is '=='
  - dc.identifer = "gb 141 staff a-m"
    The recommended server choice for this example is '=='
  - *dc.title = "lord of the flies"*
    The recommended server choice for this example is 'adj'
  - *dc.date = "2004 2006"*
    The recommended server choice for this example is 'within'
- ==
  This relation is used for exact equality matching. The term in the data is exactly equal to the term in the search.
  Examples:
  - dc.identifier == "gb 141 staff a-m"
    Search for the string 'gb 141 staff a-m' in the identifier index.
  - animal.numberOfLegs == 4
    Search for animals with exactly 4 legs.
- <>
  This relation means 'not equal to' and matches anything which is not exactly equal to the search term.
  Examples:

- dc.date <> 2004-01-01
  Search for any date except the first of January, 2004
- dc.identifier <> ""
  Search for any identifier which is not the empty string.

- <, >, <=,>=

These relations retain their regular meanings as pertaining to ordered terms (less than, greater than, less than or equal to, greater than or equal to).
Examples:

- dc.date > 2006-09-01
  Search for dates after the 1st of September, 2006
- animal.numberOfLegs < 4
  Search for animals with less than 4 legs.

5.2.2 Defined Relations

These relations are defined as being widely useful as part of a default context set.

- adj

This relation is used for phrase searches. All of the words in the search term must appear, and must be adjacent to each other in the record in the order listed. The query could also be expressed using the PROX boolean operator.
Example:

- dc.description adj "blue shirt"
  Search for 'blue' immediately followed by 'shirt' in the description.

- *all, any*

These relations may be used when the term contains multiple items to indicate "all of these items" or "any of these items". These queries could be expressed using boolean AND and OR respectively. These relations have an implicit relation modifier of 'cql.word', which may be changed by use of alternative relation modifiers.
Examples:

- dc.title all "day life"
  Search for both day and life in the title.
- dc.description any "computer calculator"
  Search for either computer or calculator in the description.

- *within*

'within' may be used with a search term that has multiple dimensions. It matches if the record's value falls completely within the range, area or volume described by the search term, inclusive of the extents given.
Examples:

- dc.date within "2002 2003"
  Search for dates between 2002 and 2003 inclusive.
- animal.numberOfLegs within "2 5"
  Search for animals that have 2,3,4 or 5 legs.
- geo.point within "45.3,19.0 45.3,20.0 46.3,19.0 46.3,19.0 "
  Search for points within the indicated polygon. Note that the (hypothetical) geo context set in this example would specify how a search term represents a polygon.

- encloses

'encloses' is used when the index's data has multiple dimensions. (This contrast with 'within',

used with a search term that has multiple dimensions.) It matches if the database's term fully encloses the search term.
Examples:

- xyz.dateRange encloses 2002
  Search for ranges of dates that include the year 2002.
- *geo.area encloses "45.3, 19.0"*
  Search for any area that encloses the point 45.3, 19.0 The (hypothetical) geo context set in this example would specify how a search term represents a point.

5.2.3 Relation Modifiers

5.2.3.1 Functional Modifiers

- stem
  The server should apply a stemming algorithm to the words within the term. For example such that *cardiology,* and *cardiovascular* both match the stem of *cardio*.
- relevant
  The server should use a relevancy algorithm for determining matches and the order of the result set.
- phonetic
  The server should use a phonetic algorithm (for example, soundex) for determining words which sound like the term. For example such that *school* would be searched when the supplied term is skool.
- fuzzy
  The server should be liberal in what it counts as a match. The details are left to the server but might include permutations of character order, off-by-one for numerical terms and so forth.
- partial
  When used with within or encloses, there may be some section which extends outside of the term. This permits for the database term to be partially enclosed, or fall partially within the search term.
- ignoreCase, respectCase
  The server is instructed to either ignore or respect the case of the search term, rather than its default behavior (which is unspecified).
- ignoreAccents, respectAccents
  The server is instructed to either ignore or respect diacritics in terms, rather than its default behavior. (Default behavior is unspecified, but respectAccents is the recommended default.)
- locale=value
  The term should be treated as being from the specified locale. Locales will in general include specifications for whether sort order is case-sensitive or insensitive, how it treats accents, and so forth. The server determines the default locale. The value is usually of the form C, french, fr_CH, fr_CH.iso88591 or similar.

Examples:

- dc.title any/stem "computing disestablishmentarianism"
  Find the local stemmed form of 'computing' and 'disestablishmentarianism', and search for those stems in the stemmed forms of the terms in titles.

- person.phoneNumber =/fuzzy "0151 795-4252"
  Search for a phone number which is something similar to '0151 795-4252' but not necessarily exactly that number.

- dc.title within/locale=fr "l m"
  Find all titles between l and m, ensure that the locale is 'fr' for determining the order for what is between l and m.

5.2.3.2 Term-format Modifiers

These modifiers specify the format of the search term to ensure that the server performs the correct comparison. These modifiers may all be used in sort keys.

- word
  The term should be broken into words according to the server's definition of a 'word' .
- string
  The term is a single item, and should not be broken up.
- isoDate
  Each item within the term conforms to the ISO 8601 specification for expressing dates.
- number
  Each item within the term is a number.
- uri
  Each item within the term is a URI.
- oid
  Each item within the term is an ISO object identifier, dot-separated format.

Examples:

- dc.title =/string Jaws
  Search in title for the string 'Jaws', rather than Jaws as a word. (Equivalent to the use of == as the relation)

- zeerex.set ==/oid "1.2.840.10003.3.1"
  Search for the given OID

- numberOfLegs/number=4
  4 is treated as a number, so it should match the number 4 (for this index) no matter how it is represented in the data.

- title =/string one
  "one" is treated as a string, not a number.

5.2.3.3 Masking

- *masked*
  This is a default modifier: all of the following masking rules and special characters are assumed for search terms, unless the unmasked modifier is included. It may be overridden by the regexp modifier. (To explicitly request this functionality, add 'cql.masked' as a relation modifier.)
  - *****
    A single asterisk (*) is used to mask zero or more characters.

- **?**
  A single question mark (?) is used to mask a single character, thus N consecutive question-marks means mask N characters.
- **^**
  Carat/hat (^) is used as an anchor character for terms that are word lists, that is, where the relation is 'all' or 'any', or 'adj'. It may not be used to anchor a string, that is, when the relation is '==' (string matches are, by default, anchored). It may occur at the beginning or end of a word (with no intervening space) to mean right or left anchored."^" has no special meaning when it occurs within a word (not at the beginning or end) or string but must be escaped nevertheless.
- ****
  Backslash (\) is used to escape '*', '?', quote (") and '^' , as well as itself. Backslash not followed immediately by one of these characters is an error.

Examples:

- dc.title = c*t
  Matches words that start with c and end in t
- dc.title adj "*fish food*"
  Matches a word that ends in fish, followed by a word that starts with food. (For example it matches "swordfish foodfight".)
- dc.title = c?t
  Matches a three letter word that starts with c and ends in t.
- dc.title adj "^cat in the hat"
  Matches 'cat in the hat' where it is at the beginning of the field
- dc.title any "^cat ^dog rat^"
  Matches cat at the beginning, dog at the beginning or rat at the end. (For example matches "cat eats dog", "fish eats rat", but not "rat eats cat".)
- dc.title == "\"Of Couse\", she said"
  Escape internal double quotes within the term.

- ***unmasked***
  Do not apply masking rules, all characters are literal.
- ***substring***
  The 'substring' modifier may be used to specify a range of characters (first and last character) indicating the desired substring within the field to be searched. The modifier takes a value of the form "start:end" where:
  - Positive integers count forwards through the string, starting at 1. E.g. "1:10" means the first through tenth character.
  - Negative integers count backwards through the string, with -1 being the last character.
  - Both start and end are inclusive of that character.
  - If omitted, start defaults to 1.
  - If omitted, end defaults to -1.

  Examples:

  - dc.title =/substring="-5:" title
  - marc.008 =/substring="1:6" 920102

- dc.title =/substring=":" "The entire title"

    - dc.title =/substring="2:2" h

- regexp

  The term should be treated as a regular expression. Any features beyond those found in modern POSIX regular expressions are considered to be server dependent. This modifier overrides the default 'masked' modifier, above. It may be used in either a string or word context.

  Example:

  - dc.title adj/regexp "(lord|king|ruler) of th[ea] r.*s"
    Match lord or king or ruler, followed by of, followed by the or tha, followed by r plus zero or more characters plus s.

5.3 Booleans

A context set cannot define booleans, as these are defined by the CQL grammar. Boolean semantics may be modified by boolean modifiers defined by a context set, and the CQL context set defined boolean modifiers in 3.3.1.

CQL itself defines the following boolean operators.

- **AND**
  The combination of two sets of records with AND will result in the set of records that appear in both of the sets.
- **OR**
  The combination of two sets of records with OR will result in the set of records that appear in either or both of the sets. (It is inclusive OR, not exclusive OR.)

- **NOT**
  The combination of two sets of records with NOT will result in the set of records that appear in the left set, but not in the right hand set. It cannot be used as a unary operator.

- **PROX**
  prox is short for “proximity”. The prox boolean operator allows the relative locations of the terms to be specified as search criteria. prox semantics is defined by its modifiers as described below.

5.3.1 Boolean Modifiers

The CQL context set defines four boolean modifiers, which are used only with the prox boolean operator.

- **distance** <symbol> <value>
  The distance that the two terms should be separated by.
    - Symbol is one of: **<, >**, **<=**, **>=**, **=**, **<>**
      If the modifier is not supplied, it defaults to <=.
    - Value is a non-negative integer. If the modifier is not supplied, it defaults to 1 when unit=word, or 0 for all other units.
- unit=<value>
  The type of unit for the distance.

- Value is one of: '**paragraph** ,**sentence**, **word**, **element.** The default is 'word'. These values are explicitly undefined. They are subject to interpretation by the server. See “Note About Proximity Units” below.

- unordered
  The order of the two terms is unimportant. This is the default.

- **ordered**
  The order of the two terms must be as per the query.

Examples:

- cat prox/unit=word/distance>2/ordered hat
  Find 'cat' where it appears more than two words before 'hat'

- cat prox/unit=paragraph hat
  Find cat and hat appearing in the same paragraph (“same” meaning within zero paragraphs, as distance default to 0 when paragraph is the unit) in either order (unordered default)

Note about Proximity Units

As noted above proximity units 'paragraph', 'sentence', 'word' and 'element' are explicitly undefined when used by the CQL context set. Other context sets may assign them specific values.

Thus compare "prox/unit=word" with "prox/xyz.unit=word" (where ‘xyz’ is an arbitrary hypothetical context set). In the first, 'unit' is a prox modifier from the CQL set, and as such its values are undefined, so 'word' is subject to interpretation by the server. In the second, 'unit' is a prox modifier defined by the xyz context set, which may assign the unit 'word' a specific meaning.

Other context sets may define additional units, for example, 'street':

prox/xyz.unit="street"

Note that this approach, 'prox/xyz.unit="street"', is preferable to 'prox/unit=xyz.street'. In the first case, 'unit' is a modifier defined in the xyz context set, and 'street' is a value defined for that modifier. In the second, 'unit' is a modifier from the cql context set, with a value defined in a different set. so its value would have to be one that is defined in the cql context set. Pairing a modifier from one set with a value from another is not a good practice.

6 The Sort Context Set

The sort context set defines a set of index modifiers to be used within a sortby clause

The URI for this context set is: info:srw/cql-context-set/1/sort-v1.0. The recommended short name is: **sort**

Note that CQL does not permit index modifiers except within a sort clause. For example, in the CQL query: "author=wolfe sortby title" 'sortby title' is a sort clause; 'title' is an index. 'author', which is the primary index of query, may not have a modifier, but 'title', which is the index of the sort clause, may. Thus for example, in the CQL query: "author=wolfe sortby title/ascending" 'ascending' is an index modifier.

Index Modifiers

Modifier	Description
ignoreCase	Case-insensitive sorting: for example, unit and UNIT sort together.
respectCase	Case-sensitive sorting: for example, unit and UNIT sort separately.
ignoreAccents	Accent-insensitive sorting: for example sorensen and sørensen sort together.
respectAccents	Accent-sensitive sorting: for example sorensen and sørensen sort separately.
unicodeCollate=value	Specfies the Unicode collation level. The value should be a small integer as described in the Unicode Collation Algorithm report at www.unicode.org/reports/tr10 This modifier supersedes any of the above four modifiers. (None of the above should be used when 'unicodeCollate' is used.)
descending	Sort in descending order.
missingOmit	Records that have no value for the specified index are omitted from the sorted result set.
missingFail	Records that have no value for the specified index cause the search/sort operation to fail with the diagnostic info:srw/diagnostic/1/93.
missingLow	Records that have no value for the specified index are treated as if they had the lowest possible value, so that they sort first in ascending order and last in descending order.
missingHigh	Records that have no value for the specified index are treated as if they had the highest possible value.
missingValue=value	Records that have no value for the specified index are treated as if they had the specified value.
locale=value	Sort according to the specified locale, which in general includes specifications for whether sorting is case-sensitive or insensitive, how it treats accents, etc. The value is usually of the form C, french, fr_CH, fr_CH.iso88591 or similar

Examples

- dc.creator=plews sortby dc.title/sort.respectCase
  Sort by title, case sensitive
- dc.creator=plews sortby dc.title/sort.respectCase/sort.descending
  Sort case sensitive and in descending order
- dc.creator=plews sortby dc.date/sort.missingOmit
  Sort by date: records that have no date field are omitted from the result set.

- dc.creator=plews sortby dc.date/sort.missingValue=1970

  Sort by date: records that have no date field are sorted as though they had a date of 1970.

A. Diagnostics

Normative Annex

The diagnostics below are defined for use with the following namespace:

info:srw/diagnostic/1

The number in the first column identifies the specific diagnostic within that namespace (e.g., diagnostic 2 below is identified by the uri: i*nfo:srw/diagnostic/1/2*).

When CQL is used together with SRU, the Detail column indicates what should be returned in the details field. If this column is blank, the format is 'undefined' and the server may return whatever it feels appropriate, including nothing.

Number	**Description**	***Detail***	**Notes/Examples**
10	Query syntax error		The query was invalid, but no information is given for exactly what was wrong with it.
12	Too many characters in query	Maximum supported	The length (number of characters) of the query exceeds the maximum length supported by the server.
13	Invalid or unsupported use of parentheses	Character offset to error	The query couldn't be processed due to the use of parentheses. Typically either they are mismatched, or in the wrong place. Eg. (((fish) or (sword and (b or) c)
14	Invalid or unsupported use of quotes	Character offset to error	The query couldn't be processed due to the use of quotes. Typically that they are mismatched Eg. "fish'
15	Unsupported context set	URI or short name of context set	A context set given in the query isn't known to the server. Eg. foo.title any fish.
16	Unsupported index	Name of index	The index isn't known, possibly within a context set. Eg. dc.author any sanderson (dc has a creator index, not author)
18	Unsupported combination of indexes	Space delimited index names	The particular use of indexes in a boolean query can't be processed. Eg. The server may not be able to do title queries merged with description queries.
19	Unsupported relation	Relation	A relation in the query is unknown or unsupported. Eg. The server can't handle 'within' searches for dates, but can handle equality searches.
20	Unsupported relation modifier	Value	A relation modifier in the query is unknown or unsupported by the server. Eg. 'dc.title any/fuzzy starfish' when fuzzy isn't supported.

21	Unsupported combination of relation modifers	Slash separated relation modifiers	Two (or more) relation modifiers can't be used together. Eg. dc.title any/cql.word/cql.string "star fish"
22	Unsupported combination of relation and index	Space separated index and relation	While the index and relation are supported, they can't be used together. Eg. dc.author within "1 5"
23	Too many characters in term	Length of longest term	The term is too long. Eg. The server may simply refuse to process a term longer than a given length.
24	Unsupported combination of relation and term	Space separated relation and term	The relation cannot be used to process the term. Eg dc.title within "sanderson"
26	Non special character escaped in term	Character incorrectly escaped	Eg "\a\r\n\s"
28	Masking character not supported		A masking character given in the query is not supported. Eg. The server may not support * or ? or both
29	Masked words too short	Minimum word length	The masked words are too short, so the server won't process them because they would likely match too many terms. Eg. dc.title any *
30	Too many masking characters in term	Maximum number supported	The query has too many masking characters, so the server won't process them. Eg. dc.title any "???a*f??b* *a?"
31	Anchoring character not supported		The server doesn't support the anchoring character (^) Eg dc.title = "^jaws"
32	Anchoring character in unsupported position	Character offset	The anchoring character appears in an invalid part of the term, typically the middle of a word. Eg dc.title any "fi^sh"
33	Combination of proximity/adjacency and masking characters not supported		The server cannot handle both adjacency (= relation for words) or proximity (the boolean) in combination with masking characters. Eg. dc.title = "this is a titl* fo? a b*k"
34	Combination of proximity/adjacency and anchoring characters not supported		The server cannot handle anchoring characters.
35	Term contains only stopwords	Value	If the server does not index words such as 'the' or 'a', and the term consists only of these words, then while there may be records that match, the server cannot find any. Eg. dc.title any "the"

36	Term in invalid format for index or relation		This might happen when the index is of dates or numbers, but the term given is a word. Eg dc.date > "fish"
37	Unsupported boolean operator	Value	For cases when the server does not support all of the boolean operators defined by CQL. The most commonly unsupported is Proximity, but could be used for NOT, OR or AND.
38	Too many boolean operators in query	Maximum number supported	There were too many search clauses given for the server to process.
39	Proximity not supported		Proximity is not supported at all.
40	Unsupported proximity relation	Value	The relation given for the proximity is unsupported. Eg the server can only process = and > was given.
41	Unsupported proximity distance	Value	The distance was too big or too small for the server to handle, or didn't make sense. Eg 0 characters or less than 100000 words
42	Unsupported proximity unit	Value	The unit of proximity is unsupported, possibly because it is not defined.
43	Unsupported proximity ordering	Value	The server cannot process the requested order or lack thereof for the proximity boolean
44	Unsupported combination of proximity modifiers	Slash separated values	While all of the modifiers are supported individually, this particular combination is not.
46	Unsupported boolean modifier	Value	A boolean modifier on the request isn't supported.
47	Cannot process query; reason unknown		The server can't tell (or isn't telling) you why it can't execute the query.
48	Query feature unsupported	Feature	the server is able (contrast with 47) to tell you that something you asked for is not supported.
49	Masking character in unsupported position	the rejected term	eg, a server that can handle xyz* but not *xyz or x*yz
50	Result sets not supported		The server cannot create a persistent result set.
51	Result set does not exist	Result set identifier	The client asked for a result set in the query which does not exist, either because it never did or because it had expired.
52	Result set temporarily	Result set	The result set exists, it cannot be accessed, but will be able to be accessed again in the

	unavailable	identifier	future.
53	Result sets only supported for retrieval		Other operations on results apart from retrieval, such as sorting them or combining them, are not supported.
55	Combination of result set with search terms not supported.		Existing result sets cannot be combined with new terms to create new result sets. eg cql.resultsetid = foo not dc.title any fish
58	Result set created with unpredictable partial results available		The result set is not complete; possibly, the processing was interrupted. Some of the results may not even be valid.
59	Result set created with valid partial results available		All of the records in the result set are valid, but not all records that should be there necessarily are.
60	Result set not created: too many matching records	Maximum number	There were too many records to create a persistent result set.

B. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Kerry Blinco, Australian Department of Education, Employment and Workplace Relations

Ray Denenberg, Library of Congress
Larry Dixson, Library of Congress
Matthew Dovey, JISC
Janifer Gatenby, OCLC/PICS
Ralph LeVan, OCLC
Ashley Sanders, University of Manchester
Rob Sanderson, University of Liverpool