# CQL Introduction

Chess Query Language (CQL) was designed to allow researchers, authors, and players to search for games, problems, and studies that match specific themes.

You specify the theme you are looking for and the database to look for them, in CQL.

Then you run this CQL file using cql.exe .

This creates a new PGN file that has all the games matching your theme.

CQL specifies a small but powerful set of primitives to define chess themes. CQL can find much more complex themes than any other chess program.

Users have searched for themes like stalemates with multiple pins; games in which the same position recurs but with the winning side missing just one piece;

Novotny and Grimshaw themes, games with some number of captures on a single square, games with a certain number of black and white passed pawns, and many more.

Here you can download the [CQL.zip](http://www.arves.org/arves/images/Zip/cql.zip) file for free.

[And also on Wikipedia.](https://en.wikipedia.org/wiki/Chess_Query_Language)

References for good introductions to CQL are:

1. The [article](http://timkr.home.xs4all.nl/chess2/cql.htm) by Tim Krabbé in his Open Chess Diary.
2. [Gady Costeff's article](http://www.arves.org/arves/images/PDF/EG_PDF/eg151.pdf) in EG #151.
3. Scidb is a [Chess Information Data Base](http://scidb.sourceforge.net/help/en/CQL-Index.html) and has CQL implemented with extensions.
4. [Chess Assistent uses CQL and here is their tutorial](http://chessok.com/?p=21989).

## **Credits**

CQL was developed by Gady Costeff and Lewis Stiller.

It is Copyright (c) 2003-2004 and is free.

It also uses Scid code by Shane Hudson.

The Windows build uses the Cygwin Libraries.

Inhoudsopgave

[CQL Introduction 1](#_Toc453918623)

[Credits 2](#_Toc453918624)

[**Downloading CQL** 10](#_Toc453918625)

[**Installing CQL** 10](#_Toc453918626)

[Usage of CQL 10](#_Toc453918627)

[Example usage 11](#_Toc453918628)

[CQL Syntax 12](#_Toc453918629)

[Example of keyword parameter 13](#_Toc453918630)

[Overall functioning of CQL 14](#_Toc453918631)

[Example 15](#_Toc453918632)

[Comments in a CQL file 15](#_Toc453918633)

[Example of comments 15](#_Toc453918634)

[Piece designators 16](#_Toc453918635)

[Examples of piece designators 16](#_Toc453918636)

[Allowed piecetype designators 17](#_Toc453918637)

[Standard piecetype designators 17](#_Toc453918638)

[Additional piecetype designators 17](#_Toc453918639)

[Tagging piecetype designators 18](#_Toc453918640)

[Square designators 18](#_Toc453918641)

[Example 18](#_Toc453918642)

[Example 19](#_Toc453918643)

[Compound square designators 19](#_Toc453918644)

[Examples of compound square designators 19](#_Toc453918645)

[Examples of piece designators 19](#_Toc453918646)

[Transforms of piece designators 20](#_Toc453918647)

[Range specifiers 21](#_Toc453918648)

[Range specifier example 21](#_Toc453918649)

[The match list 23](#_Toc453918650)

[:pgn 23](#_Toc453918651)

[:output 23](#_Toc453918652)

[:year 23](#_Toc453918653)

[:result 23](#_Toc453918654)

[:gamenumber 23](#_Toc453918655)

[:composer 23](#_Toc453918656)

[:forany 24](#_Toc453918657)

[Example of match list 24](#_Toc453918658)

[The Position List 25](#_Toc453918659)

[Piece designators inside a position 25](#_Toc453918660)

[Simple keywords in a position list 26](#_Toc453918661)

[:mate 26](#_Toc453918662)

[:stalemate 26](#_Toc453918663)

[:wtm 26](#_Toc453918664)

[:btm 26](#_Toc453918665)

[:variations 27](#_Toc453918666)

[:variationsonly 27](#_Toc453918667)

[:initial 27](#_Toc453918668)

[:terminal 27](#_Toc453918669)

[:markall 27](#_Toc453918670)

[:check 27](#_Toc453918671)

[:nocheck 27](#_Toc453918672)

[:noannotate 27](#_Toc453918673)

[Examples using simple keywords 27](#_Toc453918674)

[Transformation keywords in a position list 28](#_Toc453918675)

[:flipdihedral 29](#_Toc453918676)

[:flip 29](#_Toc453918677)

[:fliphorizontal 30](#_Toc453918678)

[:flipvertical 30](#_Toc453918679)

[:flipcolor 30](#_Toc453918680)

[:flipdiagonal 30](#_Toc453918681)

[:shifthorizontal 30](#_Toc453918682)

[:shiftvertical 30](#_Toc453918683)

[:shiftmaindiagonal 30](#_Toc453918684)

[:shiftoffdiagonal 30](#_Toc453918685)

[:shiftdiagonal 30](#_Toc453918686)

[:shift 30](#_Toc453918687)

[Examples of transformation keyword uses 30](#_Toc453918688)

[Multiple transformation keywords 32](#_Toc453918689)

[Piece configuration keywords 33](#_Toc453918690)

[:piececount 33](#_Toc453918691)

[:power 34](#_Toc453918692)

[:powerdifference 34](#_Toc453918693)

[:attackcount 35](#_Toc453918694)

[:ray 36](#_Toc453918695)

[:ray examples 37](#_Toc453918696)

[:raydiagonal, :rayhorizontal, :rayvertical, :rayorthogonal 37](#_Toc453918697)

[:rayattack 38](#_Toc453918698)

[Move keywords 39](#_Toc453918699)

[:movefrom 39](#_Toc453918700)

[:moveto 39](#_Toc453918701)

[:promote 40](#_Toc453918702)

[:enpassant 41](#_Toc453918703)

[:noenpassant 41](#_Toc453918704)

[:movenumber 41](#_Toc453918705)

[Logical keywords 41](#_Toc453918706)

[:and 41](#_Toc453918707)

[:or 41](#_Toc453918708)

[:not 41](#_Toc453918709)

[Sequence keywords 42](#_Toc453918710)

[:sequence 43](#_Toc453918711)

[:gappedsequence 44](#_Toc453918712)

[Relations between positions 46](#_Toc453918713)

[:relation 46](#_Toc453918714)

[Game-related position list keywords 46](#_Toc453918715)

[:elo 46](#_Toc453918716)

[:whiteelo 46](#_Toc453918717)

[:blackelo 46](#_Toc453918718)

[:result 47](#_Toc453918719)

[Matching count keywords 47](#_Toc453918720)

[:matchcount 47](#_Toc453918721)

[:pretransformmatchcount 47](#_Toc453918722)

[Accumulator keywords 48](#_Toc453918723)

[:accumulate 48](#_Toc453918724)

[:sumrange 49](#_Toc453918725)

[Using accumulators to count passed pawns 49](#_Toc453918726)

[Tagging keywords 51](#_Toc453918727)

[The :relation keyword 52](#_Toc453918728)

[:pattern 52](#_Toc453918729)

[:originaldifferentcount 52](#_Toc453918730)

[:originalsamecount 52](#_Toc453918731)

[:shift 53](#_Toc453918732)

[:flip 53](#_Toc453918733)

[:samesidetomove 53](#_Toc453918734)

[:changesidetomove 53](#_Toc453918735)

[:ignoresidetomove 53](#_Toc453918736)

[:missingpiececount 53](#_Toc453918737)

[:newpiececount 53](#_Toc453918738)

[:variations 54](#_Toc453918739)

[:variationsonly 54](#_Toc453918740)

[Relation list examples 54](#_Toc453918741)

[Tagging 56](#_Toc453918742)

[:forany in the match list 56](#_Toc453918743)

[:tagmatch in the position list 56](#_Toc453918744)

[Piece designators and tags 56](#_Toc453918745)

[Longer example 57](#_Toc453918746)

[Other examples of tagging 58](#_Toc453918747)

[CQL FAQ 59](#_Toc453918748)

[Why is the documentation so difficult to understand? 59](#_Toc453918749)

[What is the latest version? 59](#_Toc453918750)

[Why am I getting anomalous results when I use :shift and a piece is on the edge of the board? 59](#_Toc453918751)

[How does :moveto work? 60](#_Toc453918752)

[How can I check for an unguarded guard? 60](#_Toc453918753)

[How about an unguarded guard in which black then captures the interposing white piece and another piece interposes afterward to stop the check? 62](#_Toc453918754)

[In the Dobrescu example, dobrescu.cql, should :originalsamecount 0 be :originalsamecount 0 1 ? 63](#_Toc453918755)

[How can I look for a passed pawn on the b file? 64](#_Toc453918756)

[Does :flipcolor affect :result? 65](#_Toc453918757)

[How does one represent light-squared or dark-squared bishops? 65](#_Toc453918758)

[Why does CQL crash on a simple .cql file? 65](#_Toc453918759)

[Acknowledgments: 66](#_Toc453918760)

**Downloading CQL**

CQL may be downloaded in zip format from [CQL.zip](http://www.arves.org/arves/images/Zip/cql.zip)[.](https://web.archive.org/web/20110715160043/http:/www.rbnn.com/cql/cql.zip)

Harold van der Heijden's database of chess studies, *Endstudy Study Database 2015* can be purchased from his website [Harold van der Heijden Database V 2015](http://www.hhdbv.nl/).

A tool by Emil Vlasák that makes CQL easier to use, is available from [Visual CQL](http://www.vlasak.biz/vcql.htm).

The given examples can be downloaded as a [CQL\_Examples.zip](http://www.arves.org/arves/images/Zip/CQL_Examples.zip) from here.

The introduction article by Gady Costeff [chess\_query\_language.pdf](http://www.arves.org/arves/images/PDF/chess_query_language.pdf) can be downloaded from here.

Other sites with explanations and implementations are: [Scidb](http://scidb.sourceforge.net/help/en/Overview.html) and [PGN](http://portablegamenotation.com/cqldescription.html).

**Installing CQL**

1. Unzip the file cql.zip. This will create a directory named \cql in the directory from which cql.zip was unzipped.
2. Convert Heijden's ChessBase endstudy study database (or whichever database you want to search through) to PGN format. To do this:
   * Create a new database named heijden.pgn in chessbase with a format of PGN.
   * Drag the icon for the Study Database onto the icon of the new database.
   * Move heijden.pgn into the \cql directory.

This completes the installation. You can now [run](http://www.arves.org/arves/index.php/en/25-tools/343-how-to-run-cql) CQL.

# Usage of CQL

The CQL program must be run from the command line in a DOS prompt. You should be in the \cql directory to run the program.

The program takes a single command-line argument, the name of a cql file.

## Example usage

C:\> cd cql

C:\cql> cql idealmirrormate.cql

Sample CQL files are available [here](http://www.arves.org/arves/index.php/en/25-tools/349-cql-sample-files).

The ".cql" extension is added by default if missing in the command line invocation.

The meaning of a CQL file is determined by the [CQL Syntactic conventions](http://www.arves.org/arves/index.php/en/25-tools/344-cql-syntax)

# CQL Syntax

A CQL file has the following generic structure:

(match

:pgn input\_filename

:output output\_filename

(position ... )

(position ... )

(position ... )

...

)

In this example, the "..." represent various keywords and commands to be defined below.

For example, here is a simple CQL file to find quadrupled pawns:

(match

:pgn heijden.pgn

:output out.pgn

(position

:piececount P[b2-7] 4

:shifthorizontal

:flipcolor

)

)

This will read all the studies from the file heijden.pgn and will write a PGN file named out.pgn that contains the studies in heijden.pgn that have a position with quadrupled pawns.

In order to understand the format of a @CQL file, some basic syntactic terminology will be defined.

A list is a finite sequence of items, separated by whitespace, and enclosed in parentheses.

Each item is either a string of characters without whitespace or another list.

For example

(position :shifthorizontal :flipcolor)

is a list whose three elements are the strings "position", ":shifthorizontal", and ":flipcolor" .

Similarly, the list beginning "(match" above has six elements, the last of which is itself a list containing six elements whose first element is "position" and whose last element is ":flipcolor".

A keyword is a string beginning with ":".

For example, the match list above has keywords :pgn and :output .

A keyword parameter is a list or string immediately following a keyword that accepts a keyword parameter.

Some keywords accept multiple parameters, one after the other.

### Example of keyword parameter

In the list

(match

:pgn heijden.pgn

:output out.pgn

(position

:piececount P[b2-7] 4

:shifthorizontal

:flipcolor

)

)

:pgn has keyword parameter heijden.pgn, :output has keyword parameter out.pgn, and :piececount has keyword parameters P[b2-7] and 4.

The other keywords :shifthorizontal and :flipcolor do not take keyword parameters.

A list whose first element is "match" is called a match list and defines a match filter.

A list whose first element is "position" is called a position list and defines a position filter.

## Overall functioning of CQL

CQL is invoked on a CQL file which itself must contain exactly one match list.

This match list has a :pgn keyword which defines the PGN file that has the studies (or studies) to be searched.

When CQL is started up, it reads the CQL file given to it on the command line and parses the match list.

It then opens the PGN file specifed by the :pgn keyword.

It successively reads each study in the PGN file. After it reads a study, it applies the match filter to that study.

If the match filter matches the study, the study is printed to the pgn file defined by the :out keyword.

If the match filter does not match the study, then CQL ignores the study and goes onto the next study.

To determine whether a match filter matches a study, CQL acts as follows.

It plays through each move in the study, optionally descending as well into the variations.

Each time a position is reached, each of the position filters defined in the match filter is applied to that position.

### Example

In a 40 move game in which white makes the first move and black the last move, if there are no variations then CQL will examine exactly 80 positions in the game.

If each of the position filters in the match filter match some position in the study, then the match filter is said to match the study or study.

The simplest position filter is defined by the simplest position list:

(position)

This position filter matches every position.

Hence, the following CQL script simply copies each study from its :pgn input to its :output file:

(match

:pgn heijden.pgn

:output out.pgn

(position)

)

## Comments in a CQL file

Any characters following a semicolon are ignored in a CQL file.

### Example of comments

The following CQL file is exactly the same as the one above:

; A CQL file that copies input to output

(match

:pgn heijden.pgn ; the PGN file to be copies

:output out.pgn ; where to write it

(position) ; a position filter matching anything

)

## Piece designators

A piece designator is a string that denotes a set of pieces, each of which can be on a set of squares.

A piece designator comprises a piecetype designator followed by a square designator.

The piecetype designator denotes a set of pieces and the square designator denotes a set of squares.

The set of pieces may include the "empty square piece" which denotes an empty square.

A piece designator matches a position if there is a square in the set of squares denoted by that piece designator which contains a piece in the set of piecetypes denoted by the piece designator.

### Examples of piece designators

The piece designator Ra2 denotes a white rook on a2.

The piecetype designator is R and the square designator is a2.

It will match any position with a white Rook on a2.

The piece designator bc1 denotes a black bishop on c1.

The piecetype designator is b and the square designator is c1.

It will match any position with a black Bishop on c1.

The piece designator [Rb]d1 denotes either a white rook or a black bishop on d1.

The piecetype designator is [Rb] and the square designator is d1 .

It will match any position that has either a white rook or a black bishop on d1.

The piece designator [Qr][d1,h3] denotes either a white queen or a black rook on either d1 or h3.

The piecetype designator is [QR] and the square designator is [d1,h3] .

It will match any position that has either a white queen or a black rook on either d1 or h3. (Of course, the position can also have a white queen or black rook on both d1 and h3).

## Allowed piecetype designators

These are the allowed piecetype designators:

### Standard piecetype designators

K white king

k black king

N white knight

n black knight

B white bishop

b black bishop

P white pawn

p black pawn

R white rook

r black rook

Q white queen

q black queen

# 

### Additional piecetype designators

. empty square

A any white piece

a any black piece

M white major piece

m black major piece

I white minor piece

i black minor piece

U any piece at all

? any piece or an empty square

# 

### Tagging piecetype designators

Anywhere a piecetype designator may occur, a tag name preceded by a '$' may occur.

This piecetype represents the exact piece represented by the named tag.

For more information, see the section on [tagging](http://www.rbnn.com/cql/tagging.html).

## Square designators

A square designator defines a set of squares.

An empty square designator denotes all 64 squares of the board.

### Example

The piece designator R has an empty square designator.

Hence, this piece designator denotes a white rook anywhere on the board.

A simple square designator comprises two characters, the first of which is a file, and the second a rank, in the usual chess notation.

If the file character is '?' then files a through h are designated.

If the rank character is '?' then ranks 1 through 8 are designated.

### Example

The piece designator bb3 has a simple square designator denoting the square b3, and denotes a black bishop on b3.

### Compound square designators

A compound square designator can denote more than one square.

One type of compound square designator comprises a file designator followed by a rank designator.

A file designator comprises either the name of a file or the names of two files separated by hyphens.

A rank designator comprises either the name of a rank or the name of two ranks separated by hyphens.

A three-character file designator denotes all the files inclusive between the files designated by its first and last characters.

A three-character rank designator denotes all the ranks inclusive between the ranks designated by its first and last characters.

A compound square designator can also comprise a sequence of square designators separated by commas.

The set of squares so represented is the union of the sets of squares represented by each of the constituent square designators.

Any compound square designator not contained in brackets must be contained in brackets.

### Examples of compound square designators

[a-c1] denotes the three squares a1, b1, and c1. [b3-5] denotes the squares b3, b4, and b5 [f-h6-8] denotes the squares f6, f7, f8, g6, g7, g8, h6, h7, and h8.

[a1,c4] denotes the squares a1 and c4

[a-h1-2,a8,h8] denotes the first two ranks plus a8 and h8.

[] denotes no squares at all. This is very different from the empty square designator, which designates all squares.

### Examples of piece designators

Ia1 denotes a white minor piece on a1.

It will match any position with a white bishop or white knight on a1. R[a-h1-2] will match any position with a white rook on the first two ranks.

[Ar][a-h1-2,a8,h8] will match any position that has a white piece or a black rook on the first two ranks or the corners.

[RBNQP][a1,h1,a8,h8] will match any position with a white non-king piece at the corners.

B[] denotes a white bishop on an empty set of squares. This can never match any position of course.

## Transforms of piece designators

Certain transforms are defined on piece designators.

A diagonal flip transform, applied to a piece designator, reflects each square in its square set about the main diagonal a1-h8.

For example, the diagonal flip of Ra8 is Rh1 .

The diagonal flip of [Aa][a-h1-2,f6,h7] is [Aa][a-b1-8,f6,g8].

A off-diagonal flip transform applied to a piece designator reflects each square in its square set about the a8 to h1 diagonal.

For example, the off-diagonal flip of Ra8 is Ra8.

The off-diagonal flip of [Pp][a1-8], denoting a pawn on the a file, is [Pp][a-h8], denoting a pawn on the eight rank.

A vertical flip transform applied to a piece designator reflects each square in its square set about the vertical bisector of the board.

For example, the vertical flip of Uf6 , denoting a piece on f6, is Uc6.

A horizontal flip transform applied to a piece designator reflects each square in its square set about the horizontal bisector of the board. For example, the horizontal flip of Uf6 is Uf3.

A dihedral transformation of a piece designator is either a flip transformation or a rotation by 90 degrees of the chessboard followed by a flip transformation or the identity transformation.

There are 8 distinct dihedral transformations of the board. For example, the results of applying the 8 dihedral transformations to the piece designator Qc2 is the 8 piece designators Qc2, Qb3, Qf2, Qg3, Qf7, Qg6, Qb6, and Qc7.

A unit right shift transformation, applied to a piece designator, shifts each square in its square set over to the right one square.

If a square is already at the right edge of the board, it is removed from the square set. If a square is at the left edge of the board, it is also added to the new square set.

For example, the unit right shift transformation of Qd4 is Qe4. The unit right shift transformation of Q[c2,h2] is Qd2 .

The unit right shift transformation of Q[a2,c2,h2] is Q[a2,b2,d2].

The definitions of unit left shift transformation, unit up shift transformation and unit down transformation are similar to that of unit right shift transformation, with of course the location of the appropriate board edges altering mutatis mutandis.

A right transformation is a composition of zero or more unit right transformations.

For example, the right transformation comprising three successive unit right transformations applied to Qa4 is Q[a4,b4,c4,d4], since squares on the left edge of the board are always included in the new set.

The result of applying this right transformation to Qd4 is Qg4 .

Similar definitions apply for left, up, and down transformation.

A horizontal shift transformation is either a left or a right transformation.

A vertical shift transformation is either a down or an up transformation.

A shift transformation is a transformation that can be written as 0 or more horizontal shift transformations followed by 0 or more vertical shift transformations.

There is exactly one shift transformation that sends a given square to another given square.

One possible shift transformation of Q[b3,c4] is Q[d4,e5].

A color flip transformation of a piece designator switches the colors of each piecetype in the piece set of the piece designator, and then applies a horizontal flip to the piece designator.

For example, the color flip transformation applied to [Rbp][a2,c2], either a white rook, a black bishop, or a black pawn on either c2 or a2, is [rBP][a7,c7] .

## Range specifiers

Many keyword options are followed by optional range specifiers.

A range specifier consists of either a single integer or two integers separated by whitespace.

If a range specifier has one integer, it denotes that integer.

Otherwise it denotes all the integers between its first and second integers inclusive.

### Range specifier example

The match list can have a :year keyword denoting the range of years in which the study in question must lie in order for a match to occur.

:year 1934

denotes the year 1934. 1934 is the range specifier.

:year 1934 2000

denotes the years between 1934 and 2000 inclusive. The range specifier is

1934 2000

# The match list

Any CQL file must have exactly one match list.

The match list contains keyword parameters that control the matching of each study.

The match list contains one or more position lists, each of which must match at least one position in the study in order for the match list to match that study.

The keyword parameters of the match list follow

## :pgn

This takes one parameter which denotes the name of the PGN file to search studies for.

A malformed PGN file may cause CQL to crash.

## :output

This takes one parameter which denotes the name of the PGN file to output the results to.

## :year

This takes a range specifier denoting the range of years that the study must match.

## :result

This takes a single parameter specifying the required result of the study or study.

The allowed parameters are 1/2-1/2 , 1-0 , and 0-1 .

They denote respectively a drawn study, a study in which white wins, and a study in which black wins.

The parameter 1/2-1/2 may be abbreviated to 1/2 .

## :gamenumber

This takes a range specifier.

Only studies whose number within the PGN file lies within the range specifier are considered.

## :composer

This takes one parameter, a string.

Only studies at least one of whose composers names has as substring the given parameter will match.

The string matching is case-insensitive.

## :forany

This takes two parameters, a tag name and a piece designator. See [CQL Tagging](http://www.arves.org/arves/index.php/en/25-tools/353-cql-tagging).

## Example of match list

Here is a match list that uses each parameter except :forany.

It has one empty position list as well.

(match

:pgn heijden.pgn

:output out.pgn

:composer costeff

:result 1-0

:gamenumber 10 1000

:year 1934 2000

(position)

)

This match list will look in studies 10 through 1000 in heijden.pgn which end in a white win and were played (or composed) between 1934 and 2000, by any composer whose name matches "costeff".

# The Position List

A position list (or position filter) is applied to a position and it either matches or it does not.

The keywords in the position list determine how the match is performed, and the piece designators define what is matched.

Some keywords themselves take arguments that are lists of positions.

Some keywords modify the way other options work, perform their matching.

By default, when a position filter matches a position in a study, if the study itself matches then the study is printed out with the word MATCH preceding the position that matches.

## Piece designators inside a position

The major work in a position is usually done by its piece designators.

Any position list can have 0 or more piece designators.

Without any keywords, a position filter matches a position if and only if each piece designator matches that position.

For example, the position list:

(position Ra3 )

will match any position with a white rook on a3.

The position list

(position [RQ][a1-8])

will match any position with a white major piece (queen or rook) on the a-file.

The position list

(position [RQ][a1-8] q[h1-8])

will match any position with a white major piece on the a-file and a black queen on the h-file.

The position list

(position .d4 id3 U[a1,h1,a8,h8])

will match any position with an empty square on d4, a black minor piece on d3, and a piece on one of the corners.

Note that in consequence of this repeating piece designators has no effect.

The position lists:

(position R Q)

and

(position R R Q Q)

each match any position with a white rook and a white queen.

## Simple keywords in a position list

Some keywords take no arguments. These are called simple keywords.

Transform keywords will be discussed in the next section.

These simple keywords are:

### :mate

Match only if the position is a mate.

### :stalemate

Match only if the position is a stalemate

### :wtm

Match only if white is to move

### :btm

Match only if black is to move

### :variations

Look in the variations to find matches

### :variationsonly

Look only in the variations to find matches, not in the main line

### :initial

This is the first position in the study or study

### :terminal

This is the last position in the study or study

### :markall

Do not stop after the first match is found, but keep looking.

### :check

One side is in check

### :nocheck

Neither side is in check

### :noannotate

Do not print "MATCH" when this position list matches a position in the study or study.

### Examples using simple keywords

This position list

(position R Q :mate)

matches a position with a white rook and a white queen in which one side is in mate.

We can force white to be in mate:

(position R Q :mate :wtm)

Now white must be mated.

This

(position R Q :initial)

only matches if White has a rook and queen in the first move (of the study, presumably).

We can force white to be in check on the first move:

(position R Q :initial :check :wtm)

### Transformation keywords in a position list

Certain keywords within a position list specify that the position list should match a position if some transformation of the position list matches the position.

For example, the :fliphorizontal keyword specifies that the position list matches a position if either the original position list matches the position, or if the position list, when flipped about the horizontal bisector, matches the position.

To apply a transformation to a position list we apply the transformation to each piece designator that occurs in the position, and we recursively apply the transformation to any other position lists contained in that position list. If the transformation is a color flip translation, then we also flip interchange any :btm keywords with :wtm ones, and we flip the :result keyword arguments appropriately.

For example, consider the position list

(position Rc2 Bh8 :wtm)

This matches any position with white to move that has a white rook on c2 and a white bishop on h8.

The result of applying a unit left shift transformation to this position list is the new position list:

(position Rb2 Bg8 :wtm)

that matches any position with white rook on b2 and white bishop on g8 with white to move.

Similarly, the result of applying a horizontal flip to this position list is the position list:

(position Rb8 Bg1 :wtm)

The result of applying a color flip transformation to the

(position Rb2 Bg8 :wtm)

is thus

(position rb7 bg1 :btm)

Remember that shifts interact specially with certain squares on the edge.

For example the result of applying a unit up shift to this position list:

(position R[a1,a8,h1,a8])

is the position list:

(position R[a1,a2,h1,h2])

A transformation set is any set of transformations.

A transformation set applied to a position list is the set of all position lists that result from applying any transformation in the transformation set to the position list.

Each transformation keyword is associated to a transformation set, each of which also includes the identity transformation.

The transformation keywords and their associated transformation sets are:

### :flipdihedral

all dihedral transformations. Some chesspositions can be rotated (4x) and mirrored (2x) and are still valid.

So a chessposition can be transformed to a maximum of 8 when there are no pawns and when there is no castling possible.

### :flip

same as

:flipdihedral

### :fliphorizontal

the horizontal flip transformation.

### :flipvertical

the vertical flip transformation.

### :flipcolor

the color flip transformation.

### :flipdiagonal

the diagonal flip transformation.

### :shifthorizontal

the horizontal shift transformations.

### :shiftvertical

the vertical shift transformations.

### :shiftmaindiagonal

the main-diagonal shift transformations.

### :shiftoffdiagonal

the off-diagonal shift transformations.

### :shiftdiagonal

the off-diagonal and main-diagonal shift transformations.

### :shift

          the shift transformations.

### Examples of transformation keyword uses

Consider this position list:

(position Rc3 qg3)

which matches any position with a white rook on c3 and a black queen on g3.

We can add a :shifthorizontal keyword:

(position Rc3 qg3 :shifthorizontal)

The set of transformed position lists is:

(position Rc3 qg3)

(position Rb3 qf3)

(position Ra3 qe3)

(position Rd3 qh3)

A position will match the new position list, if there is a white Rook and black queen on the third rank and the white rook is four squares to the left of the black queen.

Similarly, the position list:

(position Rc3 qg3 :shiftvertical)

will match any position in which there is a white rook on the same rank as a black queen such that the rook is on the c file and the queen on the g file.

The position list:

(position Rc3 qg3 :shift)

will match any position with a white Rook and a black queen on the same rank, in which the white rook is four squares to the left of the black queen.

The position list:

(position Rc3 qg3 :fliphorizontal)

will match any position in which either the white rook is on c3 and the black queen is on g3, or the white rook is on c6 and the black queen on g6.

The position list:

(position Rc3 qg3 :flipdiagonal)

will match a position in which either a white rook is on c3 and a black queen on g3, or in which a white rook is on c3 and a black queen is on c7.

The position list:

(position Rc3 qg3 :flip)

The following position list matches any position in which a white rook is somewhere behind a white pawn on the c file:

(position Pc2 Rc1 :shiftvertical)

Because of the way edge squares are treated, this will match a position in which, say, the white rook is on c2 and the white pawn on c7.

To match any position in which a white rook is behind a white pawn on the same file, use:

(position Pc2 Rc1 :shift)

### Multiple transformation keywords

When multiple transformation keywords occur, the associated transformation set is any transformation that can be expressed as a composition of transformations from the corresponding transformation sets.

This feature is most often used when looking for a piece configuration that can occur anywhere on the board, possibly rotated or reflected: adding :shift :flip to the position specifier does this.

For example, this position list

(position Nd4 kf3 :shift :flip)

matches any position in which a white knight attacks a black King, while

(position Nd4 kf3 :shift :flip :flipcolor)

matches any position in which a knight attacks a king of the opposite color.

Note: This example can be written much more efficiently as

(position :attackcount N k 1 100 :flipcolor)

however.

## Piece configuration keywords

Some position list keywords apply the geometrical configuration of the pieces themselves.

Most of these can take a range specifier argument as well which counts the number of times the configuration occurs in the position.

These piece configuration keywords are:

### :piececount

This keyword takes a piece designator argument followed by a range specifier.

It matches the position if and only the number of occurrences of the piece designator in the position is given by the range specifier.

For example,

(position :piececount R 2)

will match any position with exactly two white rooks - a position with three white rooks would fail to match.

The position list

(position

:piececount [Rr][a1-8] 1 3

:piececount [Qq][a-h8] 4

Nd4)

will match any position with a white knight on d4, between one and three rooks on the a-file, and exactly four queens on the eight rank.

Another way to write this position list is:

(position

Nd4

:piececount [Rr][a?] 1 3

:piececount [Qq][?8] 4

)

To find Miniatures (up to 7 figures in the starting position) with at least and one piece:

(position :initial :piececount U 3 7)

### :power

This keyword takes as argument a piece designator followed by a range specifier. It matches positions for which the total power of all pieces in the position matching the piece designator lies within the range specifier.

The power of a piece is their standard chess material power: Q=9, R=5, B=3, N=3, P=1, K=0.

For example,

(position :power a 6)

will match positions in which the total power of the black pieces on the board equals 6.

(position :power Aa-h1-4 10 25)

will match positions in which the total power of the white pieces on the white half of the board is between 10 and 25 inclusive.

### :powerdifference

This keyword takes as argument a piece designator followed by a range specifier. It matches positions in which the difference in power between the white and black pieces matching the piece designator (that is, the numeric value of the power of the matching black pieces subtracted from the power of the matching white pieces) lies within the range specifier.

For example,

(position :powerdifference U 4)

matches positions in which the white pieces have power precisely 4 greater than the black pieces, (U stands any piece of either color).

(position :powerdifference [Rr] -5 10)

matches positions in which white has between one fewer and two more rooks than black, (a rook has a value of 5, so 10 means 2 rooks).

(position :powerdifference [RrBbNn] -1000 -2)

matches positions in which black is up at least an exchange, (-1000 is an indifferent high chosen low value).

[This Example](http://www.arves.org/arves/index.php/en/25-tools/378-cql-queen-sacrifice) shows how to search for queen sacrifices using :powerdifference.

### :attackcount

This keyword takes as arguments two piece designators followed by a range specifier.

The first piece designator is the attacker piece designator.

The second piece designator is the attacked piece designator.

An attack by a piece to a square occurs if the piece could move to that square were it empty and disregarding any possible checks.

A valid attack is an attack by a piece on a square that matches the attacker piece designator to a square on which is a piece that matches the attacking piece designator.

The :attackcount keyword matches a position if the total number of valid attacks in the position lies with the range given by its range specifier.

For example,

(position :attackcount A k 2)

matches any position in which white has a double attack on the black king.

(position :attackcount A k 2 :flipcolor)

matches any position in which a king is under double attack.

(position :attackcount A .h3 1)

matches a position in which exactly one white piece attacks an empty h3 square.

(position :attackcount [RB][a-c8] ??1 5 9

matches a position in which the number of times a white rook or white bishop on a8, b8, or c8 attacks a square on the first rank is between 5 and 9 inclusive.

(position :attackcount n [KRQ] 3 100)

matches a position for which the number of times a black knight attacks a white King or white major piece, is at least 3.

### :ray

A ray specifier is a list of piece designators.

A ray is a contiguous set of squares on a straight line on the chessboard parallel to an axis, the a1-h1 diagonal, or the a8-h8 diagonal.

Let R by a ray with ordered set of squares in a contiguous line S1, S2,...,Sn.

Suppose spec is a ray specifier (G1, G2,...,Gk) where each G1,...,Gk is a piece designator.

The ray R is valid with respect to the ray specifier if G1 matches the piece on S1, and if Gk matches the piece on Sn, and if the remaining G's can be paired to some subset of the squares in the ray in an order-preserving way such that each G matches the piece on its corresponding square and such that all unpaired squares in the ray are empty.

The :ray keyword takes a ray specifier and an optional range specifier.

If the range specifier is absent, it is taken to be 1 1000. Otherwise, :ray matches a position if the number of rays in the position valid with respect to the ray specifier lies within the range specifier.

### :ray examples

(position :ray (k Q))

matches a position in which a white queen attacks the black king.

(position :ray (Qa? n k[?5-8]))

matches a position in which a white queen on the a-file pins a black knight to the black king on the last four ranks.

(position :ray (Qa? Qc? Qe? kg?))

matches a position in which either three queens and the black king are on the same rank, with exactly one empty square between them, and one of the queens is on the a-file, or a position in which the three queens and king form a diagonal with one empty squares between successive pieces and the queens on the a, c, and e files.

more exactly, 4 type of positions are found:

Qa1 Qc1 Qe1 kg1 or Qa1 Qc3 Qe5 kg7 or  Qa2 Qc2 Qe2 kg2 or Qa2 Qc4 Qe6 kg8  
or 4 other possibilities being 180 degree transpositions of the above.

(position :ray (Q n k) 2 4)

between two and for black knights are pinned to the black king.

### :raydiagonal, :rayhorizontal, :rayvertical, :rayorthogonal

These keywords function exactly like :ray but the given rays are restricted to being diagonal, horizontal, vertical, and orthogonal respectively.

A horizontal ray is a ray that is horizontal. A vertical ray is a ray that is vertical.

A diagonal ray is a ray that is not vertical or horizontal.

An orthogonal ray is a ray that is not diagonal.

Thus

(position :rayorthogonal (R b k))

matches position in which a black bishop is pinned by a white rook against the black king.

(position :raydiagonal (Q B n k))

matches positions in which there is a white Queen-bishop battery pinning a black knight to the black king.

(position :rayvertical (R[a1-8] b k))

matches positions in which a white rook on the a file pins a black bishop to the black king on the a file.

(position :rayhorizontal (K . . k))

matches positions in which the kings are on the same rank and separated by at least two empty squares.

### :rayattack

has the same syntax as :ray. It has the semantics of :ray, except that only orthogonal rays that start on a square containing a rook or queen are included, and only diagonal rays starting on a square containing a bishop or queen are included.

For example,

(position :rayattack (A a k))

matches a position in which a black piece is pinned to the black king.

(position :rayattack (Ad5 a [rk][a-h1-4]) 3)

matches positions in which there are three black pinned by a white piece on d5 against either a black rook or king in the first four ranks.

(position :stalemate :rayattack (A a k) 2 8 :flipcolor)

finds multiple-pin stalemates.

## Move keywords

In a position, the next move is the move about to be played, if any.

A move has three characteristics: it's from designator, it’s to designator and it's promotion designator.

These refer respectively to the piece and square from which the piece moves; to the piece and square to which it is going to move (but before it actually moves there), and to the piece and square to which it promotes, if any.

For example, the to designation of a white rook on a4 capturing a black knight on d4 is nd4, and it's from designation is Ra4.

The to designation of a move to e4 that is not a capture is .e4.

### :movefrom

takes a single parameter, a piece designator. It matches a move whose from designator matches that piece designator.

For example,

(position :movefrom Ra3)

matches a position in which the next move is of a white rook on a3.

(position :movefrom U?8)

matches a position in the next move is of some piece on the 8th rank.

### :moveto

takes a single parameter, a piece designator. It matches a move whose to designator matches its parameter.

For example,

(position :moveto .a3)

matches a position in which the next move is to an empty square on a3.

(position :moveto R[b1-8])

matches a position in which the next move is to capture a white rook on the b file.

Note that this is quite different from the natural PGN interpration of its being a move of a white rook to the b file.

That would be done by:

(position :movefrom R :moveto ?b?)

which matches a move of the white rook to the b file.

### :promote

takes a single parameter, a piece designator.

It matches a move if that move is a promotion whose promotion designator matches its parameter.

For example,

(position :promote [RBN])

matches a position whose next move is a white underpromotion.

(position :promote [BN]a8)

matches a position whose next move is a white promotion to bishop or knight.

(position :promote [Re8] :movefrom Pf7 :moveto q)

matches a position in which the next move is of a pawn on f7 capturing a black queen on e8 and promoting to a rook.

Castling is considered a move of the king two squares:

(position :movefrom [Kk]e? :moveto .[c?,g?])

matches any position in which the next move is castles.

There are also some special keywords for handling en-passant:

### :enpassant

The next move is an en-passant captue.

### :noenpassant

The next move is not an en-passant capture.

### :movenumber

:movenumber takes a range specifier as parameter and matches positions only if the current move number is within the specified range.

The move number computation assumes white moves first, and might be one off otherwise.

For example,

(position :movenumber 10 20 :check)

matches a position in which there is a check between moves 10 and 20.

(position :movenumber 35 :mate)

matches a position in which move 35 is mate.

## Logical keywords

There are three ways to combine position lists using logical operations: :and, :or, and not.

### :and

takes a list of position lists and matches a position only if each of them match the position.

### :or

takes a list of position lists and matches a position only if at least one of them matches the position.

### :not

matches a position if the containing position list would not match the position without the :not.

For example,

(position

:wtm

:check

:and

((position Ke2 kg2 :shift :flip)

(position Ne2)))

matches positions in which the kings are in opposition, there is a white knight on e2, and white is in check.

(position

:wtm :check

:or

((position Ke1 kg1 :shift :flip)

(position Ne2)))

matches positions in which white is in check and either the kings are in opposition or there is a white knight on e2 (or both).

(position Ra3 :not)

is equivalent to

(position [aQBNPK.]a3)

Meaning a position where on a3 is not a white rook.

## Sequence keywords

Two keywords can be used to match sequences of positions that occur from a given position, :sequence and :gappedsequence.

### :sequence

takes a single argument that is a list of position lists. It matches a given position if and only successive occurring positions, beginning with the current position, match each corresponding element in its argument.

For example, the position list

(position Re8

:sequence

((position Qf3 .g2)

(position .f3 Qg2)))

matches exactly those positions with a white rook on e8 such that:

1. In that position, white has a queen on f3 and g2 is empty, and
2. After the next move, f3 is empty and there is a white queen on g2.

In consequence, this position list is identical in effect to

(position Re8 :movefrom Qf3 :moveto .g2)

The following more typical usage of :sequence will search for all queen staircase checking maneuvers by either side:

(position

:shift :flip :flipcolor

:sequence

((position :movefrom Qa2)

(position :check)

(position :movefrom Qb2)

(position :check)

(position :movefrom Qb3)

(position :check)

(position :movefrom Qc3)

(position :check)

(position :movefrom Qc4)

); end the sequence

); end the position

This position list, matches any position with a white queen on a2 for which there are successively moves by queens on the indicated squares giving check; the :shift, :flip, and :flipcolor search for this variation reflected or rotated anywhere on the board.

### :gappedsequence

takes one argument, a list of position lists.

A position list A is said to be a refinement of a position list B if A may be formed from B by inserting 0 or more position lists of the form (position) into the top level of B.

For example,

((position Na4) (position) (position Rb2))

refines

((position Na4) (position Rb2))

because it was formed by inserting a single position list into the latter list.

A :gappedsequence with argument B matches a position if and only if there is a refinement A of B such that a :sequence with argument A would match that position.

That is, :gappedsequence is like :sequence except that the sequence of matching positions may include intervening positions from the study or study.

For example,

(position

:gappedsequence

((position :movefrom Ra3)

(position :movefrom Rb4)

); end the gappedsequence

); end the position

will match any position for which the positions that occur successively in the study starting with that position comprise 0 or more positions, followed by a position from which a white rook on a3 moves, followed by 0 or more positions, followed by a position from which a white rook on b4 moves.

In consequence the same effect can be obtained much more efficiently via:

(position

:initial

:gappedsequence

((position :movefrom Ra3)

(position :movefrom Rb4)

)

)

The following position list matches a position in which a knight is on d4 at some point in the study after that position:

(position

:gappedsequence

((position)

(position Nd4)

)

)

Note that the first element of the argument to :gappedsequence, (position), is used to ensure that only knights on d4 in positions that occur after the position to be matched are considered.

This fragment is used to find all studies in which some [Knight visits at least twenty different squares.](http://www.arves.org/arves/index.php/en/25-tools/359-cql-a-knight-visits-at-least-twenty-different-squares)

The total number of :sequence and :gappedsequence position lists that can occur at the top level of a position list cannot exceed 1.

To get the effect of both a :gappedequence and :sequence keywords in one position list, use :and .

## Relations between positions

### :relation

takes one argument, a relation specification. is used to search for studies with pairs of positions P1 and P2 that satisfy certain properties.

The relation specification keywords are described [here](https://web.archive.org/web/20110715155947/http:/www.rbnn.com/cql/relation.html).

## Game-related position list keywords

Several keywords relate to the game parameters and are not affected by the contents of the position; they are included as position list keywords instead of match list keywords so the :flipcolor can be used with them.

These are:

### :elo

this parameter expects a range specifer; the ELO of at least one of the composers must lie within the specified range

### :whiteelo

this parameter takes a range specifer; the ELO of the white composer must lie within its range

### :blackelo

this parameter takes a range specifier; the ELO of the black composer must lie within its range.

### :result

this parameter takes one argument, the same as the argument to :result in a match list.

For example,

(position

:result 1-0

:whiteelo 0 2300

:blackelo 2600 3000

:flipcolor

)

searches for positions in which a 2300 composer or below defeated a 2600 composer or above.

## Matching count keywords

Two keywords pertain to the number of positions which a position list matches. These each take a range specifier as parameter. They can only be used in a position list at the top level of a match list.

### :matchcount

matches if and only if the number of times this position list matches a position in the current study, lies within the range specifier.

For example,

(position :moveto [Qq] :matchcount 4 100)

matches any study in which at least four queen captures have occurred.

### :pretransformmatchcount

has the same syntax as :matchcount.

It matches a position if and only if there is some transform in the transformation set associated with the position list in which it occurs such that, were the position list replaced by a new position list formed from the first by deleting all transformation keywords, applying the transform to that position list, and changing this keyword to :matchcount, then the position list would match the position.

For example,

(position :moveto [Aa]d4

:pretransformmatchcount 12 1000 :shift)

)

would match any study in which at least 12 captures occur on the same square.

If the :pretransformmatchcount is replaced by :matchcount , the position list would match studies in which at least 12 captures occurred.

The "MATCH" string is printed independent of any count keywords.

## Accumulator keywords

(Note: this feature is experimental). There are two accumulator keywords, :accumulate and :sumrange .

These are used to accumulate and to test the number of transforms of the enclosing position list that match the current position.

Each time a transform of a position list matches a position, the accumulator associated with that position is incremented.

Any position with a sum range can then test if the accumulator is within a certain range.

### :accumulate

This takes one parameter, the name of an accumulator.

Each time the application of some element of the transform set of the position list matches a position, the accumulator associated with this name is incremented.

The accumulator is cleared when a new position is reached (and in consequence :accumulate should not be used inside of :not or the sequence keywords. This keyword inhibits short-circuiting of logical keywords :and and :or .

Note that :flipcolor likely has unexpected results when used with :accumulate, because the accumulator is not cleared between color flips.

### :sumrange

This takes a parameter the name of an accumulator and a range specifer.

It matches a position if the value of the given accumulator lies within the range specifier.

### Using accumulators to count passed pawns

We will show how to use :accumulate and :sumrange to look for studies that have at least 3 white passed pawns in a position.

First, we give a position filter that will match any position with a white passed pawn:

(position ;match a position with a white passed pawn

Pd2

:piececount [pP][d3-7] 0

:piececount p[c3-7,e3-7] 0

:shift

)

Now, to find at least 3 white passed pawns, we add in some accumulation code:

(position ;match a position with at least three white passed pawns

Pd2

:piececount [pP][d3-7] 0

:piececount p[c3-7,e3-7] 0

:shift

:accumulate white\_passer\_counter

:sumrange white\_passer\_counter 3 8

:markall

)

Now suppose we want to find positions with at least two black passed pawns.

We can use:

(position ;match a position with at least two black passed pawns

pd7

:piececount [pP][d2-6] 0

:piececount P[c2-6,e2-6] 0

:shift

:accumulate black\_passer\_counter

:sumrange black\_passer\_counter 2 8

:markall

)

Finally, to find positions with at least two black passers and at least three white passers, we combine these lists:

(position ; match a position with at least three white and at least two black passers

:sumrange white\_passer\_counter 3 8

:sumrange black\_passer\_counter 2 8

:markall

:and

((position

Pd2

:piececount [pP][d3-7] 0

:piececount p[c3-7,e3-7] 0

:shift

:accumulate white\_passer\_counter

)

(position

pd7

:piececount [pP][d2-6] 0

:piececount P[c2-6,e2-6] 0

:shift

:accumulate black\_passer\_counter

)

)

)

## Tagging keywords

The :tagmatch keyword takes two parameters, the name of a tag and a piece designator.

It matches only if the piece corresponding to the named tag matches the given piece designator.

For more information, see the section on [CQL Tagging: keeping track of piece identity](http://www.arves.org/arves/index.php/en/endgamestudies/chess-query-language/cql-tagging-keeping-track-of-piece-identity).

# The :relation keyword

The :relation keyword occurs in a position list. It takes one parameter, a relation specifier.

A relation specifier is a list denoting parameters to the relation keyword.

For example, the following position list,

(position

:relation (:missingpiececount A 1 100))

searches for studies in which two positions are the same except that the later one is missing one or more white pieces.

The operation of the :relation tag is as follows.

A position list with a relation tag matches a position P1 if and only if at some later point in the current study there is a position P2 such that the relationship between P1 and P2 matches the parameters defined in the relation specifier.

The meaning of each keyword inside a relation specifier is as follows.

### :pattern

The :pattern keyword signifies that all piece designators that occur at the top level of the position list that encloses the current relation list define the allowable squares.

Only allowable squares are considered in the comparisons.

By default all squares are allowable.

### :originaldifferentcount

This parameter takes a range specifier. It matches if the number of allowable squares that are different between P1 and P2 lies within the range specifier.

### :originalsamecount

This parameter takes a range specifier. It matches if the number of nonempty allowable squares that are the same between P1 and P2 lies within the range specifier.

### :shift

Allow P2 to be shifted before the comparison. However, in no case is P2 allowed to have a different piece on a square than P1 after the transform is applied.

### :flip

Allow P2 to be flipped before the comparison. However, in no case is P2 allowed to have a different piece on a square than P1 after the transform is applied.

### :samesidetomove

P2 must have the same side to move as P1.

This is the default.

### :changesidetomove

P2 must have a different side to move as does P1.

### :ignoresidetomove

Side to move information is ignored.

### :missingpiececount

This keyword takes the same parameters as does the :piececount keyword in a position list: a pieced designator followed by a range.

A missing piece is a piece that occurs in P1 on a square that is empty on P2. P2 matches if the number of these missing pieces lies within the range specifier. Unlike the case of the :piececount keyowrd, at most one :missingpiececount specifier can occur.

No shift or flip transformation specifiers in the containing position list affect the piece designator in this keyword.

The default piece designator for :missingpiececount is U and the default range is 0 .

That is, by default there can be no missing pieces.

### :newpiececount

This keyword takes the same parameters as does the :piececount keyword in a position list: a pieced designator followed by a range.

A new piece is a piece that occurs in P2 on a square that is empty on P1. P2 matches if the number of these new pieces lies within the range specifier.

Unlike the case of the :piececount keyword, at most one :newpiececount specifier can occur. No shift or flip transformation specifiers in the containing position list affect the piece designator in this keyword.

The default piece designator for :newpiececount is U and the default range is 0 .

That is, by default there can be no new pieces.

### :variations

Search for P2 within variations and in the main line. By default, P2 must be in the main line.

### :variationsonly

Search for P2 only within variations.

## Relation list examples

To search studies in which the same position occurs with white and black to move - for example, this is useful in looking for mutual zugzwangs, use

(position :relation (:changesidetomove))

The following position list searches for studies in which the configuration

Ke6 be7 be4 nd5

occurs shifted or flipped, but in which no pieces from the original position are on the same square as the pattern:

(position

Ke6 be7 be4 nd5

:shift

:flip

:markall

:relation (:pattern

:shift

:flip

:originalsamecount 0

:samesidetomove

:variations

)

)

If one modified the :originalsamecount parameter to, say, 0 3 from its current value of 1, then the pattern in the second position would be allowed to overlap with at most three squares of the pattern in the original position.

# Tagging

It is possible to track the identity of a particular piece.

Each piece in the initial position of the study or study is assigned a unique nonnegative integer, called its mark.

Promotion does not change the mark of a piece.

A tag is a named mark. We say that a tag is bound to the value of a mark.

A tag represents the particular piece whose mark equals the mark of the tag. The following keywords manipulate the mark:

## :forany in the match list

A tag is defined by the :forany keyword in a match list. The :forany keyword takes two parameters: the alphanumeric name of a tag and a piece designator.

The match filter is then conceptually run once in which each tag is bound to each possible mark, such that two conditions hold:

* No two tags may be bound to the same mark.
* A tag may only be bound to a mark that represents a piece that either matches, or eventually matches, the piece designator associated with that tag. Square information associated with that piece designator is ignored.

At most two tags may be contained inside a match list.

Tags may not be used in a matchlist if a :relation occurs in a position list inside that match list.

## :tagmatch in the position list

A :tagmatch keyword in a position list takes two parameters, the name of a tag and a piece designator.

The position list match fails on the current position if the piece and square corresponding to the mark of the tag does not match the given piece designator. For example, assuming there is a tag named "foo", this position will only match if "foo" is a white pawn:

(position :tagmatch foo P)

## Piece designators and tags

A piece designator can have a piecetype designator that is the name of a tag preceded by the character '$'.

Such a piecetype represents the particular piece represented by the named tag. Because tag names are alphanumeric, square designators in such a piece designator must be enclosed in brackets.

For Example:

(position $foo[a1,d4])

The piece represented by the tag named "foo" must be on a1 or d4.

## Longer example

Consider the problem of recognizing all studies or studies in which the same rook visits all four corners of the board.

This is shown in the file [CQL Rook visits each corner](http://www.arves.org/arves/index.php/en/25-tools/384-cql-rook-visits-each-corner), and is reproduced here:

; Find all studies in which the same rook visits all four corners of the board.

(match

:pgn heijden.pgn

:output out.pgn

:forany rook [rR] ; loop over the possible rooks

(position $rook[a1])

(position $rook[h1])

(position $rook[h8])

(position $rook[a8])

)

Here, the :forany keyword in the match list introduces a tag named rook which is constrained to range over white or black rooks, or over pawns that eventually promote to one of those.

The first position list matches studies in which there is a position such that the tag rook represents a rook on a1.

The next position list matches studies in which there is a postiion such that the tag represents a rook on h1.

The tag represents the same mark, that is, the same piece, between the position lists. The :forany keyword will sequentially bind the tag to each allowed mark.

Since a match list matches a study only if all its constituent position filters match it, the code does what we want here.

## Other examples of tagging

The files

* [Excelsior](http://www.arves.org/arves/index.php/en/25-tools/354-cql-excelsior) A pawn starts on its second rank and promotes.
* [Double Excelsior](http://www.arves.org/arves/index.php/en/25-tools/355-cql-double-excesior) Both a white and black pawn have excelsiors; the white pawn underpromotes
* [Many checks](http://www.arves.org/arves/index.php/en/25-tools/385-cql-many-checks), The same piece delivers at least 30 checks in the study or study.
* [Rook traverses a rectangle](http://www.arves.org/arves/index.php/en/25-tools/357-cql-a-rook-traverses-a-rectangle) in this case, a 4x3 rectangle.
* [Knight chases other Knight](http://www.arves.org/arves/index.php/en/25-tools/358-cql-knight-chases-other-knight) constantly trying to sacrifice itself.
* [Knight visits at least twenty different squares.](http://www.arves.org/arves/index.php/en/25-tools/359-cql-a-knight-visits-at-least-twenty-different-squares)

present some other examples of tagging.

# CQL FAQ

You might first read the [chess\_query\_language.pdf](http://www.arves.org/arves/images/PDF/chess_query_language.pdf) and look at the examples before checking this FAQ.

## Why is the documentation so difficult to understand?

The original manual.txt is no longer available.

Therefore here is an [CQL\_Manual.docx](http://www.arves.org/arves/images/DOC/CQL_Manual.docx) file free to download with all the CQL explanation articles from this Arves website. It has an keyword index as well.

However, if there are still questions, we are happy to help answer them and to help with CQL files. Just send an email to Gady Costeff.

Some more explanation can be found at [ChessOK](http://chessok.com/?p=21989) and here about [VisualCQL](http://portablegamenotation.com/cqlvisual.html).

## What is the latest version?

Version 3.02

This version supports [Tagging](http://www.arves.org/arves/index.php/en/endgamestudies/chess-query-language/cql-tagging-keeping-track-of-piece-identity) , which keeps track of piece identity.

## Why am I getting anomalous results when I use :shift and a piece is on the edge of the board?

When a piece designator refers to a square on the left edge of the board, for example, and a right shift is performed on the position, the new designator continues to include the original square on the left edge.

For example,

(position

Pa4 Nd4 :shifthorizontal)

will match any position in which White has a knight on d4, e4, f4, g4, or h4 and in which White has a pawn on the fourth rank at least three squares to the left of that knight.

If, conversely, one wanted to search for positions in which White has a knight on d4, e4, f4, g4, or h4 and in which White has pawn on the fourth rank exactly three squares to the left of the knight, then the following fragment would be used:

(position

Pb4 Ne4 :shifthorizontal)

Similar rules apply for shifts in other directions: pieces are always "shifted in" to the edge of the board.

It is worth noting that this feature does give pattern matching power that would be difficult to obtain using other features. For example, consider the following fragment:

(position

Pa4 Pa5 Nd4 Nd5 :shifthorizontal)

This fragment would match positions with a pair of pawns, one behind the other, on the fourth and fifth ranks, at least three files to the left of a similarly arranged pair of knights. This pattern could not be readily expressed using other primitives without this "shift-in" feature.

## How does :moveto work?

:moveto takes a piece designator, not a square.

Thus,

:moveto d4

will not work. You would have to use

:moveto ?d4

## See [CQL\_Manual.docx](http://www.arves.org/arves/images/DOC/CQL_Manual.docx) for other examples.

## How can I check for an unguarded guard?

Unguarded guard is a theme in which

* White is in check
* White moves a piece to an empty square to stop the check
* After moving the piece, that piece is unguarded

Use this form (use :flipcolor to search for black avoiding check and so on, of course).

;;Find positions with white to move such that white

;;is in check from a sliding piece and

;;on white's next move interposes a piece that is unguarded

(match

:pgn heijden.pgn

:output out.pgn

(position

:wtm

:moveto .d4

:check

:or

((position :rayorthogonal ([qr] .d4 K))

(position :raydiagonal ([qb] .d4 K))

)

:sequence

((position)

(position :attackcount A Ad4 0)

)

:shift :markall)

)

In fact Tim Krabbé observes the :ray specifiers are superfluous here, since a white move to an empty square when in check must stop a ray check.

## How about an unguarded guard in which black then captures the interposing white piece and another piece interposes afterward to stop the check?

That is, we are looking for a position such that:

* White is in check
* White moves a piece to an empty square to stop the check
* After moving the piece, that piece is unguarded
* That piece is then captured by Black with check
* White is still in check, and interposes another piece to an empty square to stop this second check.

Note the use of a "dummy square" d4, in conjunction with a :shift.

;;Find positions with white to move such that white is in check from a sliding piece and

;;on white's next move interposes a piece that is unguarded.

;;Then black captures that piece in such a way that white remains

;;in check and white subsequently interposes another piece to stop the check

(match

:pgn heijden.pgn

:output out.pgn

(position

:wtm

:moveto .d4

:check

:or

((position :rayorthogonal ([qr] .d4 K))

(position :raydiagonal ([qb] .d4 K))

)

:sequence

((position)

(position :attackcount A Ad4 0

:movefrom [rqb] ; this might pick up some extra captures from the wrong piece

:moveto Ad4)

(position :check :moveto . :movefrom [QRBNP])

)

:shift :markall)

)

## In the Dobrescu example, dobrescu.cql, should :originalsamecount 0 be :originalsamecount 0 1 ?

This position filter was:

(position

Ke6 be7 be4 nd5

:shift

:flip

:markall ; find each occurrence

:relation (:pattern ; only pay attention to the pieces we found

:shift ; allow shift

:flip ; allow flip

:originalsamecount 0 ; disregard identity matches

:samesidetomove ; same side to move

:variations ; look in variations

)

)

It depends what you are looking for.

:originalsamecount 0

will find the Dobrescu study, for which this CQL file was written.

It will not find positions that it is not designed to find. In particular, it will not find positions in which any of the pieces in the target overlap any of the pieces in the pattern. It is reasonable to use

:originalsamecount 0 1

as well, however. It all depends on what you are trying to find.

Try it both ways and see what new positions you get!

## How can I look for a passed pawn on the b file?

Use this position filter

(position Pb2 :piececount p[a-c3-7] 0 :shiftvertical :flipcolor)

## Does :flipcolor affect :result?

No. Because CQL was originally designed to search for studies, and there are no 0-1 studies, this feature was not implemented.

## How does one represent light-squared or dark-squared bishops?

The following can be cut-and-pasted into a CQL file, and is the piece designator for a light-squared white bishop:

B[a1,a3,a5,a7,b2,b4,b6,b8,c1,c3,c5,c7,d2,d4,d6,d8,e1,e3,e5,e7,f2,f4,f6,f8,g1,g3,g5,g7,h2,h4,h6,h8]

A dark-squared bishop is:

B[a2,a4,a6,a8,b1,b3,b5,b7,c2,c4,c6,c8,d1,d3,d5,d7,e2,e4,e6,e8,f1,f3,f5,f7,g2,g4,g6,g8,h1,h3,h5,h7]

See [samecolorbishops](http://portablegamenotation.com/samecolorbishops.html) for an example.

## Why does CQL crash on a simple .cql file?

The only known cause of CQL crashing is due to a malformed PGN file.

Some large Chessbase databases do contain invalid studies which, when converted to CQL, cause the crash.

You can use the output of CQL to pinpoint the approximate location of the bad study and delete it manually.

The easiest way to do this is to view the output of the console, which prints a dot every 100 studies.

Suppose you have narrowed the crash so you know it occurss somewhere between study 650500 and 650700.

Then create a new .cql file like this:

(match :pgn baddatabase.pgn :output out.pgn

:gamenumber 650499 650701

(position)

)

Because CQL prints a message with the game number at every matched study, this will print the number of the last correct study matched; the number one higher than that is the invalid study. This game should then be deleted either from the PGN database or the Chessbase database (if you are using chessbase to generate the PGN) that has the corrupt game. If the latter you must then regenerate the PGN database.

## **Acknowledgments:**

- Creators: Gady Costeff and Lewis Stiller.

- Portions of the code use Scid by Shane Hudson.

- The Windows build uses the Cygwin libraries.

- There is a VisualCQL tool available on the website of [Emil Vlasak](http://www.vlasak.biz/vcql.htm)

Index.

:accumulate 48

:and 41

:attackcount 35

:blackelo 47

:btm 26

:changesidetomove 53

:check 27

:composer 23

:elo 46

:enpassant 41

:flip 29, 53

:flipcolor 30, 46, 60, 65

:flipdihedral 29

:fliphorizontal 28, 30, 31

:flipvertical 30

:forany 24, 56

:gamenumber 65

:gappedsequence 44

:ignoresidetomove 53

:initial 27

:markall 27

:matchcount 47

:mate 26

:missingpiececount 53

:movefrom 39

:moveto 39, 60

:newpiececount 53

:noannotate 27

:nocheck 27

:or 41

:originaldifferentcount 52

:originalsamecount 52, 63

:output 14, 23

:pattern 52

:pgn 14, 23

:piececount 14, 33, 53, 65

:power 34

:powerdifference 34

:pretransformmatchcount 47

:promote 40

:ray 36

:raydiagonal 37, 63

:rayhorizontal 37

:rayorthogonal 37, 63

:rayvertical 37

:relation 46, 52

:result 23, 47, 65

:samesidetomove 53

:sequence 43

:shift 30, 32, 53

:shiftdiagonal 30

:shifthorizontal 30, 31, 59

:shiftmaindiagonal 30

:shiftoffdiagonal 30

:shiftvertical 30, 31, 32

:stalemate 26

:sumrange 49

:tagmatch 51, 56

:terminal 27

:variations 27, 54

:variationsonly 27, 54

:wtm 26

:year 22, 23

? 18

allowable squares 52

any black piece 18

any piece 18

any piece at all 18

any white piece 18

black major piece 18

black minor piece 18, 26

brackets 19

color flip transformation 21

commas 19

Comments 15

compound square designator 19

crash 65

Cygwin libraries 66

dark-squared bishops 65

diagonal flip 20

dihedral 20

Dobrescu 63

download 1

dummy square 62

Emil Vlasak 66

empty square 17, 18

empty square designator 18, 19

Excelsior 58

file designator 19

find quadrupled pawns 12

gamenumber 23

horizontal flip 20

horizontal shift transformation 21

**Installing** 10

keyword 13

keyword parameter 13

light-squared white bishop 65

list 13

main diagonal 20

mark 56

match list 14, 23

no squares 19

non-king piece 20

not 41

off-diagonal flip 20

Open Chess Diary 1

opposition 42

passed pawns 50

PGN database 66

piece designator 16

Piece designators inside a position 25

piecetype designator 56

piecetype designators 17, 18

position filter 14, 25

position list 14, 25

power of a piece 34

Promotion 56

Range specifiers 21

rayattack 38

rectangle 58

right transformation 21

semicolon 15

shift transformation 21

simple keywords 26

Square designators 18

Tagging 56

Tim Krabbé 1

transforms 20

underpromotes 58

underpromotion 40

unguarded guard 62

union of the sets of squares 19

unit left shift 21

unit right shift 20

vertical flip 20

vertical shift transformation 21

Visual CQL 10

VisualCQL 59, 66

white major piece 18

white minor piece 18

whiteelo 46