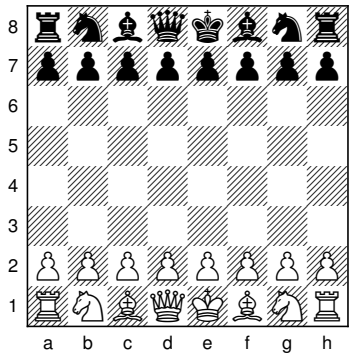
chessboard: A package to print chessboards

Ulrike Fischer

November 1, 2020



Contents

1 Changes **1**

2 Introduction **2**
2.1 Bugs and errors . . . 3
2.2 Requirements . . . 4
2.3 Installation . . . 4
2.4 Robustness: using \chessboard in moving arguments . . . 4
2.5 Setting the options . . . 5
2.6 Saving optionlists . . . 7
2.7 Naming the board . . . 8
2.8 Naming areas of the board . . . 8
2.9 FEN: Forsyth-Edwards Notation . . . 9
2.10 The main parts of the board . . . 9

3 Setting the contents of the board **10**
3.1 The maximum number of fields . . . 10

3.2 Filling with the package skak . . . 11
3.3 Clearing . . . 12
3.4 Adding single pieces . . . 12
3.5 Adding FEN-positions . . . 13
3.6 Saving positions . . . 15
3.7 Getting the positions of pieces . . . 16
3.8 Using saved and stored games . . . 17
3.9 Restoring the running game . . . 17
3.10 Changing the input language . . . 18

4 The look of the board **19**
4.1 Units for lengths . . . 19
4.2 Some words about box sizes . . . 19
4.3 Margins . . . 20
4.4 Borders . . . 21
4.5 The size of the boardfont . . . 23
4.6 Changing the boardfont . . . 23
4.7 Coloring and emphasizing the board chars . . . 25
4.7.1 In short . . . 25
4.7.2 Introduction: About composed chars and encodings . . . 25
4.7.3 Setting the default colors of the board . . . 28
4.7.4 Applying the colors to the whole board . . . 29
4.7.5 Emphasising and coloring individual areas . . . 30
4.7.6 Transparency/opacity . . . 32
4.8 Labels . . . 32
4.9 The mover . . . 35

5 Controlling the printing **38**
5.1 Printing partial boards . . . 38
5.2 Rotating the board . . . 39
5.3 Hiding the content of the board . . . 40
5.3.1 Hiding the content of fields . . . 40
5.3.2 Hiding piecetypes . . . 40
5.3.3 Showing the content of fields . . . 41
5.3.4 Showing piecetypes . . . 42

6 “Decoration”: Colors, background, fancy borders, highlighting **43**
6.1 The pgf-pictures . . . 43
6.1.1 Naming of the pgf related keys . . . 44
6.1.2 Executing the drawing commands . . . 44
6.1.3 Drawing on and under fields . . . 45
6.1.4 Drawing on regions . . . 46
6.1.5 Drawing move related . . . 47
6.1.6 Choosing what is drawn: pgf styles . . . 48

6.1.7 Introduction to the predefined pgf styles . . . 49
6.1.8 Setting graphic properties . . . 54
6.1.9 A special shortcut key for background border . . . 59
6.1.10 Special shortcut keys for background color . . . 60
6.1.11 Clearing the pgf-pictures . . . 61
6.1.12 Clipping the pgf pictures . . . 63
6.1.13 Trimming . . . 64
6.2 Using a graphic as background . . . 68
6.3 Using the commands of the package skak . . . 69
6.4 Intelligent highlighting . . . 70

7 Extending the game **70**
7.1 Adding new pieces . . . 70
7.2 Using \chessboard for other games . . . 71

8 Compability with other packages **72**
8.1 skak . . . 72
8.2 texmate . . . 72
8.3 beamer . . . 72
8.4 animate . . . 73

1 Changes

Attention!

From version 1.5 on the documentation uses the (new) package xskak instead of skak. The most notable difference is that in some examples \newchessgame instead of \newgame is used. But in most cases all examples should work with skak alone too. The package xskak adds quite a lot new keys to \chessboard to show arbitrary positions from previously parsed games. Read its documentation if you want to use them.

In version 1.3 I made a lot of changes. I tried to preserve the behaviour of existing keys and commands. But is possible that older documents will look different when processed with this version.

Notably two things can give problems: The value of the padding key now affects much more objects (marks) in the pgf pictures. So it could be necessary to reset the padding. And the applycolor has changed its behaviour. It will now also affect the foreground picture.

2020-11-01 (Version 1.9) Corrected a bug reported by Phelype Oleinik. colorbackareas (and other area keys) didn't process lists of areas correctly.

2019-06-23 (Version 1.8) Uploaded the source of the documentation to ctan. No longer use \arabic internally, to avoid problems with packages redefining the command. (Issue #1).

2011-03-17 (Version 1.7) Changed definition of the triangle mover style. It now uses tikz and no longer amssymb. chessboard no longer loads amssymb (it clashes with xunicode).

2011-03-13 (Version 1.7) Corrected a bug with spacing when the amsart is used.

2008-11-27 (Version 1.6) Corrected a bug in `getpiecelists` (empty lists were undefined).

2007-12-11 (Version 1.5) Added the key `getpiecelists` to retrieve the positions of pieces. Quite a lot new keys to show arbitrary positions from previously parsed games are added by the package xskak. Read its documentation if you want to use them.

2007-08-20 (Version 1.5) Added curvemove-style to draw moves. Corrected some bugs. Changed a lot internally to adapt the package to xskak.sty.

2007-07-03 (Version 1.4) Rewrote the code that process the keys saved globally (with `\chessboard`) and in styles (with `\storechessboardstyle`) to go around a problem due to a change in xkeyval 2.5. Corrected some errors in the documentation.

2006-07-20 (Version 1.3) Rearranged and rewrote the keys and commands for the pgf-pictures. Now all styles and marks can be used in both pictures. Added definitions for partial borders. Extended trim and clip commands. xifthen is now required.

2006-06-22 Added keys for fancy borders, added local commands for the boardarea and the printarea. Disabled the option skaknew as it not longer works in the package chessfss.

2 Introduction

What is does

`\chessboard` This package offers a command `\chessboard[`⟨*key=value list*⟩`]` to print boards of chess positions and similar games. It is intended to replace the `\showboard` command of the package skak which has some deficiencies:

- To print a special position one always has to type the complete FEN.
- It frames all boards with a rule of 1pt – which is okay for a large board but doesn't look good on small boards.
- It's difficult (up to impossible) to color some squares.
- Even for simple markings like crosses or the mover the package skak use postscript. That makes it difficult to use pdfLaTeX.
- It's difficult to print partial boards.
- It's difficult (up to impossible) to print boards with exotic pieces e.g. fairy chess.
- It's impossible to print boards with unusual dimension or unusual labels used by other games e.g. 10×10 boards with numbers instead of characters.

- The labels at the bottom changes the baseline which makes it difficult to align boards.
- Some commands e.g., `\notationoff` of the package skak redefines `\skakboard`, so it is difficult to patch the command e.g. to get larger margins.

With the package chessboard you have now full control about size, content and look of the board. I have tried to make `\chessboard` as flexible as possible. So I added a lot of options to change values that control the size, layout and filling of the board. In most cases you can ignore them but being able to change them can come handy if you need something unusual.

But I didn't tried to stretch flexibility too much. The main aim of the package is to print chessboards easily, so e.g. the inputs use the naming conventions of chess. That means that – as the files are numbered alphabetically – the width of the board is restricted to 26 fields (27 with option `zero`). The board is build with alternating black and white fields, changing to e.g. three colors could be difficult. The pieces are named with simple ASCII-chars, so the number of different pieces is restricted. (It is perhaps possible to define piecenames consisting of two or more chars, and to use such names by putting braces around the chars e.g., `{KK}a3`. A small test worked fine for me, but I wouldn't bet that it really works everywhere.)

What it doesn't

The package chessboard doesn't offer commands to type captions or titles or a list of the chessboards. There are no options that control the placement of the board in the text flow, e.g. to center it or to let it float. This is a design decision. `\chessboard` prints only chessboards like `\includegraphics` inserts only graphics.

2.1 Bugs and errors

I'm quite sure that they are bugs and errors in the package. I did find some quite horrible while making the documentation and I'm sure I overlooked some. E.g. I have a lot of doubts if spaces and braces in keys saved in a style or with `\setchessboard` are handled correctly in every case.

If you have questions ask them in the newsgroups `comp.text.tex` or `de.comp.text.tex`. I'm reading these groups regularly and I'm much better in answering there than in answering e-mails.

If you find errors in this text (this includes wrong english) or in the package, you can write me a bugreport at `skak@nililand.de`. A bugreport should contain a complete *minimal* example and the log-file of the pdfLaTeX run (this is the engine that makes a pdf!).

Before sending a bug record check the following things:

If you get mysterious error messages

- Check the commas in the *key=value lists*.

Missing or faulty pgf-decorations

- Check if the borders or marks are hidden under other decorations or under board chars with solid fieldmasks.
- Check if trimming is responsible: try the key `trim=false`.
- Check the state of the key `pgf`.
- Try another previewer, zoom in and out, print the document.
- Go through your log-file and search for chessboard warnings.

2.2 Requirements

The package chessboard uses some primitives of eTEX. It needs a recent version of the package chessfss (chess font selection), xkeyval (*key=value*-syntax) and xifthen. It also needs the packages pgfcore, pgfbaseshapes (from pgf for the highlighting commands), pst-node (from pstricks).

2.3 Installation

Run `chessboard.ins` through LaTeX and then put the four `.sty`-files somewhere where LaTeX finds them. `<texmf>/tex/latex/chessboard/` is a good place. Update the filename database.

2.4 Robustness: using `\chessboard` in moving arguments

I have used `\chessboard` in tabulars and in moving arguments. It seems to work fine, so I would say the command is quite robust.

Putting chessboards in the table of contents or the header will in most case not look very good, so it is sensible to avoid it with `\section[short text]{\chessboard Title}`. But if you want to do some advices:

- Boards in the title and in the table of contents are built at different times during the run, when perhaps different options and different games are active, so they can look different if you are not careful.
- You can't put boards in the bookmarks of a pdf, use `\texorpdfstring` from hyperref to provide a replacement.
- Some pagestyles puts a command to uppercase around the header. This affects keys in the optional argument of `\chessboard` and will give error messages about undefined keys. Try to surround the `\chessboard` command with `\lowercase{...}`.

2.5 Setting the options

The package chessboard use the keyval syntax ⟨*key*⟩=⟨*value*⟩.

It differs in some respects to other packages with keyval syntax:

- There are really *very* much keys.
- Some keys don't set properties of a board (like a width) but *do* things like drawing.
- The order of the keys can matter.
- Some effects are achieved by using two or more keys after one other.

There are keys that need a certain type of argument, e.g. a length or a list of fields. There are boolean keys that accept only the values `=true` and `=false` where the value `=true` can always be omitted. And at last there are keys that works as *commands*, you can use whatever argument you want or leave it blank. Don't confound this commands with boolean keys. E.g. `clearboard`, `clearboard=hallo` and even `clearboard=false` will always clear the board.

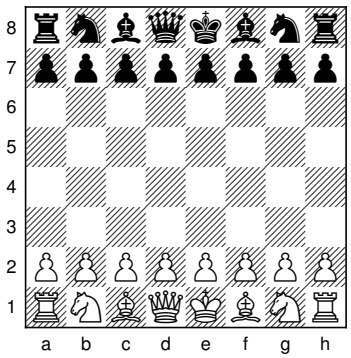
I'm struggling permanently with the naming of the keys. Giving them names that are not to long, but descriptive and a bit systematic isn't easy. Also it isn't easy to decide if a special effect should be achieve with one key (which gives a short input) or with a combination of keys (with is more flexible as it can be used to implement similar effects but gives longer input). While the package evolved I sometimes tumbled in problems and inconsistencies of the current naming scheme and had to reconsider my decisions and implemented better names or new key combinations. I didn't delete or disable older, now perhaps obsolate keys in any case, as I wanted to avoid that older documents breaks. I also sometimes add a key that is simply a copy of an existing key only to get a more consistent naming. So ist quite possible that you can get the same result with different key combination. For a lot of things there isn't one correct way – you can choose the one which suits you more.

It's easy to make mistakes when using a lot of keys. So if you get mysterious errors check that you don't have forgot a comma between the keys, that you put braces around lists used as value, that you didn't use a backslash before a key, that you didn't used a list of fields when a list of pieces is requested.

`\setchessboard` You can set an option either for one board by using the optional argument of `\chessboard` or by using `\setchessboard{`⟨*key=value list*⟩`}` which sets the keys for all following boards in the same environment/group.

Default setting in this Dokumentation:

```
Default setting in this Dokumentation:
\setchessboard{
    smallboard,
    showmover=false}
\newchessgame
\chessboard
```



Even if they share the same name, keys used in the argument of `\chessboard` are implementated differently than the keys used in `\setchessboard`. They do different things: In the first case the key gets "executed" while in the second case the *key=value* pair is only stored for later use. So it is quite possible that a key works with `\chessboard` but fails because of a bug when used in `\setchessboard` or vice-versa.

Values set by keys in the argument of `\chessboard` will if appropriate overwrite the ones set by `\setchessboard`.

The keys belong roughly to two classes: There are the keys that set one property of the board, e.g. the width of a rule or the size. Such keys always have a default and only the last value set is used, it overwrites all previous values. Internally I call such keys "set-keys". In the second class there are keys that can be used more than once, e.g. keys that add a piece or a mark or a color to the board. I call such keys "fill-keys".[1]

The "set-keys" are always processed first. Inside the classes the keys are processed in general from "left to right", so later keys can overwrite values set by earlier keys. This is also true for values hidden in a `style`.

For each "set-key" you will find in this documentation a box like this:

`zero=`⟨*true*|*false*⟩	`zero=false, zero`	`false`

It shows on the left the general description of the possible values, in the middle one or more examples of correct input, and on the right the default value.

For each "fill-key" you will find in this documentation a box like this:

`clearranks=`⟨*list of ranks*⟩	`clearranks=8`

It shows on the left the general description of the possible values and on the right one or more examples of correct input.

[1] I did realize only quite late that using keys not only to set some parameters but also for the real content is a bit unusual. It is a bit as if one would use keys for the items of a list or the lines of tabular or the code in a listing – in most cases quite a bad idea. But at my opinion it works fine in this special case because a chessboard is a quite restricted room with only 64 fields.

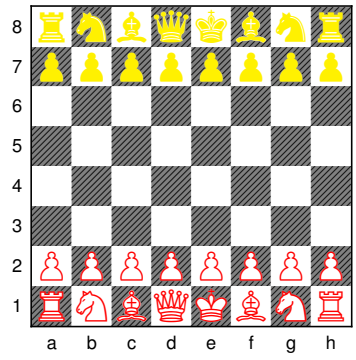
2.6 Saving optionlists

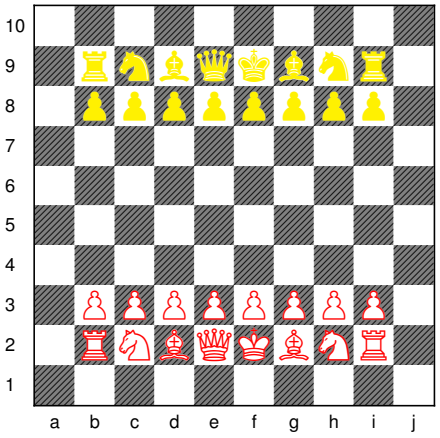
style=⟨*name*⟩	style=mystyle

chessboardstyle

With the \storechessboardstyle{⟨*name*⟩}{⟨*key=value list*⟩} it is possible to store a list of *key=value*-pairs in a "style" and to use this style instead of the keys themselves.

```
\storechessboardstyle{10x10}{%
 maxfield=j10,
 clearboard,
 startfen=b9,
 restorefen=current,
 labelleftwidth=1.5ex}

\storechessboardstyle{red-yellow}{%
 boardfontencoding=LSBC3,
  whitepiececolor=red,
  blackpiececolor=yellow,
  setfontcolors}

\setchessboard{style=red-yellow}
\newchessgame
\chessboard

\chessboard[style=10x10]
```

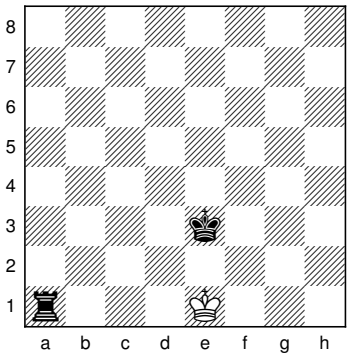




Attention

\storechessboardstyle and \setchessboard only store the *key=value* pair, they don't expand anything and they don't do anything with them. So be careful when using commands that could be redefined later:

```
\def\mylist{Ke1, qe2, kf3}
\setchessboard{setpieces=\mylist}
\def\mylist{Ke1, ra1, ke3}
\chessboard
```



2.7 Naming the board

In chess the rows of the board are called *ranks* and are numbered from 1 to 8. The columns are called *files* and are "numbered" from a to h.[2]

The package chessboard use this naming conventions for all inputs: Every time ⟨*field*⟩ is used in this documentation, you should give a char from a–z followed by a number.

2.8 Naming areas of the board

Rectangular areas are normally described by giving the coordinates of the left bottom corner and the right upper corner. But when using the Forsyth Edwards Notation (FEN) chessboards are filled starting from the left upper corner a8 and then going on to the right and down to the right bottom corner – like the normal typesetting direction. So it is quite natural to use for FEN related areas this corners. As it would be awkward to have two different ways to describe areas, I decided to use the fen convention everywhere: Each area has a *startfield*, the left upper corner, and a *stopfield*, the right bottom corner. If some area related keys don't seem to work, check that you have used the correct corners!

An area can also be given as two fields separated by an hyphen. When using this input you don't have to worry about the corners, the package will sort them:
⟨*area*⟩=⟨*a corner*⟩-⟨*the opposite corner*⟩.

`\printarea` `\board` `\chessboard` sets at the start this two command to the current values of the printarea and the total board as given by the keys `zero`, `maxfield` and `printarea`. You can use this commands in later keys.

[2] I have some difficulties to remember the english names, so I use the following mnemonics: **F**i**L**e = co**L**umn = **L**inie = al**F**abet, **R**ank = **R**ow = **R**eihe = numbe**R**ed

2.9 FEN: Forsyth-Edwards Notation

FEN describes a chess position. It consist of 6 fields, separated by spaces. The first field represents the placement of the pieces on the board. The second field represents the active color. A lower case “w” is used if White is to move; a lower case “b” is used if Black is the active player. The third field represents castling availability. The fourth field is the en passant target square. The fifth field is a nonnegative integer representing the halfmove clock. The sixth and last field is a positive integer that gives the fullmove number.

In the first field the board contents are specified starting with the eighth rank and ending with the first rank (“in typesetting direction” when the board has the standard orientation). For each rank, the squares are specified from file a to file h. White pieces are identified by uppercase piece letters (“PNBRQK”) and black pieces are identified by lowercase piece letters (“pnbrqk”). Empty squares are represented by the digits one through eight; the digit used represents the count of contiguous empty squares along a rank. A solidus character “/” is used to separate data of adjacent ranks.

\chessboard handles FEN-input but also only “FEN-like”-input. If given a complete FEN \chessboard will store all fields in commands and evaluate the first field to fill the board. But if they are less than six fields, \chessboard will not complain but simply fill up the missing fields with some default values.

In the first field there don’t need to be exact 8 pieces in each rank and exactly 8 rank descriptions. When there are more pieces than needed, \chessboard will ignore them. When there are less pieces than needed then \chessboard will leave the remaining fields in this rank untouched.

The FEN can also be stored in a command. So all \chessboard will accept all the following inputs as ⟨*FEN*⟩:

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR
8qK
10 w - - 0 10
\myfen
```

2.10 The main parts of the board

Each chessboard is build by printing three “layers” on top of each other:

1. The background layer

This is a pgf-picture that lies under the board. It can be used for background colors or fancy borders or other background decoration.

2. The main board layer

This is the main part of the board. It lies above the background picture. It uses only “normal” LaTeX objects like rules and chars. Labels, movers and figures are all build together in this layer.

3. The mark layer

This layer is again a pgf picture now over the board. It can be used to draw crosses, arrows etc on the board.

The sections 3 to 5 describe how to manipulate the main layer. The section 6 describes how to handle the pgf pictures.

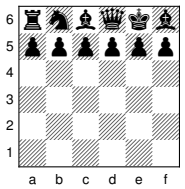
3 Setting the contents of the board

3.1 The maximum number of fields

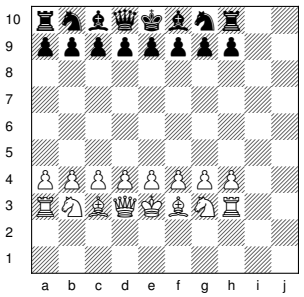
`maxfield=`⟨*field*⟩ `maxfield=j10` h8

Chessboards have 8×8 fields. So this is the default. If you want another size you should declare a new right upper corner with `maxfield=`⟨*Field*⟩ e.g., `maxfield=j10` for a 10×10 board. Don't use this key if you want only to print a partial board, there is another key to achieve this. Use it to change the maximum size of a board. All fields and areas you use in other arguments should lie inside this maximum size.

```
\newchessgame
\chessboard[tinyboard,
         maxfield=f6]
```



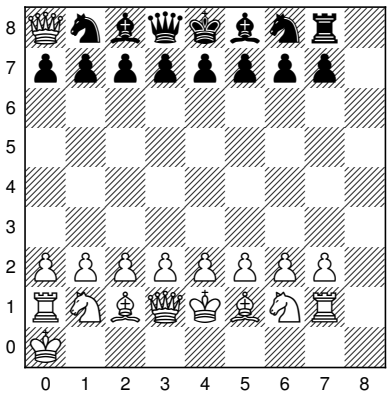

```
\newchessgame
\chessboard[tinyboard,
         maxfield=j10,
         labelleftwidth=2ex]
```



zero=⟨*true*|*false*⟩	zero=false	false

I was told that some games numbers the fields starting with 0. So I added this option. When set to true the rank numbers start with 0 and a file named Z is added left to the file a.

```
\newchessgame
\chessboard[zero,
         labelbottomformat=\arabic{filelabel},
         addpieces={KZ0, QZ8}]
```

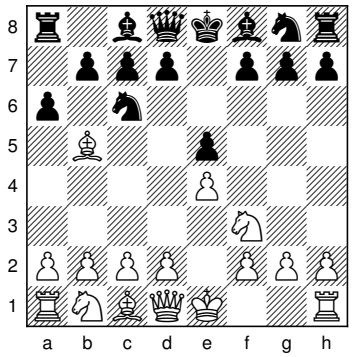


3.2 Filling with the package skak

When the package skak is loaded and there is a running game, \chessboard will fill the board with the current position.

1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6

```
\newchessgame
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5 a6}
\chessboard
```



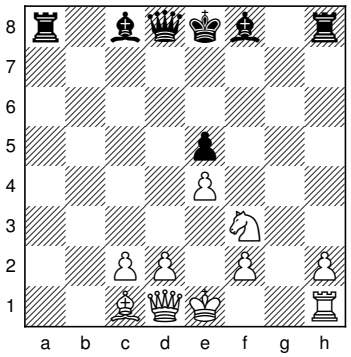
3.3 Clearing

clearboard=⟨*arbitrary*⟩	clearboard
cleararea=⟨*area*⟩	cleararea=a1-c3
clearareas=⟨*list of areas*⟩	clearareas={a1-c3,g7-h8}
clearfile=⟨*file*⟩	clearfile=g
clearfiles=⟨*list of files*⟩	clearfiles={g,h}
clearrank=⟨*rank*⟩	clearrank=5
clearranks=⟨*list of ranks*⟩	clearranks={8,7}
clearfield=⟨*field*⟩	clearfield=c6
clearfields=⟨*list of fields*⟩	clearfields={b5,c6}

You can't prevent that \chessboard fills up the board with the running game, so if you need an empty board, you should clear it – either with the keys mentioned here, or by using one of the keys described later that clears the board before adding new pieces.

1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6

```
\newchessgame
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5 a6}
\def\ranklist{7 ,6 }
\def\cleararea{a1-b2}
\chessboard[clearfiles=g,
         clearranks=\ranklist,
         cleararea=\cleararea,
         clearfields={b5,c6}]
```



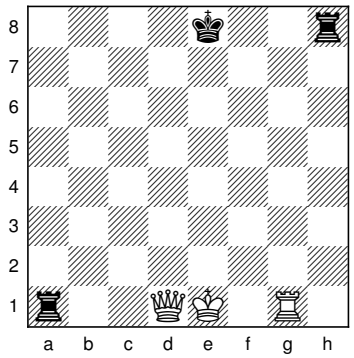
3.4 Adding single pieces

setpieces=⟨*List of piece positions*⟩	setpieces={Ke4, Qe1, kd6}
addpieces=⟨*List of piece positions*⟩	addpieces={Ke4, Qe1, kd6}

The setpieces first clears the board and then set the pieces, the addpieces only set the new pieces but lets the existing pieces undisturbed.

1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6

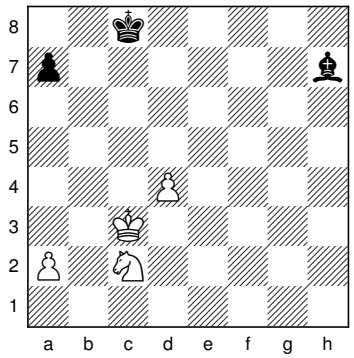
```
\newchessgame
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5 a6}
\def\piecelist{Ke1,Qd1 , Rg1}
\chessboard[setpieces={ra1, rh8, ke8},
         addpieces=\piecelist]
```



setwhite=⟨*List of piece positions*⟩	`setwhite={Ke4, Qe1, kd6}`
addwhite=⟨*List of piece positions*⟩	`addwhite={Ke4, Qe1, kd6}`
setblack=⟨*List of piece positions*⟩	`setblack={Ke4, Qe1, kd6}`
addblack=⟨*List of piece positions*⟩	`addblack={Ke4, Qe1, kd6}`

It is a bit cumbersome to have to be careful to use uppercase chars for the white and lowercase chars for the black piece. With the keys `setwhite`, `addwhite`, `setblack` and `addblack` you can add white and black pieces without caring about the cases. Like the keys for adding pieces, the keys starting with `set` first clears the board, so if you don't want to loose all the set pieces, you should use such a key only once.

```
\def\whitepieces{kc3, nc2, pa2, Pd4}
\chessboard[setwhite=\whitepieces,
         addblack={Kc8,bh7, pa7}]
```

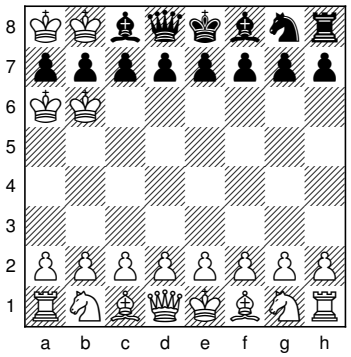


3.5 Adding FEN-positions

setfen=⟨*FEN*⟩	`setfen=\myfen, rnbqk/8/RNBQK`
addfen=⟨*FEN*⟩	`addfen=\myfen, rnbqk/8/RNBQK`

With this keys you can add pieces to the board by given a FEN. The `setfen` will clear the board, `addfen` will left pieces outside the fields described by the FEN undisturbed.

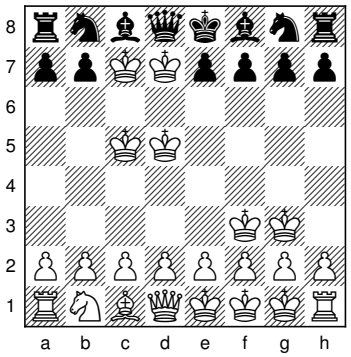
```
\newchessgame
\def\myfen{KK//KK}
\chessboard[addfen=\myfen]
```



`startfen=`⟨*field*⟩	`startfen=b7`

As a default `\chessboard` fills the board with a FEN-position starting at the left upper corner. You can shift this start field with the key `startfen`.

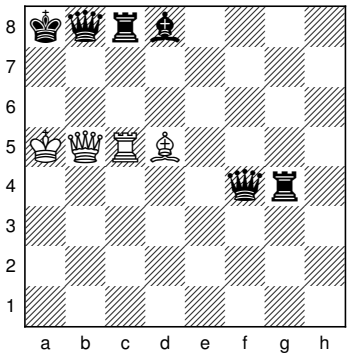
```
\newchessgame
\def\myfen{KK//KK}
\chessboard[startfen=c7, addfen=KK//KK,
         startfen=f3, addfen=\myfen]
```



`startfill=`⟨*field*⟩	`startfill=b7`
`stopfill=`⟨*field*⟩	`stopfill=f2`
`fillarea=`⟨*area*⟩	`fillarea=c2-d4`

With this keys you can restrict the part of the board that is filled by a FEN. Only the pieces inside the fillarea are set.

```
\newchessgame
\def\myfen{kqrb///KQRB}
\chessboard[setfen=\myfen, startfen=e4, fillarea=f2-g4,
  addfen=\myfen]
```



3.6 Saving positions

The package skak has the command `\storegame{`⟨*name*⟩`}` that stores the FEN of the current game in an internal command, and the command `\savegame{`⟨*name*⟩`}` that stores the FEN in a file `{`⟨*name*⟩`}.fen`

The package chessboard defines similar keys. The key `storefen=`⟨*name*⟩ stores the FEN in an internal command *with the same name* as used by `\storegame{`⟨*name*⟩`}`. That means that you can exchange the games stored by the package skak and the package chessboard, but it also means that the package chessboard can *overwrite* games stored with the package skak. So be careful!

`storefen=`⟨*name*⟩	`storefen=game1`
`savefen=`⟨*name of file*⟩	`savefen=game`

`storefen` saves the current position as FEN to a internal command. `savefen` saves the current position as FEN to a file.

`startstore=`⟨*field*⟩	`startstore=b7`
`stopstore=`⟨*field*⟩	`stopstore=f2`
`storearea=`⟨*area*⟩	`storearea=c2-d4`

With this keys you can restrict the area that is stored or saved. But attention: the package skak absolutly doesn't like "illegal" FENs that doesn't describe a full 8×8 board.

mover=⟨*w*|*b*⟩	mover=w
castling=⟨*castling possibilies*⟩	castling=Kq
enpassant=⟨*field*|-⟩	enpassant=c4
halfmove=⟨*number*⟩	halfmove=14
fullmove=⟨*number*⟩	fullmove=20

With this keys you can set the rest of the FEN-fields. Apart from mover the values are only used if you save FENs. The values are also set, if you use in the argument of a setfen or addfen key complete FENs instead of only the position part.

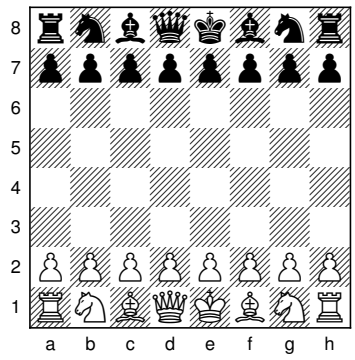
3.7 Getting the positions of pieces

getpiecelists=⟨⟩	getpiecelists

New in version 1.5 there is a key, that retrieves the position of the pieces on the board. For each piece type there is a list called \cblist⟨*piece*⟩.. The lists can be used everywhere where a list of fields is expected. E.g. it is possible to use the lists to highlight all pawns (see section 6.1.3).

Unlike almost all other commands of chessboard this commands are saved globally. This makes it possible to use the lists after the board:

```
\newchessgame
\chessboard[getpiecelists]

White pieces: \king\cblistK, \queen\cblistQ,
          \rook\cblistR, \bishop\cblistB,
          \knight\cblistN, \pawn\cblistP.

Black pieces: \king\cblistk, \queen\cblistq,
          \rook\cblistr, \bishop\cblistb,
          \knight\cblistn, \pawn\cblistp.
```



White pieces: ♔e1, ♕d1, ♖a1,h1, ♗c1,f1, ♘b1,g1, ♙a2,b2,c2,d2,e2,f2,g2,h2.

Black pieces: ♔e8, ♕d8, ♖a8,h8, ♗c8,f8, ♘b8,g8, ♙a7,b7,c7,d7,e7,f7,g7,h7.

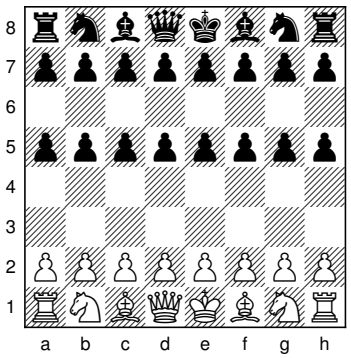
3.8 Using saved and stored games

When you are using the package skak, you can use the saved games to set the start position of a game with the the package skak-commands `\restoregame{`⟨*name*⟩`}` (restores the game stored with `\storegame`) and `\loadgame{`⟨*name*⟩`}` (loads the game saved with `\savegame`). This only works if you have saved the position of a normal 8×8-board!

`restorefen=`⟨*name*⟩	`restorefen=game1`
`loadfen=`⟨*name*⟩	`loadfen=game`

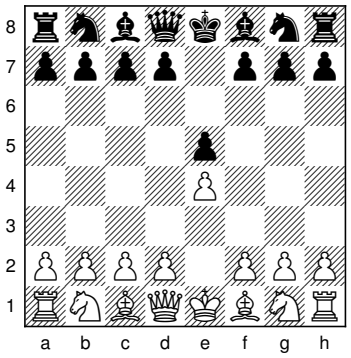
You can also use the saved games to set positions on `\chessboard`. `restorefen` will use games saved with `\storegame` and `storefen`, `loadfen` will load games saved with `\savegame` and `savefen`.

```
\newchessgame
\def\gamename{rank7}
\chessboard[storearea=a7-h7, storefen=\gamename,
         startfen=a5, restorefen=rank7]
```




```
\mainline{1. e4 e5}
\savegame{file} % to file.fen
\mainline{2. Nf3 Nf6}
\def\filename{file}
\chessboard[loadfen=\filename]
```

1 e4 e5 2 ♘f3 ♘f6

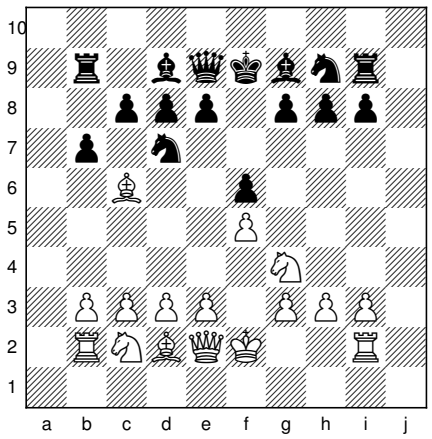


3.9 Restoring the running game

`\chessboard` saves the running game at the start under the name “current”.

1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6

```
\newchessgame
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5 a6}

\chessboard[maxfield=j10,
         clearboard,
         startfen=b9,
         restorefen=current]
```



3.10 Changing the input language

`language=`⟨*name*⟩ `language=german`

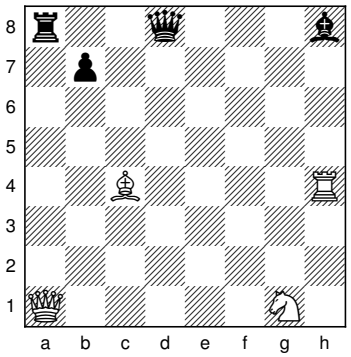
bDefineLanguage
fineTranslation

With this key you can change the language used for input of pieces. The key doesn't affect the input language of the package skak. It also doesn't affect the language of saved and restored games. `\chessboard` will always switch to english in this cases! Up to now only german translations are built in (I don't know how chess pieces are called in other languages), but it isn't difficult to add other languages. The code for german should be quite selfexplanatory:

```
\cbDefineLanguage{german}%
\cbDefineTranslation{german}{K}{K}%
\cbDefineTranslation{german}{Q}{D}%
\cbDefineTranslation{german}{R}{T}%
\cbDefineTranslation{german}{B}{L}%
\cbDefineTranslation{german}{N}{S}%
\cbDefineTranslation{german}{P}{B}%
\cbDefineTranslation{german}{k}{k}%
\cbDefineTranslation{german}{q}{d}%
\cbDefineTranslation{german}{r}{t}%
\cbDefineTranslation{german}{b}{l}%
\cbDefineTranslation{german}{n}{s}%
\cbDefineTranslation{german}{p}{b}%
```

You can mix the languages in the input (but I don't think you should do it, it can get quite confusing, it's better to stick to one language).

```
\def\whitepieces{lc4, sg1}
\chessboard[setpieces={qd8, ra8, bh8},
         language=german,
         addpieces={Da1, Th4},
         addwhite=\whitepieces,
         addblack={Bb7}]
```



4 The look of the board

4.1 Units for lengths

For many of the following commands you will have to enter a length. This can be an absolute length like 1in, 2cm, 4pt. But often you will prefer a relative length. In most cases the font related length 1ex, 1em and `\baselineskip` will be the same as a half of the square width, the width of the square and the total height of a square.[3] The package uses internally the three lengths offered by the package chessfss – `\len@cfss@squarewidth`, `\len@cfss@squaretotalheight` and `\len@cfss@squaredepth` –. In most cases they will only give inside `\chessboard` (locally) the correct sizes, but if you use the key `psset` they are set globally.

4.2 Some words about box sizes

In TEX almost everything is either a box (a character box, a line box, a page box, a box with a tabular in it ...) or a space. And larger objects are build by putting boxes and spaces in a surrounding box. In the real world a box has always to be larger as its content. In the virtual TEX-world a lot of magic is possible: The content can be much larger than the surrounding box or even be partly or completely outside the surrounding box.

```
x% shows the current line
\fbox{% a box
\raisebox{3ex}[0pt][0pt]{%
   \makebox[0pt]{%
   large content outside the box}}%
x% and here something after the fbox
```

large content outside the box

x□x

[3] I learned on the hard way that you shouldn't rely that fonts sets em and ex correctly, e.g. the font skak doesn't do it and so the units default to zero which can be quite confusing. So I decided to set the fontdimen values explicitly.

So the outside size of a box can be quite different from the inside size. And the outside baseline of a box can be at quite different place as the inside baseline.

In the concrete case of `\chessboard` the outside size of the board is determined through the size of the font, the fields shown and the margin. All other things like border, label, highlighting, mover will not affect this size. The baseline of a chess board is always at the bottom of the bottom rank. Margins, labels or borders or anything else will not change this. If you want to move the baseline you will have to use a `\raisebox`.

4.3 Margins

The "outside size" ("the bounding box") of a chess board is the size of the printed fields plus a margin.

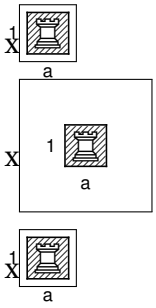
`marginleftwidth=`⟨*length*⟩	`marginleftwidth=1em`	`1em`
`marginrightwidth=`⟨*length*⟩	`marginrightwidth=1em`	`1em`
`margintopwidth=`⟨*length*⟩	`margintopwidth=1em`	`1em`
`marginbottomwidth=`⟨*length*⟩	`marginbottomwidth=1em`	`1em`
`hmarginwidth=`⟨*length*⟩	`hmarginwidth=1em`	`1em`
`vmarginwidth=`⟨*length*⟩	`vmarginwidth=1em`	`1em`
`marginwidth=`⟨*length*⟩	`marginwidth=1em`	`1em`

With this keys you can set and change the thickness of the margin. As you can see you can set each margin differently, but there are also keys to set more than one margin at one time.

`marginleft=`⟨*true*|*false*⟩	`marginleft`	`true`
`marginright=`⟨*true*|*false*⟩	`marginright=false`	`true`
`margintop=`⟨*true*|*false*⟩	`margintop`	`true`
`marginbottom=`⟨*true*|*false*⟩	`marginbottom`	`true`
`hmargin=`⟨*true*|*false*⟩	`hmargin`	`true`
`vmargin=`⟨*true*|*false*⟩	`vmargin`	`true`
`margin=`⟨*true*|*false*⟩	`margin`	`true`

You can disable the margin either by setting its width to 0pt or with this boolean keys.

```
x\fbox{\chessboard[printarea=a1-a1,
               marginwidth=0pt]}

x\fbox{\chessboard[printarea=a1-a1,
              marginwidth=0.5cm]}

x\fbox{\chessboard[printarea=a1-a1,
              marginwidth=0.5cm,
              margin=false]}
```



4.4 Borders

The borders are made with rules. You will perhaps notice that on screen the edges looks like as if they don't contact properly. This is a problem of the rather bad resolution of a screen. When you zoom in you can see that the edges are fine. If this bothers you, you can try the fancy borders made with the pgf-commands decribed later. As they are made with one line, they seem to have better edges – but you can't color them individually and it isn't yet possible to disable the borders of single sides.

borderleftwidth=⟨*length*⟩	borderleftwidth=1em	0.04em
borderrightwidth=⟨*length*⟩	borderrightwidth=1em	0.04em
bordertopwidth=⟨*length*⟩	bordertopwidth=1em	0.04em
borderbottomwidth=⟨*length*⟩	borderbottomwidth=1em	0.04em
hborderwidth=⟨*length*⟩	hborderwidth=1em	0.04em
vborderwidth=⟨*length*⟩	vborderwidth=1em	0.04em
borderwidth=⟨*length*⟩	borderwidth=1em	0.04em

With this keys you can set and change the thickness of the border. The border doesn't change the bounding box of the board. If there isn't an adequate margin it will stick outside!

borderleft=⟨*true*|*false*⟩	borderleft	true
borderright=⟨*true*|*false*⟩	borderright=false	true
bordertop=⟨*true*|*false*⟩	bordertop	true
borderbottom=⟨*true*|*false*⟩	borderbottom	true
hborder=⟨*true*|*false*⟩	hborder	true
vborder=⟨*true*|*false*⟩	vborder	true
border=⟨*true*|*false*⟩	border	true

You can disable the border either by setting its width to 0pt or with this boolean keys.

borderleftcolor=⟨*color*⟩	borderleftcolor=red	black
borderrightcolor=⟨*color*⟩	borderrightcolor=red	black
bordertopcolor=⟨*color*⟩	bordertopcolor=red	black
borderbottomcolor=⟨*color*⟩	borderbottomcolor=red	black
hbordercolor=⟨*color*⟩	hbordercolor=red	black
vbordercolor=⟨*color*⟩	vbordercolor=red	black
bordercolor=⟨*color*⟩	bordercolor=red	black

With this keys you can set and change the color of the border.

You must load a color package like xcolor or color to be able to use colors![4]

```
\raggedright
some text above the board\\
the board will overwrite this!\\
left%
\def\borderwidth{12pt}
\def\mycolor{red!50}
\chessboard[tinyboard,
         marginleftwidth=1em,
         margintopwidth=0pt,
         bordertopwidth=\borderwidth,
         borderleftcolor=\mycolor,
         bordertopcolor=yellow!25,
         marginright=false]%
right\\
text below the board
```

some text above the board
the bo e this!
left right
text below the board

[4] I must correct myself: the pgf-package will load xcolor, so you should load a color package only if you want to use it with other options

4.5 The size of the boardfont

boardfontsize=⟨*length*⟩	boardfontsize=10pt	20pt
fontsize=⟨*length*⟩	fontsize=10pt	20pt
tinyboard=⟨*arbitrary*⟩	tinyboard	
smallboard=⟨*arbitrary*⟩	smallboard	
normalboard=⟨*arbitrary*⟩	normalboard	default
largeboard=⟨*arbitrary*⟩	largeboard	

With this keys you can change the size of the board font. The keys `tinyboard` etc. gives the same sizes as the similar commands `\tinyboard` of the package skak, they also change the size of the font of the labels.

You can also change the size of the board font with the commands described in the documentation of the package chessfss. But you should be aware that the keys used in `\setchessboard` or `\chessboard` will always win over the commands from the package chessfss.

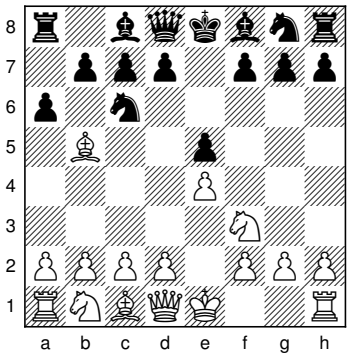
4.6 Changing the boardfont

Instead of using the keys described below you can also change the board font with the commands described in the documentation of the package chessfss. But you should be aware that the keys used in `\setchessboard` or `\chessboard` will always win over the commands from the package chessfss.

boardfontfamily=⟨*family name*⟩	boardfontfamily=maya	skak/skaknew

With the above key you can change the board font by setting the family. This works quite similar as for text font where you can switch e.g. from times to helvetica by changing the font family. The font families skak and skaknew will work from the start. You must install other families before being able to use them!

```
\chessboard[boardfontfamily=maya]
```

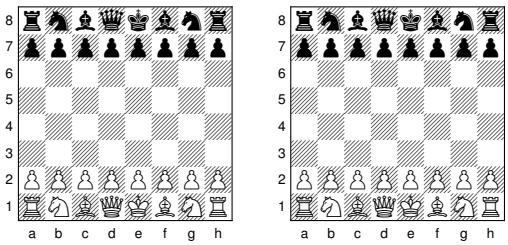


`boardfontseries=`⟨*series name*⟩ `boardfontseries=b`

Setting the series to bold (b) only works for a quite small set of fonts which have the chars in two weights. But to tell the truth: I don't find the result very satisfying.

There are no keys to change the shape of the font as I really don't know what an italic (`\itshape`) or slanted board font should be. (And I never saw such a font).

```
\newchessgame
\chessboard[boardfontfamily=millennia,
         tinyboard]
\chessboard[boardfontfamily=millennia,
         boardfontseries=b,
         tinyboard]
```



`boardfontencoding=`⟨*encoding*⟩ `boardfontencoding=LSB1` LSB

With this key you can change the encoding of the board font. While in a normal document you probably seldom or never need to bother with encodings they are highly useful in connection chess board if you want to add colors to the chars on the board. When using other encodings that the standard LSB the chars on the board are composed with special chars that allows to color e.g. the lines of the black field differently to the figure on the field. I will come back to this when I describe how to color the chars.

There are simple encodings called LSB, LSB1, LSB2 ..., which works with almost every font family. And there are special encodings LSBC1, LSBC2 ..., which works only with professional chess fonts with special chars. Happily the default board font, the free font skaknew, is such a professional chess font.

If you want to use one of the extended encodings LSB1, LSB2, etc and LSBC1, LSBC2 etc you must load the definition files e.g. with `\usepackage[LSB1,T1]{fontenc}`. You find more informations in the documentation of the package chessfss.

4.7 Coloring and emphasizing the board chars

4.7.1 In short

- Coloring the board or parts of the chars is a two step process:

  First you set the colors you want, then you apply them.
- Colors can be applied to the whole board (to set the default colors of the white and black pieces and fields). This done with the keys `setfontcolors` and `addfontcolors`.
- Colors can be applied to part of the board together with other commands like `\bfseries` (e.g. to emphasize special fields or ranks) by first enabling color emphasing with the key `coloremph` and then by setting the area or field that should get the colors with e.g. the key `empharea`.

4.7.2 Introduction: About composed chars and encodings

In simple chess fonts each field with and without a figure on it is *one* char. You can color such a char but – as in the case of normal chars – only in one color:

This is certainly not very satisfying. You probably would like to be able to color the white figures diffently from the black ones, and the black fields in a third color.

To be able to do this, one has to use more than one char to build a field and to print them on top of each other. Look e.g. at this example:

It is build by putting four different objects on top of each other:

At start there is a standard yellow colorbox

Above there is a black empty square colored in blue:

Then comes a solid, filled and colored char with the same shape as the king:

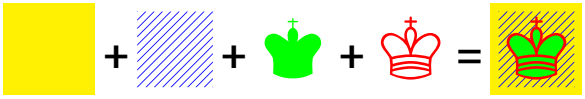
Table 1: The contruction rules of encoding LSBC3

Encoding LSBC3

Layer:	fieldmask	field	piecemask	piece	result
WhiteSquare					
BlackSquare					
WhiteOnWhite					
WhiteOnBlack					
BlackOnWhite					
BlackOnBlack					

At last there is the char for the king:

So



There are a lot of other possibilities to compose the board fields. Some more “poor man’s” compositions that use only chars present in every chess font, other that need special masking chars which only special chess fonts provide.

Each such construction is connected to a font encoding: By changing the encoding of the board font you change the rules that compose the board chars and so you change also the options to color the chars.

In the package chessfss I have defined a bundle of such “construction rules” (=encodings) to use and color composed board chars. As it would be a strain if each of this construction rules would let to a different set of commands to color the chars I had to systemize the construction rules. For this I have divided the possible individual components of a composed chars in four logical layers and assign to each layer generic color commands that are used in the definition of the composed chars and can be used to color the components. The four layers are named fieldmask, field, piecemask and piece.

As an example table 1 shows a tabular that demonstrates how the encoding LSBC3 is built. As you can see the chars are built by putting up three or four chars on top of each other. (The green and yellow colors are not the default settings!). The documentation of the package chessfss shows the other encodings.

Table 2: The internal color commands

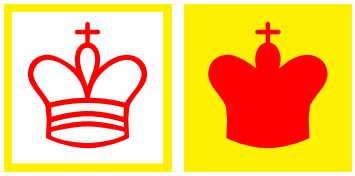
layer	*internal color commands*	*default definition*
fieldmask	\cfss@whitefieldmaskcolor	\color{white}
	\cfss@blackfieldmaskcolor	\color{gray}
field	\cfss@whitefieldcolor	{}
	\cfss@blackfieldcolor	{}
piecemask	\cfss@whiteonwhitepiecemaskcolor	\color{white}
	\cfss@whiteonblackpiecemaskcolor	\color{white}
	\cfss@blackonwhitepiecemaskcolor	\color{white}
	\cfss@blackonblackpiecemaskcolor	\color{white}
piece	\cfss@whitepiececolor	{}
	\cfss@blackpiececolor	{}

Table 2 shows the internal commands used to color the different part of a composed char. You will probably never need them but they will give you an idea what can be colored (I hope that is clear that it makes only sense to change the colors of layers that are used in the actual encoding – you can't color an solid fieldmask that isn't there).

I would like to draw your attention to two things:

1. The field-layers distinguish between black and white field while the piece layer distinguish between black and white pieces and the piecemask offers all possible combinations.
2. The fieldmask and the piecemask are colored by default and so they will overwrite or block out external colors from \colorbox or \textcolor commands:

```
%Uses commands from chessfss.sty
\setboardfontencoding{LSBC3}%
\setboardfontsize{2cm}

\colorbox{yellow}{%
 \textcolor{red}{\WhiteKingOnWhite}}
%
\setboardfontcolors{
 whitefieldmask=yellow,
 whiteonwhitepiecemask=red}
\colorbox{yellow}{%
 \textcolor{red}{\WhiteKingOnWhite}}
```



So coloring the board is not really difficult: one must only redefine the internal commands mentioned above to the wanted color. That's what the \setboardfontcolors command of the package chessfss does and it will work fine with chessboard.

Naturally I also wanted to add keys for \chessboard and \setchessboard to change the colors. This is a bit more complicated because I wanted not only to be able to change the

colors of the whole board but also of single fields or areas. And I had also to add keys to change the color in the two pgf pictures. Finding names for all this keys and effects wasn't easy. And I hope that it isn't to confusing.

4.7.3 Setting the default colors of the board

`clearfontcolors=`⟨*arbitrary*⟩	`clearfontcolors`

When you set with the following keys a color for one of the font layers, `\chessboard` adds simply the color definition to an internal command (the "font color stack"). When you use later a key to apply the colors to the board or an area this internal command is inserted at the start of the board or the fields of the area. The internal command can get quite long if you set a lot of colors. With the key `clearfontcolors` you can empty the internal command (this will not destroy the colors that you have already applied).

`whitefieldmaskcolor=`⟨*color*⟩	`whitefieldmaskcolor=red!50`
`blackfieldmaskcolor=`⟨*color*⟩	`blackfieldmaskcolor=red!50`
`fieldmaskcolor=`⟨*color*⟩	`fieldmaskcolor=red!50`
`whitefieldcolor=`⟨*color*⟩	`whitefieldcolor=red!50`
`blackfieldcolor=`⟨*color*⟩	`blackfieldcolor=red!50`
`fieldcolor=`⟨*color*⟩	`fieldcolor=red!50`
`whiteonwhitepiecemaskcolor=`⟨*color*⟩	`whiteonwhitepiecemaskcolor=red!50`
`whiteonblackpiecemaskcolor=`⟨*color*⟩	`whiteonblackpiecemaskcolor=red!50`
`blackonwhitepiecemaskcolor=`⟨*color*⟩	`blackonwhitepiecemaskcolor=red!50`
`blackonblackpiecemaskcolor=`⟨*color*⟩	`blackonblackpiecemaskcolor=red!50`
`whitepiecemaskcolor=`⟨*color*⟩	`whitepiecemaskcolor=red!50`
`blackpiecemaskcolor=`⟨*color*⟩	`blackpiecemaskcolor=red!50`
`onwhitepiecemaskcolor=`⟨*color*⟩	`onwhitepiecemaskcolor=red!50`
`onblackpiecemaskcolor=`⟨*color*⟩	`onblackpiecemaskcolor=red!50`
`piecemaskcolor=`⟨*color*⟩	`piecemaskcolor=red!50`

whitepiececolor=⟨*color*⟩	whitepiececolor=red!50
blackpiececolor=⟨*color*⟩	blackpiececolor=red!50
piececolor=⟨*color*⟩	piececolor=red!50

With this keys you set the colors of the different parts of a composed char. The colors are saved to an internal stack. A key like whitepiececolor saves the color for one of the layer color commands. piecemaskcolor will set the color for all four piecemasks in one go.

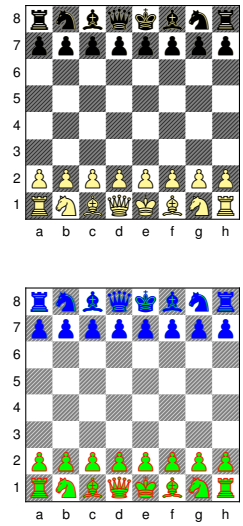
The keys colorwhite and colorblack of the previous version of this package are copies of the keys whitepiececolor and blackpiececolor.

4.7.4 Applying the colors to the whole board

setfontcolors=⟨*arbitrary*⟩	setfontcolors
addfontcolors=⟨*arbitrary*⟩	addfontcolors

This keys will put the "font color stack" in a command that is executed at the start of the board. So the colors will affect any fields that don't get individual colors with the command described in the next subsection. setfontcolors will replace any font colors that have been set before, addfontcolors will add the new colors to perhaps already existing ones.

```
\def\mycolor{yellow!50}
\setchessboard{
 boardfontencoding=LSBC3,
 piecemaskcolor=\mycolor,
 setfontcolors}

\chessboard[tinyboard]

\def\mycolor{green}% will set the
% piecemask to green:
\chessboard[tinyboard,
          whitepiececolor=red,
          blackpiececolor=blue,
          fieldcolor=white,
          addfontcolors]
```

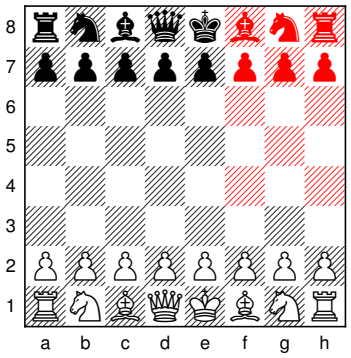


4.7.5 Emphasising and coloring individual areas

emphstyle=⟨*commands*⟩	emphstyle=\bfseries

With this key you can set a command that should be applied to an area. The commands can be whatever you want, (it is a good idea not use something that takes up some space). In my opinion the only senseful commands are `\color` and – for the few fonts that have a bold boardfont – `\bfseries`. You can use the key e.g. for simple coloring.

```
\newchessgame
\def\empharea{ h8-f4 }
\chessboard[emphstyle=\color{red},
         empharea=\empharea]
```

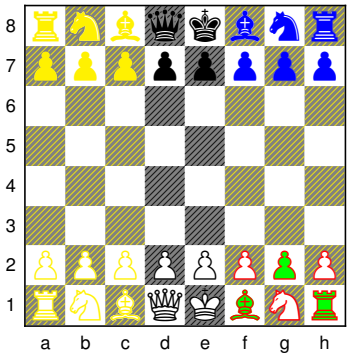


You can't color with `emphstyle` the fieldmask and the piecemask of chars as they have a default color that takes precedence.

coloremph=⟨*true*|*false*⟩	coloremph

With this key you enable or disable color emphasising. When set to true, font colors set earlier will be added to emphstyle and so used to color the area. The colors are added after the commands of the key `emphstyle` and so will eventually overwrite them. This allows more complicated coloring:

```
\newchessgame
\def\empharea{ h8-f1 }
\chessboard[boardfontencoding=LSBC3,
         whiteonwhitepiecemaskcolor=green,
         whitepiececolor=red,
         blackpiececolor=blue,
         emphstyle=\color{yellow},
         %doesn't affect the default white color
         %of the piecemask
         empharea=a1-c8,
         coloremph,
         empharea=\empharea]
```



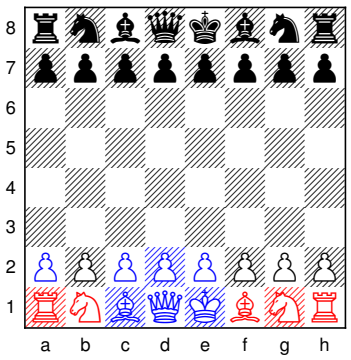
coloremphstyle=⟨*commands*⟩	coloremphstyle=\bfseries

This key combines the keys `coloremph` and `emphstyle`.

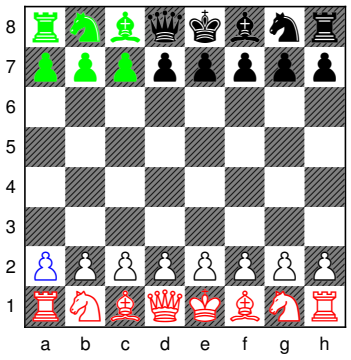
emphboard=⟨*arbitrary*⟩	emphboard
empharea=⟨*area*⟩	empharea=a1-c3
emphareas=⟨*list of areas*⟩	emphareas={f6-h7,a1-c3}
emphfile=⟨*file*⟩	emphfile=a
emphfiles=⟨*list of files*⟩	emphfiles={g,h}
emphrank=⟨*rank*⟩	emphrank=5
emphranks=⟨*list of ranks*⟩	emphranks={8,7}
emphfield=⟨*field*⟩	emphfield=a3
emphfields=⟨*list of fields*⟩	emphfields={b5,c6}

This keys apply the emphasize commands and eventually the colors to the area they define.

```
\def\empharea{ h8-f4 }
\chessboard[boardfontfamily=millennia,
          coloremphstyle=\bfseries,
          empharea=\empharea,
          whitepiececolor=red,
          emphrank=1,
          whitepiececolor=blue,
          emphareas={a2-a2, c1-e2},
          coloremph=false,
          emphfield=f2]
```




```
\chessboard[boardfontencoding=LSBC3,
          coloremph,
          emphfield={f1},
          whitepiececolor=red,
          emphranks={1},
          whitepiececolor=blue,
          emphfields=a2,
          blackpiececolor=green,
          empharea=a8-c7]
```



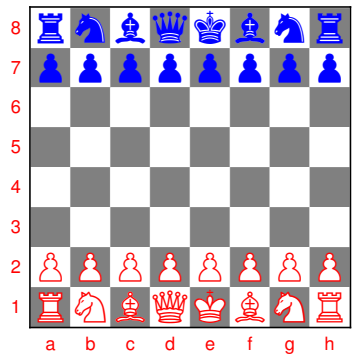
The key `colorpieces` doesn't work anymore, it will issue only a warning. Use `emphfields` instead.

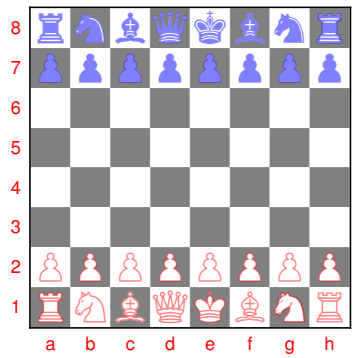
4.7.6 Transparency/opacity

Pgf has commands to set the opacity of a color. This works only for some drivers and output formats (e.g. with pdfLaTeX), but it also works outside pgf pictures in running text. The main drawbacks are that the opacity settings gets lost at a pagebreak and that TeX-groups and boxes are not respected, that means that you must reset the opacity explicitly. It is possible to add to boxes that use internally `\color@begingroup/""\color@endgroup`[5] some code that resets the opacity outside the box to 1. As `\chessboard` uses only such safe boxes it is possible to smuggle transparency in parts of the composed chars. But the whole is not very safe so use it at your own risk.

```
\makebox[0pt][l]{\rule{12em}{10pt}}%
\color{red}%
\rule{1em}{14pt} ABC
\pgfsetfillopacity{0.5}%
\rule{1em}{14pt} ABC
\pgfsetfillopacity{1}%
\rule{1em}{14pt} ABC

\makeatletter
\let\color@endgroupORI\color@endgroup
\def\color@endgroup
 {\color@endgroupORI\pgfsetfillopacity{1}}

\chessboard[boardfontencoding=LSBC4,
   whitepiececolor=red,
   blackpiececolor=blue,
   setfontcolors]

\def\cfss@whitepiececolor
 {\pgfsetfillopacity{0.5}\color{red}}
\def\cfss@blackpiececolor
 {\pgfsetfillopacity{0.5}\color{blue}}

\chessboard[boardfontencoding=LSBC4]
```

ABC ABC ABC





4.8 Labels

In the package chessfss I used the name sidefont for the label. I did it mostly because I didn't dare to use the command `\labelfont`, I was quite sure that somewhere in the packages for LaTeX someone else had already used this name. When using keys name clash are not a

[5] Almost all LaTeX-boxes but not e.g. `\mbox` and `\makebox` when used without optional argument.

problem as internally a unique prefix is used, so I decided to use names starting with “label” in this package.

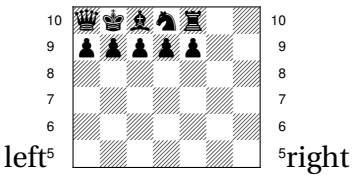
`labelleft=`⟨*true*|*false*⟩	`labelleft`	`true`
`labelright=`⟨*true*|*false*⟩	`labelright=false`	`false`
`labeltop=`⟨*true*|*false*⟩	`labeltop`	`false`
`labelbottom=`⟨*true*|*false*⟩	`labelbottom`	`true`
`hlabel=`⟨*true*|*false*⟩	`hlabel`	`false`
`vlabel=`⟨*true*|*false*⟩	`vlabel`	`false`
`label=`⟨*true*|*false*⟩	`label`	`false`

You can disable the labels with this boolean keys.

`labelleftwidth=`⟨*length*⟩	`labelleftwidth=1em`	`1ex`
`labelrightwidth=`⟨*length*⟩	`labelrightwidth=1em`	`1ex`
`hlabelwidth=`⟨*length*⟩	`hlabelwidth=1em`	`1ex`

The labels on the left side are set raggedleft, the ones on the right side raggedright at the distance given by the keys. This width don’t change the size of the board! The top and bottom labels are centered.

```
\def\mywidth{1em}
left%
\chessboard[maxfield=j10,
          tinyboard,
          printarea=d5-j10,
          hmarginwidth=1em,
          hlabelwidth=\mywidth,
          hlabel,
          vlabel=false]%
right
```



`labelleftlift=`⟨*length*⟩	`labelleftlift=1em`	`0.35em`
`labelrightlift=`⟨*length*⟩	`labelrightlift=1em`	`0.35em`
`hlabellift=`⟨*length*⟩	`hlabellift=1em`	`0.35em`

With this keys you can move up and down the left and right labels.

labeltoplift=⟨*length*⟩	labeltoplift=1em	1.1\baselineskip
labelbottomlift=⟨*length*⟩	labelbottomlift=1em	0.2\baselineskip
vlabellift=⟨*length*⟩	vlabellift=1em	-

With this keys you can set the distance of the baselines of the top and bottom labels from the board sides. \baselineskip, ex and em refer in this keys *not* to the size of the board, but to the size of the label font – I thought this would be easier to handle.

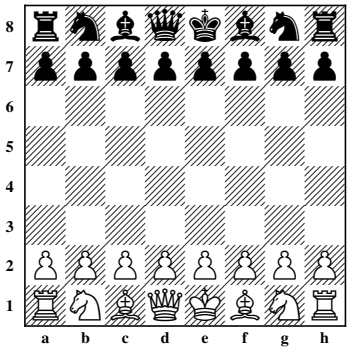
labelleftfont=⟨*fontcommand*⟩	labelleftfont=\itshape	\sfdefault
labelrightfont=⟨*fontcommand*⟩	labelrightfont=\itshape	\sfdefault
labeltopfont=⟨*fontcommand*⟩	labeltopfont=\itshape	\sfdefault
labelbottomfont=⟨*fontcommand*⟩	labelbottomfont=\itshape	\sfdefault
hlabelfont=⟨*fontcommand*⟩	hlabelfont=\itshape	\sfdefault
vlabelfont=⟨*fontcommand*⟩	vlabelfont=\itshape	\sfdefault
labelfont=⟨*fontcommand*⟩	labelfont=\itshape	\sfdefault

With this keys you can set the font used by the labels. As a default (this is set in the package chessfss) the sans serif font of the document is used. You can also use the the package chessfss commands to change the font.

labelfontsize=⟨*length*⟩	labelfontsize=0.5em	10pt

This key sets the font size of the label. The default value is set by the key normalboard. There aren't keys to set the size of each label separatly. If you really need different sizes, use the key hlabelfont etc (and be careful when using font relative sizes).

```
\setchessboard{labelfont=\bfseries}
\setsidefontfamily{ptm} % from chessfss.sty
\chessboard[labelfontsize=6pt]
```



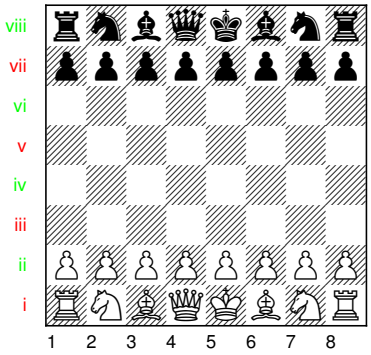
labelleftformat=⟨*commands*⟩	labelleftformat	1
labelrightformat=⟨*commands*⟩	labelrightformat	1

labeltopformat=⟨*commands*⟩	labeltopformat	h
labelbottomformat=⟨*commands*⟩	labelbottomformat	h
hlabelformat=⟨*commands*⟩	hlabelformat	1
vlabelformat=⟨*commands*⟩	vlabelformat	h
labelformat=⟨*commands*⟩	labelformat	

With this keys you can control how the number of the files and ranks are printed. `ranklabel` and `filelabel` are counters that contain the current rank and file. You can use them freely (even change if you want).

The commands can be a lot of things. You can e.g. use them to color the labels or to move the labels around. When you use commands with optional arguments you must put braces around the value of the key!

```
\def\mylabelformat{%
 {\makebox[0pt][r]{%
  \ifthenelse{\isodd{\value{ranklabel}}}
    {\color{red}\roman{ranklabel}}
    {\color{green}\roman{ranklabel}}}}}

\chessboard[hlabelformat=\mylabelformat,
   vlabelformat=\arabic{filelabel}\hfill]
```



4.9 The mover

The “mover” is a sign at the side of the board that indicates which player is to move. `\chessboard` gets this information either from the FEN (if it contains the information as is the case when the FEN comes from the package skak), or through the following key:

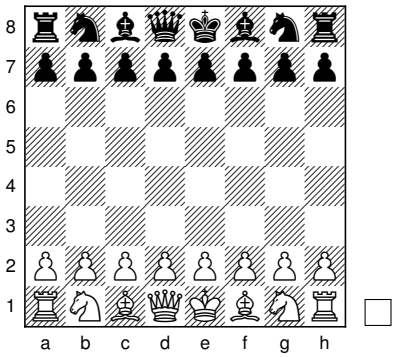
mover=⟨*w*|*b*⟩	mover=b

You can use this key more than once, e.g. to control the mover field in the FEN that you save, the last value is used for the print.

showmover=⟨*true*|*false*⟩	showmover=false	true

This key switch the print of the mover sign on and off.

```
\chessboard[smallboard,showmover=true]
```



moversize=⟨*length*⟩	moversize=1ex	1em

This key sets the fontsize of the mover. It only has an effect if the mover sign comes from a font – and if the definition doesn't change the fontsize.

The value is saved in an inner command with the name `\board@val@moversize`, you can use this command in your own mover definitions.

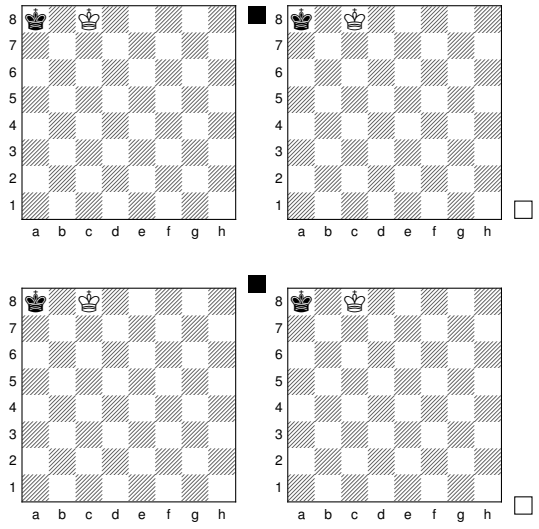
moverlift=⟨*length*⟩	moverlift=1ex	0pt

movertoplift=⟨*length*⟩	movertoplift=1ex	0pt

moverbottomlift=⟨*length*⟩	moverbottomlift=1ex	0pt

With this keys you can move the mover vertically. Positive values will move the bottom mover up and the top mover down[6]. 0pt will align the bottom mover along the baseline, and the top of the upper mover along the upper side of the board.

```
\setchessboard{showmover}%
\chessboard[tinyboard, setfen=k1K b]%
\chessboard[tinyboard, setfen=k1K w]


\chessboard[tinyboard, setfen=k1K b,
         moverlift=-1ex]%
\chessboard[tinyboard, setfen=k1K w,
         moverlift=-1ex]
```

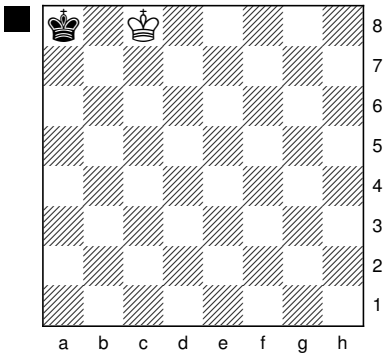


[6]This isn't perhaps the correct meaning of the word "lift" but I guessed that user expect the mover to move either both towards or apart.

movershift=⟨*length*⟩	movershift=1ex	1ex
movertopshift=⟨*length*⟩	movertopshift=1ex	1ex
moverbottomshift=⟨*length*⟩	moverbottomshift=1ex	1ex

With this keys you can shift the mover horizontaly. If you want to put it on the left side of the board, use a large negative value.

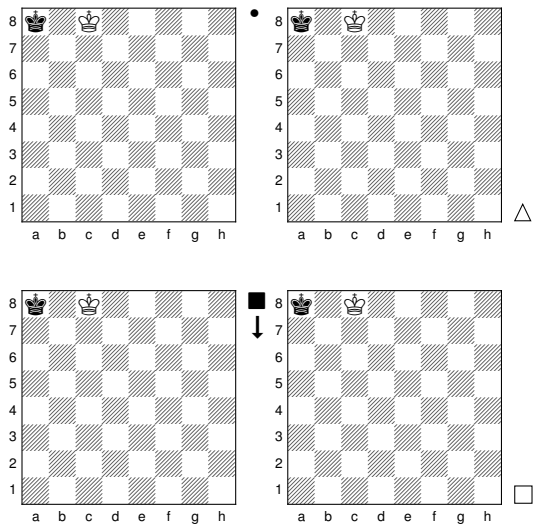
```
\setchessboard{showmover}%
\chessboard[setfen=k1K b,
         movershift=-9em,
         labelleft=false,
         labelright]%
```



moverstyle=⟨*typename*⟩	moverstyle=squarearrow	square

With this key you can change the mover type. Up to now four types are defined, but you can define your own one if you want.

```
\setchessboard{showmover}%
\chessboard[tinyboard, setfen=k1K b,
         moverstyle=circle]%
\chessboard[tinyboard, setfen=k1K w,
         moverstyle=triangle]

%This moverstyle needs pifont and graphics!
\chessboard[tinyboard, setfen=k1K b,
         moverstyle=squarearrow]%
%This will give a warning and square is used:
\chessboard[tinyboard, setfen=k1K w,
         moverstyle=unknown]
```



efineMoverStyle With this command you can define your own moverstyle. \cbDefineMoverStyle takes six arguments (the first is optional). The first mandatory argument is the name of the style, the

following arguments get the definitions for the mover for black and white for normal and inverse board.

`\cbDefineMoverStyle[`⟨*commands*⟩`]{`⟨*name of style*⟩`}{`⟨*white top*⟩`}`
`{`⟨*white bottom*⟩`}{`⟨*black top*⟩`}{`⟨*black bottom*⟩`}`.

The listing shows the definition of the mover “squarearrow”:

```
\cbDefineMoverStyle%
 %#1: optional, can be used e.g. for checks
 %#2=style name, #3=white top, #4=white bottom,
 %#5=black top, #6=black bottom
   [\@ifundefined{rotatebox}%
    {\PackageError{chessboard}%
     {You must load the package graphics or graphicx
      if you want to use the mover style squarearrow}{}}%
    {}%
   \@ifundefined{ding}%
    {\PackageError{chessboard}%
     {You must load the package pifont
      if you want to use the mover style squarearrow}{}}%
    {}]
   {squarearrow}% #2
   {\rotatebox{-90}{$\square$\,\ding{222}}}%
   {\rotatebox{90}{$\square$\,\ding{222}}}%
   {\rotatebox{-90}{$\blacksquare$\,\ding{222}}}%
   {\rotatebox{90}{$\blacksquare$\,\ding{222}}}%
```

5 Controlling the printing

5.1 Printing partial boards

`print=`⟨*true*|*false*⟩	`print=false`	`true`

With this key you can disable the printing of the board. This is useful if you want to use `\chessboard` to save FEN.

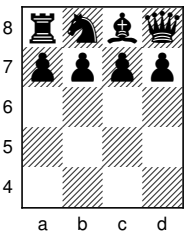
```
left\chessboard[print=false]right
```

leftright

`startprint=`⟨*field*⟩	`startprint=b7`
`stopprint=`⟨*field*⟩	`stopprint=f2`
`printarea=`⟨*area*⟩	`printarea=c2-d4`

With this keys you can define the area that gets printed.

```
\newchessgame
left\chessboard[printarea=a4-d8]right
```

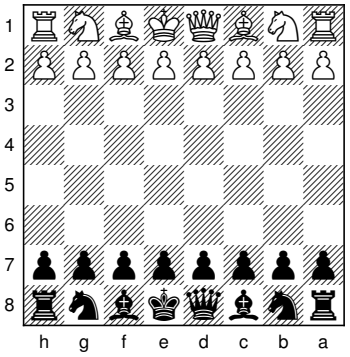
left  right

5.2 Rotating the board

`inverse=`⟨*true*|*false*⟩	`inverse=false`	`false`

With this key you can print the black side at the bottom. It won't switch labels or margins etc.

```
\newchessgame
left\chessboard[inverse]right
```

left  right

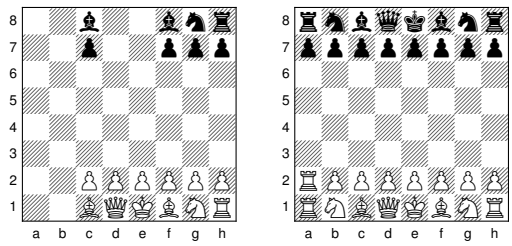
5.3 Hiding the content of the board

5.3.1 Hiding the content of fields

hideboard=⟨*arbitrary*⟩	hideboard
hidearea=⟨*area*⟩	hidearea=a1-c3
hideareas=⟨*list of areas*⟩	hideareas={a1-c3,g7-h8}
hidefile=⟨*file*⟩	hidefile=g
hidefiles=⟨*list of files*⟩	hidefiles={g,h}
hiderank=⟨*rank*⟩	hiderank=5
hideranks=⟨*list of ranks*⟩	hideranks={8,7}
hidefield=⟨*field*⟩	hidefield=c6
hidefields=⟨*list of fields*⟩	hidefields={b5,c6}

With this keys you can hide pieces on certain fields. The pieces are not deleted, e.g. when you save the position to FEN they will be there. But the content of hidden fields don't appear if you add pieces to this fields!

```
\newchessgame
\def\myfiles{a,b}
\setchessboard{tinyboard}
\chessboard[hidefiles=\myfiles,
         addpieces=Ra2,
         hidearea=d7-e8,
         storefen=myfen]
\chessboard[restorefen=myfen]
```

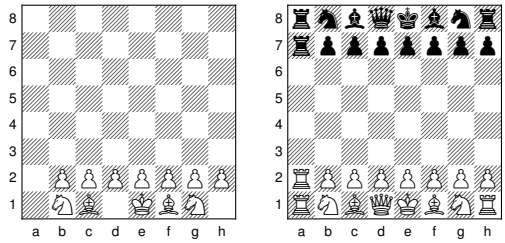


5.3.2 Hiding piecetypes

hidepiece=⟨*piece char*⟩	hidepiece=q
hidepieces=⟨*list of piece chars*⟩	hidepieces={K, b}
hidewhite=⟨*arbitrary*⟩	hidewhite
hideblack=⟨*arbitrary*⟩	hideblack
hideall=⟨*arbitrary*⟩	hideall

This keys doesn't hide the content of fields but piecestypes like the kings or the pawns. Here too: you can't add pieces if their type is hidden.

```
\newchessgame
\def\mypieces{R,Q}
\setchessboard{tinyboard}
\chessboard[hidepieces=\mypieces,
         addpieces={Ra2, ra7},
         hideblack,
         storefen=myfen]%
\chessboard[restorefen=myfen]
```

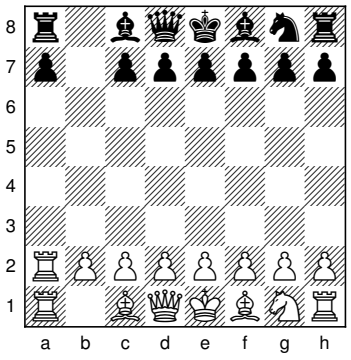


5.3.3 Showing the content of fields

showboard=⟨*arbitrary*⟩	showboard
showarea=⟨*area*⟩	showarea=a1-c3
showareas=⟨*list of areas*⟩	showareas={a1-c3,g7-h8}
showfile=⟨*file*⟩	showfile=g
showfiles=⟨*list of files*⟩	showfiles={g,h}
showrank=⟨*rank*⟩	showrank=5
showranks=⟨*list of ranks*⟩	showranks={8,7}
showfield=⟨*field*⟩	showfield=c6
showfields=⟨*list of fields*⟩	showfields={b5,c6}

With this keys you can let reappear the content of hidden fields.

```
\newchessgame
\def\myfiles{a,b}
\chessboard[hidefiles=\myfiles,
         addpieces=Ra2,
         showfiles=a,
         showranks=2]
```

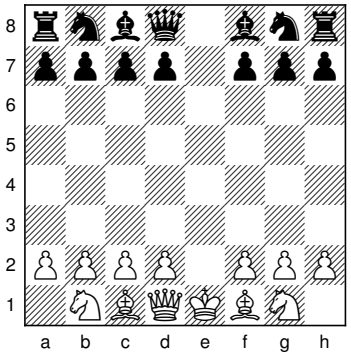


5.3.4 Showing piecetypes

showpiece=⟨*piece char*⟩	showpiece=q
showpieces=⟨*list of piece chars*⟩	showpieces={K, b}
showwhite=⟨*arbitrary*⟩	showwhite
showblack=⟨*arbitrary*⟩	showblack
showall=⟨*arbitrary*⟩	showall

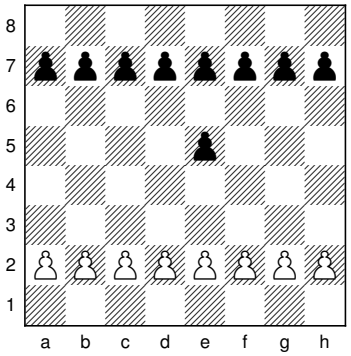
And this keys lets reappear hidden piecestypes. Attention: hidden pieces on hidden fields reappear only if you use the key to show the field *and* the key to show the piece!

```
\newchessgame
\def\myfiles{a,e}
\chessboard[hidefiles=\myfiles,
         hidepieces={K,R},
         showfiles=a,
         showranks=1,
         showpieces=K]
```



Keys like showonlypawns or showonlybut could be easily added as they are simple combinations of the existing keys:

```
\newchessgame
\chessboard[hideall,
         showpieces={P,p},
         addpieces=pe5]
```



6 “Decoration”: Colors, background, fancy borders, highlighting

You must load a color package like xcolor or color to be able to use colors![7]

Coloring and decorating the board is a bit complicated as there is so much that can get different colors and so much different possibilities for marks, arrows, fancy borders and so on. So it can take some time to find out how to achieve a certain effect.

Don't forget that not every effect is sensible or even works everywhere. Transparency/opacity e.g. don't work for every driver or output format or printer or previewer (the documentation of pgf says that it works for dvips only with new ghostscript versions). Some colors looks good on screen but bad if printed with a greyscale printer. Colors can change if you use another monitor. Some color combinations are a problem for color blinds.

The package skak has some own highlighting commands. They use postscript/pstricks commands and they only work if you load the package skak with the option ps. That means that you have to use the `.tex`→`.dvi`→`.ps`→`.pdf`-way to process the document. If you want to use pdfLaTeX directly you have to use a package like pst-pdf to first make pdf graphics from all boards and then include this graphics during the pdfLaTeX-run. I find this quite cumbersome and sometimes it is even difficult to get it to work.

So I decided to use pgf to implement my own highlighting commands. But I also tried to offer a possibility to use the commands of the package skak.

The pgf-commands used by the package chessboard need only pgfcore.sty and pgfbaseshapes-package.sty from the pgf-bundle. If you need pgf-commands from other parts of the bundle, you will have to load the styles yourself.

`pgf=`⟨*true*|*false*⟩	`pgf`	`true`

Pgf-commands works fine with LaTeX and pdfLaTeX but in some case (e.g. if you use `\chessboard` in an environment that calls the environment `preview`) they can induce errors. So I added a key to disable all the pgf-commands. It is also useful to disable most of the highlighting.

6.1 The pgf-pictures

The decorations in the two other layers (background layer and mark layer) use pgf-commands – or to be more precise: the two other layers are pgf-pictures.

In the first version of the package chessboard the background picture wasn't used much. I had only implemented some code to color the fields. But then I got the request to add some

[7] I must correct myself: the pgf-package will load xcolor, so you should load a color package only if you want to use it with other options.

more commands for fancy borders which did mean that I had to transfer the code which draws borders in the foreground picture to the background picture. And I got a request for coloring fields in the foreground picture which did mean that I had to transfer the code from the background to the foreground.

So I decided to rewrite and extend the codes so that each highlighting/drawing in the pgf-pictures can be done in either the background or the foreground or in both. I don't know if someone will ever have a need for a cross under the figures, but why shouldn't I offer some code to do it?

I also tried to unify the keys that affect the pgf-pictures, to improve the possibilities for the user to control which pgf-picture is affected and put a bit more order in the keys. At the end quite a lot of the code has been rearranged and changed.

6.1.1 Naming of the pgf related keys

There are more or less three set of keys

- Keys which affect only the foreground picture. The keys starts with or contains the string "**mark**".
- Keys which affect only the background picture. The keys starts with or contains the string "**back**".
- Keys that saves or set properties that are used in both picture, e.g. to change the color or the linewidth. Such keys start or contain in most cases the string "**pgf**" (with some exceptions like the key for clipping and trimming). For a lot of such keys there exist a shorter (but less descriptive) name without the "pgf". In most cases there isn't a difference between the two.

6.1.2 Executing the drawing commands

A chessboard consists of fields on which figures make moves and so the decoration of the board is field and move orientated.

There are three different types of graphical object:

- You can draw something on a set of single *fields* e.g., put a cross on each of the fields or put a border around each field. To get these pictures on or under the field the inner commands first shift the origin to the middle of field and then inserts the code stored in the style definition in the foreground or background image.
- You can draw something on or around a *region* – a rectangle made of fields –, e.g. by putting a border around the center or a rank or the whole board. Here the inner commands shift the origin to the left upper corner of the region. The name of the other corner is available in the style definition through #1.

- And you can mark a *move*. The main difference between a move and a region is that the order of the "corners" given in the argument matters. For moves the origin is in the first corner (the from field).

The drawing is done by first setting with various key the style and the properties of the pgf object and then executed by telling `\chessboard` which fields, moves or regions should get the drawings.

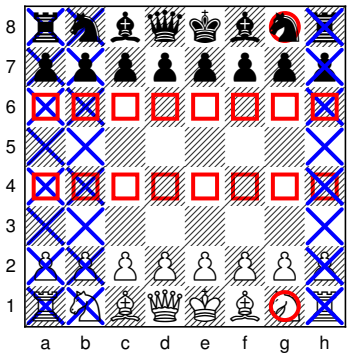
6.1.3 Drawing on and under fields

`markboard=`⟨*arbitrary*⟩	`markboard`
`markarea=`⟨*area*⟩	`markarea=a1-c3`
`markareas=`⟨*list of areas*⟩	`markareas={a1-c3,d5-e5}`
`markfiles=`⟨*file*⟩	`markfiles=g`
`markfiles=`⟨*list of files*⟩	`markfiles={g,h}`
`markrank=`⟨*rank*⟩	`markrank=7`
`markranks=`⟨*list of ranks*⟩	`markranks={8,7}`
`markfield=`⟨*field*⟩	`markfield=c6`
`markfields=`⟨*list of fields*⟩	`markfields={b5,c6}`
`backboard=`⟨*arbitrary*⟩	`backboard`
`backarea=`⟨*area*⟩	`backarea=a1-c3`
`backareas=`⟨*list of areas*⟩	`backareas={a1-c3,d5-e5}`
`backfiles=`⟨*file*⟩	`backfiles=g`
`backfiles=`⟨*list of files*⟩	`backfiles={g,h}`
`backrank=`⟨*rank*⟩	`backrank=7`
`backranks=`⟨*list of ranks*⟩	`backranks={8,7}`
`backfield=`⟨*field*⟩	`backfield=c6`
`backfields=`⟨*list of fields*⟩	`backfields={b5,c6}`

This keys executes the pgf-commands stored in the style command for each single field.

In the following example you should look at the difference it makes if you put a cross or a border in the background or the foreground.

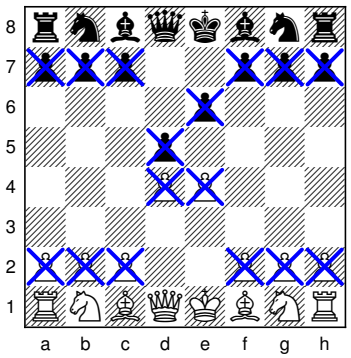
```
\newchessgame
\def\myarea{a1-b8}
\chessboard[pgfstyle=cross,
         color=blue,
         markfile=h,
         backarea=\myarea,
         color=red,
         padding=-0.2em,
         pgfstyle=circle,
         markfields=g1,backfields=g8,
         linewidth=0.1em,
         pgfstyle=border,
         markranks=6, backrank=4]
```



The new key `getpiecelists` can be used to mark e.g. all pawns on a board:

1 e4 e6 2 d4 d5

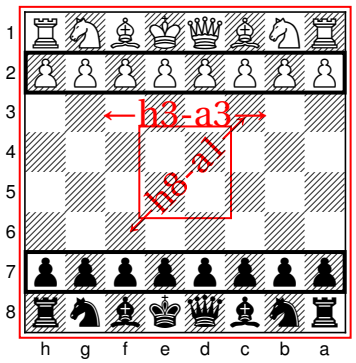
```
\newchessgame
\mainline{1. e4 e6 2.d4 d5}

\chessboard[pgfstyle=cross,
         color=blue,
         getpiecelists,
         markfields={\cblistP,\cblistp}]
```



6.1.4 Drawing on regions

`markregions=`⟨*list of regions*⟩	`markregions={a1-c3, b7-c6}`
`markregion=`⟨*list of regions*⟩	`markregion={a1-c3, b7-c6}`
`backregions=`⟨*list of regions*⟩	`backregions={a1-c3, b7-c6}`
`backregion=`⟨*list of regions*⟩	`backregion={a1-c3, b7-c6}`

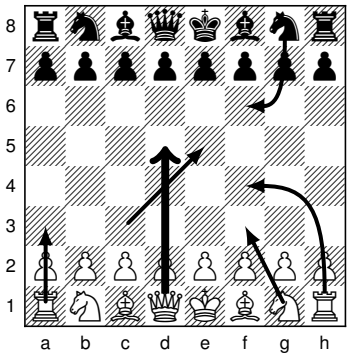
```
\newchessgame
\def\pawnranks{a2-h2, h7-a7}
\chessboard[inverse,
         pgfstyle=border,
         markregions=\pawnranks,
         color=red, padding=0.3ex,
         linewidth=0.1ex,
         backregions={\board,d4-e5},
         pgfstyle=text,
         text=$\leftarrow$%
             \currentbk-\currentwq
             $\rightarrow$,
         markregion=a3-h3,
         pgfstyle={[rotate=45]text},
         backregion=h8-a1]
```



6.1.5 Drawing move related

markmoves=⟨*list of moves*⟩	markmoves={a1-c3, b7-c6}
markmove=⟨*list of moves*⟩	markmove={a1-c3, b7-c6}
backmoves=⟨*list of moves*⟩	backmoves={a1-c3, b7-c6}
backmove=⟨*list of moves*⟩	backmove={a1-c3, b7-c6}

```
\def\mymove{d1-d5}%
\chessboard[pgfstyle=straightmove,
         markmoves={a1-a3, c3-e5, g1-f3},
         pgfstyle=knightmove,
         backmoves={g8-f6, h1-f4},
         arrow=to, linewidth=0.2em,
         markmove= \mymove]
```



I'm not very good in designing beautiful curves and moves arrows. If someone could offer me better code I will be happy to implement it.

6.1.6 Choosing what is drawn: pgf styles

markstyle=⟨*[options]style name*⟩	markstyle=cross
backstyle=⟨*[options]style name*⟩	backstyle=cross
pgfstyle=⟨*[options]style name*⟩	pgfstyle=[left]text

With this keys you set the style the drawing commands will use. `markstyle` sets only the style for the foreground (the "mark" picture). `backstyle` sets only the style for the background picture. `pgfstyle` sets the style for both pictures (`style` is not a shorter version of `pgfstyle`!).

If the value starts with `[` the text until `]` is interpreted as an option of the style. It depends on the definition of the style if and how this option is used (until now the text style, the circle style and (since version 1.5) the curvemove style use such options). **Attention**: to prevent `\chessboard` to interpret the `]` as the end of its optional argument you must put braces around the whole value if you want to use options.

Not every style is useful for everything, e.g. it doesn't make sense to put a border around a move. The following tabular shows the predefined styles and the types for which they work.

	fields	regions	moves
cross	✔		
circle	✔		
text	✔	✔	
border(s)	✔	✔	
color	✔	✔	
straightmove			✔
knightmove			✔
curvemove			✔

`nePgfFieldStyle`
`ePgfRegionStyle`
`inePgfMoveStyle`

With this commands you can define your own styles. The commands take two argument: the first is the name of the style, the second should be the pgf-commands needed to draw.

In previous version of the package the commands were called. `\cbDefineMarkFieldStyle`, `\cbDefineMarkRegionStyle` and `\cbDefineMarkMoveStyle`. I have changed the names to indicate that the commands will define the styles for the background picture too. The older commands still work but I suggest that you switch to the new names.

The listing shows first an example for a field mark, then an example for a region where the second corner is accessed through the #1, and at last the definition for the text mark – here it is necessary to rotate the text if the key `inverse` is true and the definition shows how the optional argument of the `pgfstyle` key can be use with #2:

```
\cbDefinePgfFieldStyle{circle}{%
   \pgfsetlinewidth{\board@pgf@linewidth}%
   \setlength\len@board@tempx{%
```

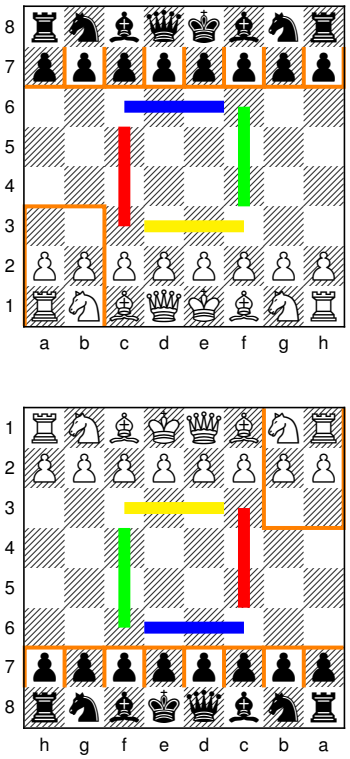
```
    \dimexpr 0.55em + \board@pgf@padding \relax}%
   \pgfpathcircle{\pgfpointxy{0}{0}}{\len@board@tempx}%
   \pgfusepath{stroke,#2}}%%

\cbDefinePgfRegionStyle{border}{%
   \pgfsetlinewidth{\board@pgf@linewidth}%
   \pgfsetcornersarced{\pgfpoint{\board@pgf@corner}{\board@pgf@corner}}
   \pgfpathrectanglecorners%
    {\pgfpointadd%
     {\pgfpoint{-\board@pgf@padding}{\board@pgf@padding}}
     {\pgfpointxy{-0.5}{0.5}}}%
    {\pgfpointadd%
     {\pgfpointanchor{#1}{center}}
     {\pgfpointadd%
      {\pgfpoint{\board@pgf@padding}{-\board@pgf@padding}}
      {\pgfpointxy{0.5}{-0.5}}}}%
   \pgfusepath{stroke}}%

\cbDefinePgfFieldStyle{text}{%
   \ifthenelse%
    {\boolean{\XKV@UFCB@locset@inverse@value}}%
    {\pgftext[rotate=180,#2]{\normalfont\board@pgf@curtext}}%
    {\pgftext[#2]{\normalfont\board@pgf@curtext}}}%
```

6.1.7 Introduction to the predefined pgf styles

Borders

Borders can be put around fields and regions. There exist also partial borders for one, two or three margins of an area, this borders are drawn and named “clockwise”. “left”, “right”, “top”, and “bottom” refer to the position of the border if the white player is at the bottom. If you use the key `inverse` the positions of the lines don’t change in relation to the players colors.

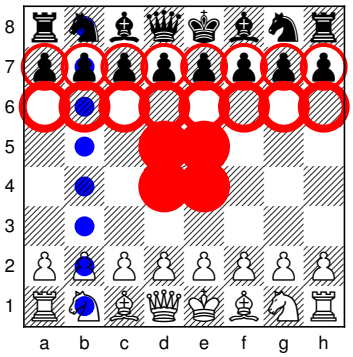
```
\newchessgame
\setchessboard{smallboard,
         pgfstyle=rightbottomborder,
         color=orange,
         markrank=7,
         pgfstyle=lefttoprightborder,
         markregion=a1-b3,
         shortenstart=-1ex,
         padding=1ex,
         linewidth=0.3em,
         pgfstyle=leftborder,color=red,
         markregion=\mycenter,
         pgfstyle=topborder,color=blue,
         markregion=\mycenter,
         pgfstyle=rightborder,color=green,
         markregion=\mycenter,
         pgfstyle=bottomborder,color=yellow,
         markregion=\mycenter}
\def\mycenter{d4-e5}
\chessboard\
\chessboard[inverse]
```



Circle

This style can only be used in connection with fields. The optional argument can be used to fill the circle (default is stroke).

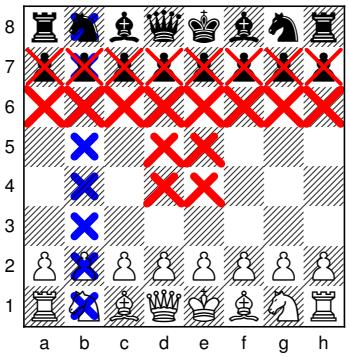
```
\newchessgame
\def\mycenter{d4-e5}
\chessboard[pgfstyle=circle,
         color=red,
         markrank=7,
         linewidth=0.2em,
         markrank=6,
         pgfstyle={[fill]circle},
         markarea=\mycenter,
         padding=-0.8ex,color=blue,
         backfile=b]
```



Cross

This style can only be used in connection with fields.

```
\newchessgame
\def\mycenter{d4-e5}
\chessboard[pgfstyle=cross,
         color=red,
         markrank=7,
         linewidth=0.2em,
         markrank=6,
         shortenstart=0.5ex,
         markarea=\mycenter,
         shortenend=0.5ex,color=blue,
         backfile=b]
```



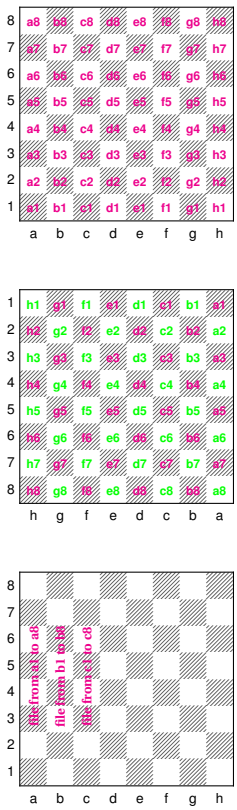
Text

Text can be used for fields and regions. It is the only style where it isn't enough to simply chose the style, you must also give a value to the text content with the key `text` (don't confuse the key "text" with the style "text"!).

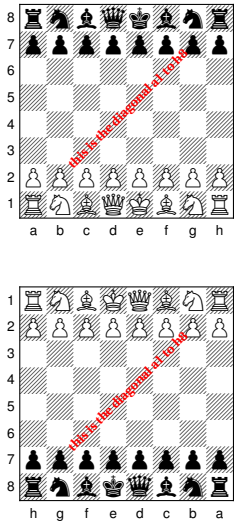
The option of the style is passed to the optional argument of the `\pgftext` command, so everything that is allowed there can be put in the option. In the definition of the text content you can use informations about the current field or the current region. `\currentbk` e.g. gives the *b*lack *k*ing corner of the region (when you draw in a region) or the field name (when you draw in a field). The opposite corner ist `\currentwq`, the *w*hite *q*ueen corner.

As a default the text is centered on the middle of the field or the region. You can use the options to move it around, please look in the documentation of pgf to find out what is possible.

```
\newchessgame
\setchessboard{tinyboard,color=magenta,clearboard}
\chessboard[
 pgfstyle=
    {[base,at={\pgfpoint{0pt}{-0.4ex}}]text},
 text= \fontsize{1.2ex}{1.2ex}\bfseries
     \sffamily\currentwq,
 markboard]

\chessboard[
 inverse,
 pgfstyle=
   {[base,at={\pgfpoint{0pt}{-0.4ex}}]text},
 text=\fontsize{1.2ex}{1.2ex}\bfseries
    \sffamily\currentwq,
 trimtocolor=black, markboard,
 trimtocolor=white,color=green,
 markboard]%

\chessboard[pgfstyle={[rotate=90]text},
         text=
         \fontsize{1.2ex}{1.2ex}
         \bfseries file from \currentwq\ to \currentbk,
         markregions={a1-a8,b1-b8,c1-c8}]%%
```



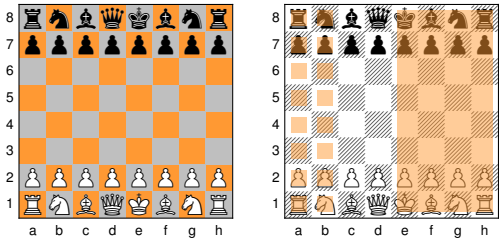

```
\newchessgame
\setchessboard{tinyboard,color=red}

\chessboard[pgfstyle={[rotate=45]text},
         text=
         \fontsize{1.2ex}{1.2ex}
         \bfseries this is the diagonal
         \currentwq\ to \currentbk,
         markregions=\board]

\chessboard[inverse, pgfstyle={[rotate=45]text},
         text=
         \fontsize{1.2ex}{1.2ex}
         \bfseries this is the diagonal
         \currentwq\ to \currentbk,
         markregions=\board]
```



Color

This style puts a colored square on or under the field or region. At the start it was meant to color the background of the fields (and so it was only defined for the background). If used for the foreground on should set an adequate opacity.

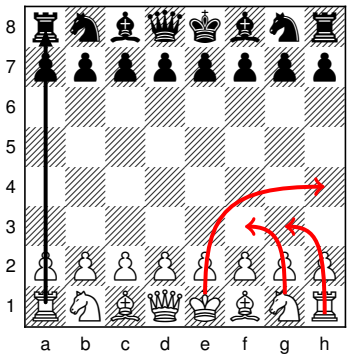
```
\newchessgame
\setchessboard{tinyboard,
      color=orange!80}
\chessboard[boardfontencoding=LSBC5,
         pgfstyle=color,
         trimtocolor=black,
         backboard,
         trimtocolor=white,
         color=gray!50,
         backboard]%
\chessboard[pgfstyle=color,
         padding=-0.2em,
         opacity=0.5,
         markfiles={a,b},
         markregion={e1-h8}]%
```



Moves

Three styles draw arrows for moves. Two are quite simple: `straightmove` which draws straight arrows, `knightmove` which make a curve suited for knight moves.

```
\newchessgame
\chessboard[pgfstyle=straightmove,
         markmove=a1-a8,
         arrow=to,linewidth=0.2ex,
         color=red,
         pgfstyle=knightmove,
         markmoves={g1-f3, e1-h4},
         shortenstart=-1ex,
         markmoves=h1-g3]%
```



curvemove New in version 1.5 there is the more sophisticated `curvemove` which draws an arbitrary bezier curve between start and end field.

`curvemove` accept an optional argument in key val syntax. (If you use it, don't forget to put braces arround the value!).

First, there are keys that lets you change the support points of the bezier curve. The curve use a local coordinate system: The x-axis goes along the line from the start point to the end point. The x-values set with the keys listed below are relative to the length of this arrow from start to end, while the y-values are absolute (unit it the width of a square). Positive y values give a curve going clockwise, negative y values give a curve going anticlockwise. With the `clockwise=false` you can change the sign of the y values.

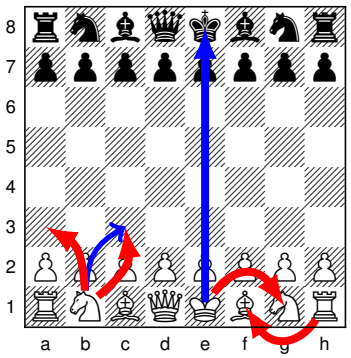
x1	= ⟨*number*⟩
x2	= ⟨*number*⟩
y1	= ⟨*number*⟩
y2	= ⟨*number*⟩
clockwise	= ⟨*true*|*false*⟩

Second, there are keys to change *locally* (only for the curve) the look of the arrow. They work like the keys for `\chessboard` with the same name: `arrow`, `pgfarrow`, `linewidth`, `pgflinewidth`, `shortenstart`, `pgfshortenstart`, `shortenend`, `pgfshortenend`.

At last there is a key `style` which you can use to reuse saved keyval-lists. Until now only style `knight` exists and there is no user command to define new ones. You want to use own keys, sent me either a feature request or do it by defining the low level command:

`\def\board@pgf@curvemovestyle@`⟨*name*⟩`{\setkeys[UFCB]{bez}{`⟨*keyval-list*⟩`}}`

```
\newchessgame
\def\mystyle{[clockwise=false,style=knight]curvemove}
\chessboard[arrow=latex,
        pgfstyle=straightmove,
        shortenstart=0.1em,
        color=blue,
        linewidth=3pt,
        markmoves={e1-e8},
        pgfstyle={[linewidth=2pt,arrow=to,
                style=knight]curvemove},
        markmove=b1-c3,
        color=red,
        pgfstyle=\mystyle,
        markmoves={b1-c3,b1-a3},
        pgfstyle=curvemove,
        markmoves={e1-g1,h1-f1}]
```



6.1.8 Setting graphic properties

As the previous examples showed you can change quite a lot things of the pictures. Now here a complete description of the needed keys. The keys don't draw anything directly. They only put commands or definitions like `\color{red}`, `\pgfsetfillopacity{0.3}` or `\def\board@pgf@arrow{to}` in the pgf-picture(s).

Let us now start with the keys which put a command in the pictures:

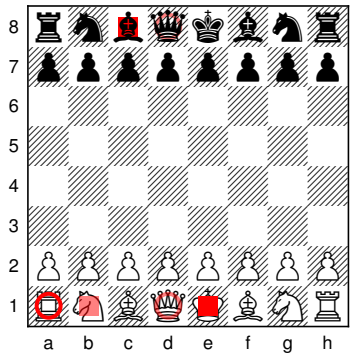
`color=`⟨*color*⟩	`color=red`
`pgfcolor=`⟨*color*⟩	`pgfcolor=red`

This keys stores a color command (`\color{red}`) in the pgf picture(s).

opacity=⟨*number*⟩	opacity=0.5
pgfopacity=⟨*number*⟩	pgfopacity=0.5
fillopacity=⟨*number*⟩	fillopacity=0.5
pgffillopacity=⟨*number*⟩	pgffillopacity=0.5
strokeopacity=⟨*number*⟩	strokeopacity=0.5
pgfstrokeopacity=⟨*number*⟩	pgfstrokeopacity=0.5

This keys add \pgfsetfillopacity{⟨*number*⟩} and/or \pgfsetstrokeopacity{⟨*number*⟩} to the pgf-picture(s). The number should be between zero and one.

```
\chessboard[padding=-0.5ex,
          color=red,fillopacity=0.5,
          markstyle=circle,markfields=a1,
          markstyle=color,markfields=b1,
          fillopacity=1,
          strokeopacity=0.5,
          pgfstyle=circle,
          markfields=d1,backfield=d8,
          pgfstyle=color,
          markfields=e1,backfield=c8]
```



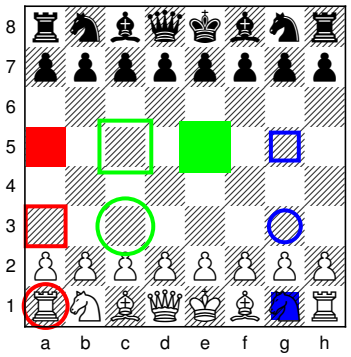
The following keys puts only a definition in the pgf-pictures. The values will only affect the styles that use the internal commands in their definitions.

padding=⟨*length*⟩	padding=0.1ex
pgfpadding=⟨*length*⟩	pgfpadding=0.1ex

This keys save \def\board@pgf@padding{⟨*length*⟩} to the pgf-picture(s). The value can be used in various situations to change the offset of a line or a border of the area that is covered by a color.

Used currently by the border styles, the styles that color fields in the background or the foreground to enlarge or reduce the covered area and the circle (the length is added to a start diameter).

```
\chessboard[color=red,
         markstyle=circle,markfields=a1,
         markstyle=border,markfields=a3,
         markstyle=color,markfields=a5,
         padding=0.3ex,color=green,
         markstyle=circle,markfields=c3,
         markstyle=border,markfields=c5,
         markstyle=color,markfields=e5,
         padding=-0.3ex,color=blue,
         markstyle=circle,markfields=g3,
         markstyle=border,markfields=g5,
         backstyle=color,backfields=g1]
```

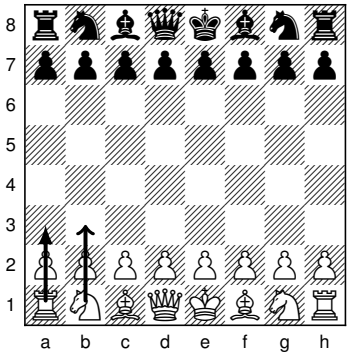


arrow=⟨*arrow type*⟩	arrow=to
pgfarrow=⟨*arrow type*⟩	pgfarrow=to

This keys save \def\board@pgf@arrow{⟨*arrow type*⟩} to the pgf-picture(s). Styles can use the command to set the arrow tips. Look at the documentation of pgf to learn about the existing arrow types (and how to define new ones).

Default is latex. Used currently by the move styles.

```
\chessboard[markstyle=straightmove,
         markmove=a1-a3,
         arrow=to,
         markmove=b1-b3]
```



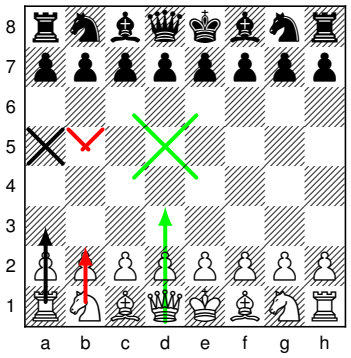
shortenstart=⟨*length*⟩	shortenstart=to
pgfshortenstart=⟨*length*⟩	pgfshortenstart=to
shortenend=⟨*length*⟩	shortenend=to
pgfshortenend=⟨*length*⟩	pgfshortenend=to
shorten=⟨*length*⟩	shorten=to
pgfshorten=⟨*length*⟩	pgfshorten=to

This keys save the definitions \def\board@pgf@shortenstart{⟨*length*⟩} or \def\board@pgf@shortenend{⟨*length*⟩} or both to the pgf-picture(s). The commands can

be used e.g. in a `\pgfsetshortenstart` command to shorten a line a the start or the end.

As default the lengths are 0pt. Used e.g. by the styles straightmove, knightmove, cross and the partial borders. The lengths are *added* to some start values that I thought reasonable. If you use large length (over 1ex) this can change the meaning of move marks as the mark will start in another field!

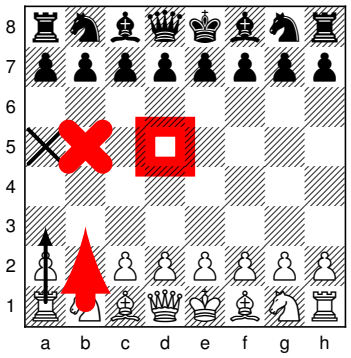
```
\chessboard[markstyle=straightmove,
          markmove=a1-a3,
          markstyle=cross,markfield=a5,
          shortenend=1ex,color=red,
          markstyle=straightmove,
          markmove=b1-b3,
          markstyle=cross,markfield=b5,
          shorten=-0.5em,color=green,
          pgfstyle=straightmove,
          backmove=d1-d3,
          backstyle=cross,backfield=d5]
```



	`linewidth=`⟨*length*⟩	`linewidth=1ex`
	`pgflinewidth=`⟨*length*⟩	`pgflinewidth=1ex`
relict	`marklinewidth=`⟨*length*⟩	`marklinewidth=1ex`
relict	`backlinewidth=`⟨*length*⟩	`backlinewidth=1ex`

This keys save `\def\board@pgf@linewidth{`⟨*length*⟩`}` to the pgf-picture(s). The command can be used e.g. in `\pgfsetlinewidth`. The keys `marklinewidth` and `backlinewidth` are remains from an older version. There put the commands only in one picture, and they ignore the values of the `usepgf` keys.

```
\chessboard[markstyle=straightmove,
          markmove=a1-a3,
          markstyle=cross,markfield=a5,
          linewidth=1ex,color=red,
          markstyle=straightmove,
          markmove=b1-b3,
          markstyle=cross,markfield=b5,
          pgfstyle=border,backfield=d5]
```

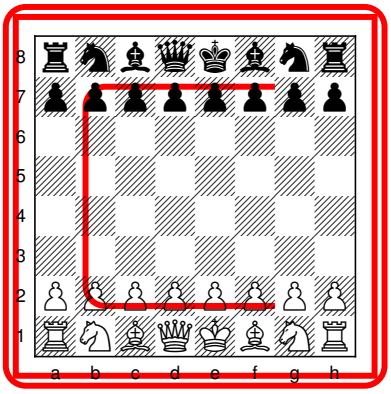


cornerarc=⟨*length*⟩	cornerarc=2pt
pgfcornerarc=⟨*length*⟩	pgfcornerarc=2pt
relict backcornerarc=⟨*length*⟩	backcornerarc=2pt

The first two keys save \def\board@pgf@corner{⟨*length*⟩} to the pgf-picture(s). The last key will save the same definition only to the background picture and it will ignore the value of the usebackpgf and usepgf keys. \board@pgf@corner can be used e.g. in \pgfsetcornersarced to round the corners of a border. To get sharp corners use the value 0pt.

The value is used currently only used in the border styles.

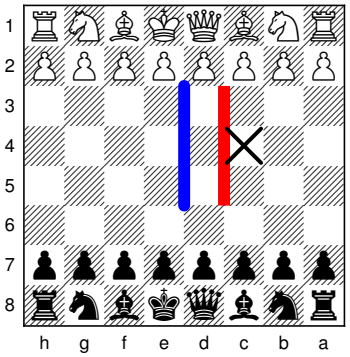
```
\chessboard[linewidth=0.3ex,color=red,
          padding=1ex,
          pgfstyle=border,
          backregion=\board,
          cornerarc=1ex,
          padding=1.5ex,
          backregion=\board,
          pgfstyle=bottomlefttopborder,
          backregion=c3-f6]
```



addpgf=⟨*(pgf)-commands*⟩	addpgf=\pgfsetroundcap

With this key you can add arbitrary (pgf)-commands to the pictures. Use it with care. When you use the key without an argument it will end a pgf scope and so reset colors and linewidth and some of the other values to their defaults.

```
\chessboard[
      showmover=false,
      inverse,markstyle=leftborder,
      color=red,linewidth=0.3em,
      markregion=d3-d5,
      color=blue,
      addpgf=\pgfsetroundcap,
      markregion=e3-e5,
      addpgf,
      markstyle=cross,
      markfield=c4]
```



usepgf=⟨*back*|*mark*|*all*|*both*|*none*⟩	usepgf

With this key you can control in which picture the commands or definitions are saved when using the keys described above.

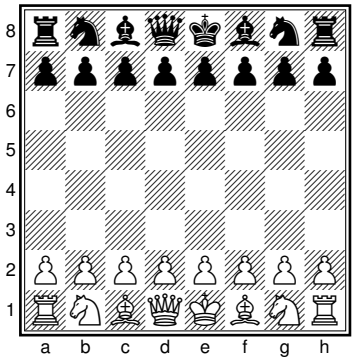
The default is to save everything in both pictures. This means that is also adds some superfluous code to the other picture, but I don't think this is a problem. In most cases you won't need the key.

6.1.9 A special shortcut key for background border

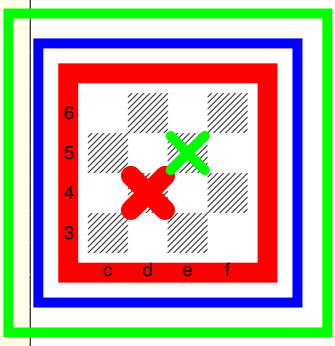
pgfborder=⟨*area*⟩	pgfborder=c3-e4

With this key you can put a border around the area in the background picture. The key saves the current style, calls the `backstyle` and the `backregions` keys to draw the border and then restores the old style. If you don't give a value to the key, the border is around the printarea.

```
\newchessgame
\chessboard[%
 showmover=false,
 border=false,
 linewidth=0.4pt,
 padding=0.4pt,
 pgfborder,
 linewidth=1pt,
 padding=2pt,
 pgfborder]
```



```
\chessboard[
 showmover=false,
 border=false,
 hlabelwidth=0.6em,
 vlabellift=1.3em,
 printarea=c3-f6,
 pgfstyle=cross,
 color=red,
 linewidth=1ex,
 padding=1ex,
 pgfborder,
 markfield=d4,
 color=blue,
 linewidth=0.5ex,
 padding=2.5ex,
 pgfborder,
 padding=0ex,
 color=green,
 trim=false,%to show the next border
 pgfborder=\board,
 markfield=e5]
```

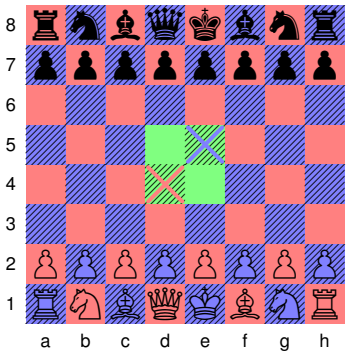


6.1.10 Special shortcut keys for background color

colorbackboard=⟨*arbitrary*⟩	colorbackboard
colorbackarea=⟨*area*⟩	colorbackarea=a1-c3
colorbackareas=⟨*list of areas*⟩	colorbackareas={a1-c3, f1-f2}
colorbackfile=⟨*file*⟩	colorbackfile=h
colorbackfiles=⟨*list of files*⟩	colorbackfiles={g,h}
colorbackrank=⟨*rank*⟩	colorbackrank=5
colorbackranks=⟨*list of ranks*⟩	colorbackranks={8,7}
colorbackfield=⟨*field*⟩	colorbackfield=g4
colorbackfields=⟨*list of fields*⟩	colorbackfields={b5,c6}
colorwhitebackfields=⟨*list of fields*⟩	colorwhitebackfields={b5,c6}
colorblackbackfields=⟨*list of fields*⟩	colorblackbackfields={b5,c6}

This keys are shortcuts to color the background of the board. They save the current style, color the fields and then restore the style.

```
\chessboard[
 showmover=false,
 border=false,
 pgfstyle=cross,
 color=red!50,
 colorwhitebackfields,
 markfield=d4,
 color=blue!50,
 colorblackbackfields,
 markfield=e5,
 color=green!50,
 colorbackarea=d4-e5]
```



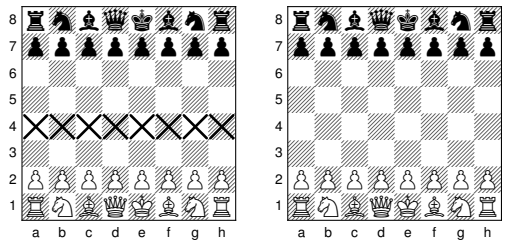
6.1.11 Clearing the pgf-pictures

Removing pgf decoration from the board is easy if you used the optional argument of `\chessboard`. But what should you do if you had added e.g. a mark with `\setchessboard` for the next four board and now want to get rid of it for the fifth? That isn't so easy as the code is burried quite deep in a large list.

On the whole there are three possibilities:

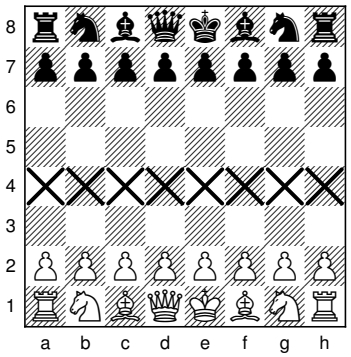
- Use grouping to keep the marks local to some boards.

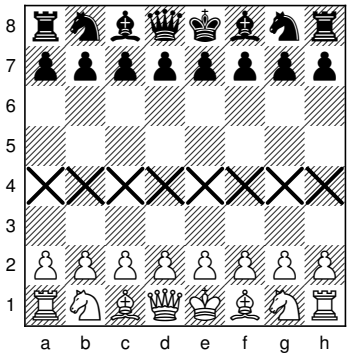
```
\newchessgame
\setchessboard{tinyboard}
{\setchessboard{markstyle=cross,
            markranks=4}
\chessboard}%
\chessboard
```



- Store the keys in a style that you can redefine:

```
\newchessgame
\storechessboardstyle{mymarks}
  {markstyle=cross,
   markranks=4}

\setchessboard{style=mymarks}
\chessboard%
\storechessboardstyle{mymarks}{}
\chessboard
```



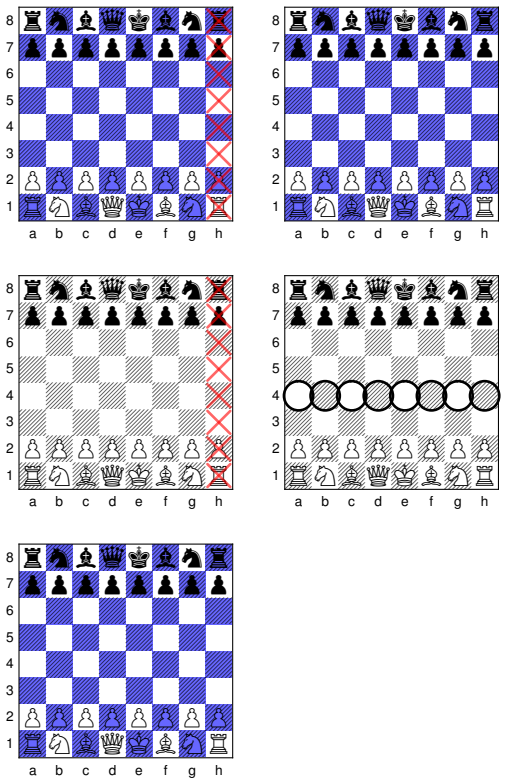


- Use the following keys to clear the pgf pictures.

`clearpgf=`⟨*arbitrary*⟩	`clearpgf`
`clearbackpgf=`⟨*arbitrary*⟩	`clearbackpgf`
`clearmarkpgf=`⟨*arbitrary*⟩	`clearmarkpgf`

(I renamed the keys `clearforegroundpgf` and `clearbackgroundpgf`, the older names will work too, but I suggest that you switch to the new ones.)

```
\newchessgame
\setchessboard{tinyboard,
            color=blue,
            opacity=0.6,
            colorblackbackfields,
            color=red,
            markstyle=cross,
            markfiles=h}
\chessboard%
\chessboard[clearmarkpgf]
\chessboard[clearbackpgf]%
\chessboard[clearpgf,
          markstyle=circle,
          markranks=4]
\setchessboard{clearmarkpgf}%
\chessboard
```



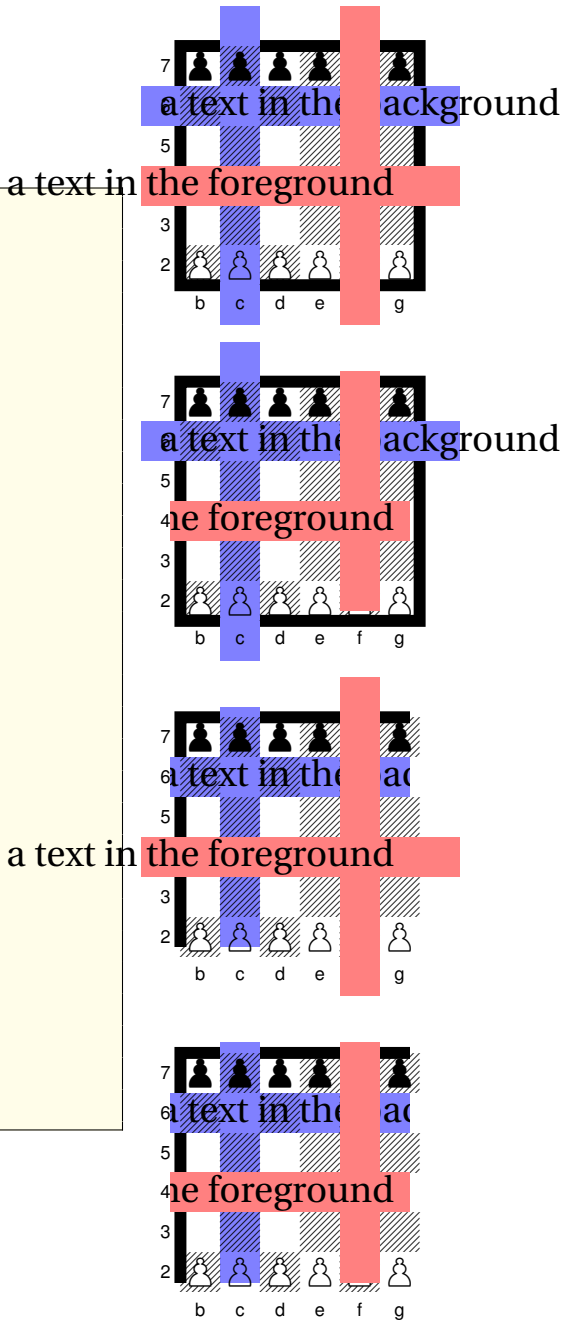
6.1.12 Clipping the pgf pictures

`clip=`⟨*key=value list*⟩	`clip`	`false`
`markclip=`⟨*key=value lists*⟩	`markclip`	`false`
`backclip=`⟨*key=value lists*⟩	`backclip`	`false`

With this keys you can add *one* clipping path to each pgf pictures. The clipping paths are always around the print area. The argument of the keys is key-value-list with the possible keys `true`, `false` to enable or disable the clipping and the four keys `left`, `top`, `right` and `bottom` which take as value a length and are used to set the padding.

As in other keys the value `true` can be omitted, but this also means that you have to use the value `false` explicitly if you only want to set some of the padding values but not enable clipping.

```
\newchessgame
\setchessboard{
  trim=false,printarea=b2-g7,
  clip={left=0.5ex,top=0.5ex,
       right=-0.5ex,bottom=-0.5ex,
       false},
  %
  border=false,linewidth=0.3em,
  pgfborder,
  %
  pgfstyle=color,
  color=blue!50,
  backrank=6,backfile=c,
  color=red!50,
  markrank=4,markfile=f,
  %
  pgfstyle=text,color=black,
  text=a text in the foreground,
  markfield=b4,
  text=a text in the background,
  backfield=f6}%

\chessboard

\chessboard[markclip]

\chessboard[backclip]

\chessboard[clip]
```



6.1.13 Trimming

Clipping cuts simply through the pgf-pictures and throws everything away that is outside the printarea. This has the drawback that in the picture there could be e.g. remains of a move arrow that started outside the shown board.

Trimming is an attempt for a more intelligent approach: Objects like crosses, circles, text, borders and arrows are discarded completely if they lie partly or completely outside the

defined scope of fields.

Trimming to a area

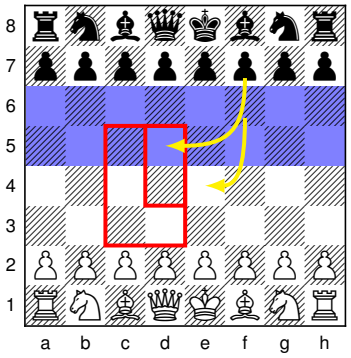
`trimarea=`⟨*area*|*empty*⟩ `trimarea`

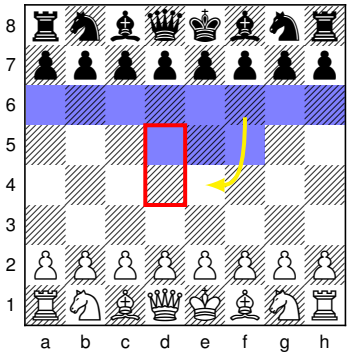
This sets and activates an trimarea. If you omit a value printarea is used. If you don't give an area but use the keyword `empty` everything following the key will be trimmed. If you want to disable trimming use `trimarea=\board` (or `trim=false`). As a default the trimarea is set to the print area (and so activated).

The value of `trimarea` will affect all *following* objects drawn with the `pgfstyle` and the related `mark...` and `back...` keys.

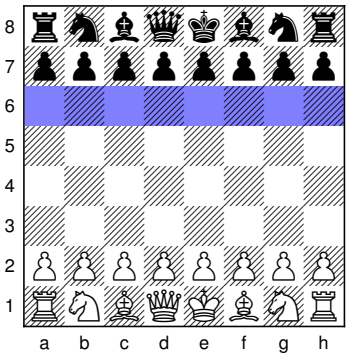
Field related objects will be drawn if the field lies in the area. Region related objects will be drawn if the complete region lies in the area. Move related objects will be drawn if the complete move lies in the area.

```
\newchessgame
\setchessboard{pgfstyle=color,
          color=blue!50,
          backrank=6,
          trimarea=\myarea,
          backrank=5,
          pgfstyle=border,
          color=red,
          markregion=d4-d5,
          markregion=c3-d5,%will disappear
          pgfstyle=knightmove,
          color=yellow,
          markmove=f6-e4,
          markmove=f7-d5%will disappear
          }
\def\myarea{a1-h8}% nothing trimmed
\chessboard

\def\myarea{d4-f6}% some deco will disappear
\chessboard
```



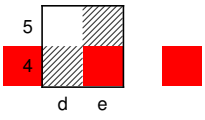


```
\newchessgame
\setchessboard{pgfstyle=color,
         color=blue!50,
         backrank=6,
         trimarea=\myarea,
         backrank=5,
         pgfstyle=border,
         color=red,
         markregion=d4-d5,
         markregion=c3-d5,%will disappear
         pgfstyle=knightmove,
         color=yellow,
         markmove=f6-e4,
         markmove=f7-d5%will disappear
         }
\def\myarea{empty}
\chessboard
```



As shown in the previous examples you can set a manual trimarea. But you should be aware that you can get unexpected results if this area is larger than the printarea:

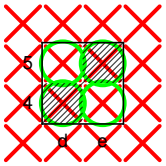
```
\chessboard[printarea=d4-e5,
         trimarea=c4-g4,
         pgfstyle=color,
         trimtocolor=white,
         color=red,
         backboard]
```



captrimtoprint=⟨*true*|*false*⟩	captrimtoprint=true

With this key you can force the *following* trimarea commands to respect the printarea.

```
\chessboard[printarea=d4-e5,
         trimarea=c3-f6,
         captrimtoprint,
         pgfstyle=cross,
         color=red,
         backboard,
         trimarea=c3-f6,%now captrimtoprint
                   %has an effect
         pgfstyle=circle,
         color=green,
         backboard]
```

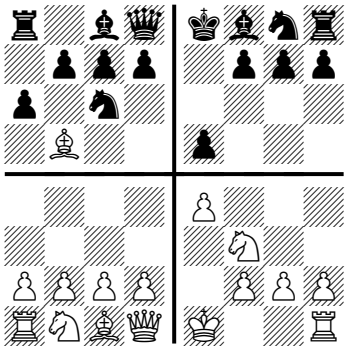


trim=⟨*true*|*false*⟩	trim=true

This is a shorter version of `trimarea`. It can only be used to set the trimarea to printarea (true) or the complete board (false).

```
\newchessgame
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5 a6}
\setchessboard{linewidth=0.1em,border=false,
   label=false,showmover=false}
\chessboard[printarea=a5-d8,
  padding=0.25em,
  trim=false,
  shortenend=-1ex,
  pgfstyle=rightborder,
  markregion=d1-d8,
  padding=-0.25em,
  pgfstyle=topborder,
  markrank=4,
  marginrightwidth=0.5em,
  marginbottomwidth=0.5em]%
\chessboard[printarea=e5-h8,
  marginleftwidth=0em,
  marginbottomwidth=0.5em]\\%
\chessboard[printarea=a1-d4,
  margintopwidth=0em,
  marginrightwidth=0.5em]%
\chessboard[printarea=e1-h4,
  margintopwidth=0em,
  marginleftwidth=0em]%
```

1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6

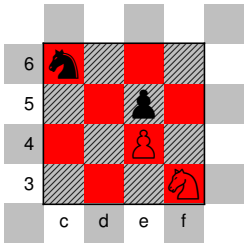


Trimming to the white or black fields

`trimtocolor=`⟨*white*|*black*|*false*⟩ `trimtocolor`

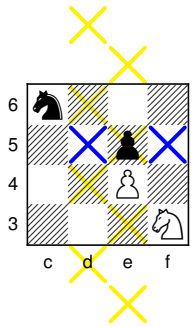
The key affects only the field related pgf-objects. Regions and moves will ignore it. The key `applycolor` from previous versions has now a similar effect[8]. This means that the key has changed its behaviour: it will now affect also the foreground picture!

```
\chessboard[printarea=c3-f6,
         trim, %not really needed, default
         pgfstyle=color,
         trimtocolor=white,
         color=red,
         backboard,
         trimarea=b2-g7,
         trimtocolor=black,
         color=gray!50,
         backboard]
```

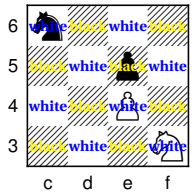


[8] I couldn't retain the older behaviour due to the changes made in the code.

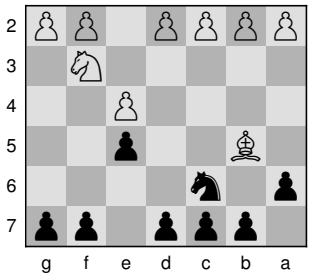
```
\chessboard[printarea=f3-c6,
         padding=.3ex,
         trimtocolor=white,
         color=blue,
         pgfstyle=cross,
         backranks=5,
         color=yellow,
         trimtocolor=black,
         trim=false,
         backfiles={d,e}]
```




```
\chessboard[padding=.3ex,
         printarea=f3-c6,
         trimtocolor=white,
         color=blue,
         pgfstyle=text,
         text=\tiny\bfseries white,
         markboard,
         color=yellow,
         trimtocolor=black,
         text=\tiny\bfseries black,
         markboard]
```




```
\chessboard[inverse, printarea=a2-g7,
         boardfontencoding=LSB2,
         color=lightgray!50,
         trimtocolor=white,
         colorbackboard,
         color=gray!60,
         trimtocolor=black,
         colorbackboard]
```



6.2 Using a graphic as background

It isn't really difficult to use a graphic as background of the board. You must use a boardfont encoding that hasn't fields and fieldmasks. And then you must use trial and error to find out which scalefactor and which viewport is fine. There is absolutly no way to do this automatically as the border of the board in the graphic is unknown. If you change the board size or the size of the left margin, you will have to adjust the values.

```
\newchessgame
\includegraphics
 [scale=0.477,viewport=-20 17 -20 17]
 {brett}%
\chessboard[showmover=false,
          border=false,
          label=false,
          boardfontencoding=LSBC5,
          boardfontsize=20pt,
          whitepiececolor=brown!70,
          blackpiececolor=black,
          setfontcolors]
```

6.3 Using the commands of the package skak

When used with the option ps the package skak puts in the middle of the right bottom field (that is either h1 or a8) a postscript node with the name BM. The highlighting commands of the package skak use this node.

To be able to use the commands also with `\chessboard` you must load the package skak with the option ps.

As arguments in the commands of the package skak you can only use the fields of a 8×8-board.

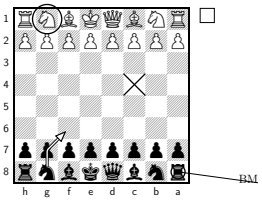
`psset=`⟨*true*|*false*⟩ `psset` `false`

This key sets the node BM at the place where the package skak expect it, *and it sets the psunits to the size of the board.* This can affect other pspictures as the changes are made after the board and so aren't local to the board (the commands of the package skak must have a fighting chance to get the correct values).

`psskak=`⟨*true*|*false*⟩ `psskak` `false`

This key redefines some internal commands of the package skak. I had to do it in order to get the highlighting commands to work. I don't think that it will destroy anything but didn't want to activate the changes unasked. The changes are also *not* local to the board (but local to the group).

```
\documentclass[parskip]{scrartcl}
\usepackage[T1]{fontenc}
\usepackage[ps]{skak}
\usepackage{chessboard}

\pagestyle{empty}
\begin{document}
\newchessgame
\chessboard[psskak,psset,pgf=false,
         inverse]
\ifpdf\else
\highlight[X]{c4}
\highlight[o]{a8}
\highlight[O]{g1}
\printknightmove{g8}{f6}
BM\pnode{A}\ncline{->}{A}{BM}
\fi
\end{document}
```



6.4 Intelligent highlighting

\chessboard doesn't know anything about chess rules. It can't do things like "show me the possible moves of the rock", "show me the allowed moves" or "highlight the last move". But as the examples show it accepts commands as arguments in the highlighting keys, so it can use the intelligence of external commands or applications. E.g. highlighting of the last move would be easy if the package skak would somehow save the last from-field and the last to-field – something it doesn't do now.

7 Extending the game

7.1 Adding new pieces

bDefineNewPiece With this command you can add new pieces that can be used on the board.

\cbDefineNewPiece[⟨*game*⟩]{⟨*color*⟩}{⟨*char*⟩}{⟨*on white*⟩}{⟨*on black*⟩}

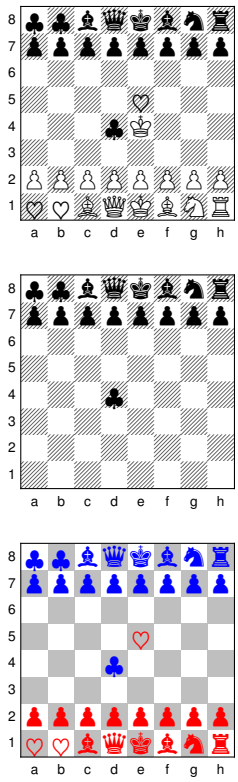
The optional argument ⟨*game*⟩ has not much use yet as there is only one sensible value `skak` which is the default anyway. So leave it out.

⟨*color*⟩ should by either `white` or `black`, it will put the character in the list used by e.g. the key `hidewhite`.

⟨*char*⟩ is the character used in FEN and the other input arguments. Use a uppercase char for white pieces and a lowercase char for the black pieces or the keys like `addwhite` won't work correctly.

⟨*on white*⟩ and ⟨*on black*⟩ should contain the commands needed to print the piece on white and black fields. The symbols should be scalable. So you should either use symbols from a font or use some resizing command from the graphicx package.

```
\makeatletter
\cbDefineNewPiece{white}{C}
  {\raisebox{\depth}{\cfss@whitepiececolor
    $\heartsuit$}}
  {\BlackEmptySquare%
   \makebox[0pt][r]{\cfss@whitepiececolor
    \raisebox{\depth}{%
     \makebox[1em]{$\heartsuit$}}}}
\cbDefineNewPiece{black}{c}
  {\raisebox{\depth}{\cfss@blackpiececolor
    $\clubsuit$}}
  {\BlackEmptySquare%
   \makebox[0pt][r]{\cfss@blackpiececolor
    \raisebox{\depth}{%
     \makebox[1em]{$\clubsuit$}}}}
\makeatother
\setchessboard{tinyboard,
  addpieces={cd4,Ce5},addwhite={ca1,Cb1},
  addblack={ca8,Cb8}}
\chessboard[addpieces={cd4,Ce5, Ke4}]
\chessboard[hidewhite]
\chessboard[boardfontencoding=LSB3,
  whitepiececolor=red,blackpiececolor=blue,
  setfontcolors,color=lightgray,
  trimtocolor=black, colorbackboard]
```



7.2 Using \chessboard for other games

I have tried to prepare `\chessboard` for other games. A lot of internal commands can be changed simply by changing the "name of the game". Up to now there isn't another game, so I don't know if this will really work.

It should be easy to make boards for checkers if I find a font somewhere. Chinese chess is a bit more complicated as one would have to decide how to make the background lines. Go is more complicated as the pieces are sometimes numbered, I don't know yet if I could

implement a good input handling – but as there is a quite good package named igo I don't think I will really look into it.

The main problem in any cases is the missing partner: The package chessboard isn't "intelligent". As already mentioned above it doesn't know anything about the rules of chess. To be able to show "the running game" it needs a partner that does the calculation.

8 Compability with other packages

8.1 skak

The package chessboard works fine with the package skak. You can use `\chessboard` everywhere instead of `\showboard` etc.

8.2 texmate

The new texmate version use the package skak to follow the game. There is no problem to use `\chessboard` where the manual suggest `\showboard`. texmate also defines its own diagram commands. They use internaly `\showboard`. I haven't made much tests until now, but it looks like after a simple `\let\showboard\chessboard` you can use `\chessboard` instead. The main problem is that you can't use the argument of `\chessboard`. You will have to set the board properties with `\setchessboard`.

8.3 beamer

I wouldn't have thought it, but I had no problem to use `\chessboard` with beamer.

The following listing shows an example. If your pdf-reader can handle annotations you can see the result in the attached file:

```
\documentclass{beamer}
\usepackage[T1]{fontenc}
\usepackage{chessboard, skak}
\begin{document}

\begin{frame}
\newchessgame
\only<1->{\setchessboard{hideall,showmover=false}}
\only<2->{\setchessboard{showpieces={p,P}}}
\only<3->{\setchessboard{showpieces={K,k}}}
\only<4->{\setchessboard{showpieces={Q,q}}}
```

```
\only<5->{\setchessboard{showpieces={R,r}}}
\only<6->{\setchessboard{showpieces={B,b}}}
\only<7->{\setchessboard{showpieces={N,n}}}
\chessboard
\end{frame}

\begin{frame}
\newchessgame\setchessboard{markstyle=border,color=blue}
\only<1>{\mainline{1.e4}}
\only<1>{\setchessboard{markfields=e4}}
\only<2>{\hidemoves{1.e4}\mainline{1... e5}}
\only<2>{\setchessboard{markfields=e5}}
\only<3>{\hidemoves{1.e4 e5}\mainline{2. Nf3}}
\only<3>{\setchessboard{markfields=f3}}
\only<4>{\hidemoves{1.e4 e5 2.Nf3}\mainline{2... Nc6}}
\only<4>{\setchessboard{markfields=c6}}

\chessboard
\end{frame}
\end{document}
```

8.4 animate

It also possible to use animate to get “moving” boards (you need a recent pdf-viewer to see the effect). But it is quite tiresome to input all the moves. That’s why I wrote the package xskak. It offers a possibility to store all positions of a game with the package skak and then to loop through them. For an example please read the documentation of xskak. Here an example of the more tiresome input:

```
\newchessgame
\unitlength1pt
\newcommand\currentboard{%
 \begin{picture}(150,150)
 \put(10,10){\chessboard}
 \end{picture}}
\begin{animateinline}[autoplay,loop]{1}%
\currentboard
\newframe\hidemoves{1.e4}%
\currentboard
\newframe\hidemoves{1... e5}%
\currentboard
\newframe\hidemoves{2. Nf3}%
\currentboard
```

```
\newframe\hidemoves{2... Nc6}%
\currentboard
\newframe\hidemoves{3.Bb5}%
\currentboard
\newframe\hidemoves{3... a6}%
\currentboard
\end{animateinline}
```

Index

A

addblack 13
addfen 13
addfontcolors 29
addpgf 58
addpieces 12
addwhite 13
applycolor (obsolete) 67
arrow 56

B

backarea 45
backareas 45
backboard 45
backclip 63
backcornerarc 58
backfield 45
backfields 45
backfiles 45
backlinewidth 57
backmove 47
backmoves 47
backrank 45
backranks 45
backregion 46
backregions 46
backstyle 48
blackfieldcolor 28
blackfieldmaskcolor 28
blackonblackpiecemaskcolor 28
blackonwhitepiecemaskcolor 28
blackpiececolor 29
blackpiecemaskcolor 28
\board – usage 8
boardfontencoding 24
boardfontfamily 23
boardfontseries 24
boardfontsize 23
border 22
border (style) 49
  bottomborder 49
  bottomleftborder 49
  bottomlefttopborder 49
  leftborder 49
  lefttopborder 49
  lefttoprightborder 49
  rightborder 49
  rightbottomborder 49
  rightbottomleftborder 49
  topborder 49
  toprightborder 49
  toprightbottomborder 49
borderbottom 22
borderbottomcolor 22
borderbottomwidth 21
bordercolor 22
borderleft 21
borderleftcolor 22
borderleftwidth 21
borderright 22
borderrightcolor 22
borderrightwidth 21
bordertop 22
bordertopcolor 22
bordertopwidth 21
borderwidth 21

C

captrimtoprint 66
castling 16
\cbDefineLanguage – usage 18
\cbDefineMarkFieldStyle (obsolete) . . . . . *see* \cbDefinePgfFieldStyle
\cbDefineMarkMoveStyle (obsolete) . . . . . *see* \cbDefinePgfMoveStyle
\cbDefineMarkRegionStyle (obsolete) . . . . . *see* \cbDefinePgfRegionStyle
\cbDefineMoverStyle – usage 37
\cbDefineNewPiece – usage 70
\cbDefinePgfFieldStyle – usage 48
\cbDefinePgfMoveStyle – usage 48
\cbDefinePgfRegionStyle – usage 48
\cbDefineTranslation – usage 18
\cblist⟨*piece*⟩ 16
\chessboard – usage 2
circle (style) 50
  fill (option) 50
cleararea 12
clearareas 12
clearbackgroundpgf (obsolete) *see* clearbackpgf
clearbackpgf 62
clearboard 12
clearfield 12
clearfields 12
clearfile 12
clearfiles 12
clearfontcolors 28
clearforegroundpgf (obsolete) *see* clearmarkpgf
clearmarkpgf 62
clearpgf 62
clearrank 12
clearranks 7, 12
clip 63
  bottom (subkey) 63
  false (subkey) 63
  left (subkey) 63
  right (subkey) 63
  top (subkey) 63
  true (subkey) 63
color 54
color (style) 52
colorbackarea 60
colorbackareas 60
colorbackboard 60
colorbackfield 60
colorbackfields 60
colorbackfile 60
colorbackfiles 60
colorbackrank 60
colorbackranks 60
colorblack (obsolete) 29
colorblackbackfields 60
coloremph 30
coloremphstyle 30
colorpieces (obsolete) 31
colorwhite (obsolete) 29
colorwhitebackfields 60
cornerarc 58
cross (style) 50
\currentbk 51
\currentbq 51
\currentwk 51
\currentwq 51
curvemove (style) 53
  arrow (key) 54
  clockwise (key) 53
  linewidth (key) 54
  pgfarrow (key) 54
  pgflinewidth (key) 54
  pgfshortenend (key) 54
  pgfshortenstart (key) 54
  shortenend (key) 54
  shortenstart (key) 54
  style (key) 54
  x1 (key) 53
  x2 (key) 53
  y1 (key) 53
  y2 (key) 53

E

empharea 31
emphareas 31
emphboard 31
emphfield 31
emphfields 31
emphfile 31
emphfiles 31
emphrank 31
emphranks 31
emphstyle 30

enpassant 16

F

fieldcolor 28
fieldmaskcolor 28
fillarea 14
fillopacity 55
fontsize 23
fullmove 16

G

getpiecelists 16

H

halfmove 16
hborder 22
hbordercolor 22
hborderwidth 21
hideall 40
hidearea 40
hideareas 40
hideblack 40
hideboard 40
hidefield 40
hidefields 40
hidefile 40
hidefiles 40
hidepiece 40
hidepieces 40
hiderank 40
hideranks 40
hidewhite 40
hlabel 33
hlabelfont 34
hlabelformat 35
hlabellift 33
hlabelwidth 33
hmargin 20
hmarginwidth 20

I

inverse 39

K

knightmove (style) 53

L

label 33
labelbottom 33
labelbottomfont 34
labelbottomformat 35
labelbottomlift 34
labelfont 34
labelfontsize 34
labelformat 35
labelleft 33
labelleftfont 34
labelleftformat 34
labelleftlift 33
labelleftwidth 33
labelright 33
labelrightfont 34
labelrightformat 34
labelrightlift 33
labelrightwidth 33
labeltop 33
labeltopfont 34
labeltopformat 35
labeltoplift 33
language 18
largeboard 23
linewidth 57
loadfen 17

M

margin 20
marginbottom 20
marginbottomwidth 20
marginleft 20
marginleftwidth 20
marginright 20
marginrightwidth 20
margintop 20
margintopwidth 20
marginwidth 20
markarea 45
markareas 45
markboard 45
markclip 63
markfield 45
markfields 45
markfiles 45
marklinewidth 57
markmove 47
markmoves 47

markrank 45
markranks 45
markregion 46
markregions 46
markstyle 48
maxfield 10
mover 15, 35
moverbottomlift 36
moverbottomshift 37
moverlift 36
movershift 37
moversize 36
moverstyle 37
movertoplift 36
movertopshift 37

N

normalboard 23

O

onblackpiecemaskcolor 28
onwhitepiecemaskcolor 28
opacity 54

P

padding 55
pgf 43
pgfarrow 56
pgfborder 59
pgfcolor 54
pgfcornerarc 58
pgffillopacity 55
pgflinewidth 57
pgfopacity 55
pgfpadding 55
pgfshorten 56
pgfshortenend 56
pgfshortenstart 56
pgfstrokeopacity 55
pgfstyle 48
piececolor 29
piecemaskcolor 28
print 38
printarea 39
\printarea – usage 8
psset 69
psskak 69

R

restorefen 17

S

savefen 15
scope 58
setblack 13
\setchessboard – usage 6
setfen 13
setfontcolors 29
setpieces 12
setwhite 13
shorten 56
shortenend 56
shortenstart 56
showall 42
showarea 41
showareas 41
showblack 42
showboard 41
showfield 41
showfields 41
showfile 41
showfiles 41
showmover 35
showpiece 42
showpieces 42
showrank 41
showranks 41
showwhite 42
smallboard 23
startfen 14
startfill 14
startprint 38
startstore 15
stopfill 14
stopprint 39
stopstore 15
storearea 15
\storechessboardstyle – usage 7
storefen 15
straightmove (style) 53
strokeopacity 55
style 7

T

text (style) 51

options 51
tinyboard 23
trim 66
trimarea 65
trimtocolor 67

U

usepgf 59

V

vborder 22
vbordercolor 22
vborderwidth 21
vlabel 33
vlabelfont 34
vlabelformat 35
vlabellift 34
vmargin 20
vmarginwidth 20

W

whitefieldcolor 28
whitefieldmaskcolor 28
whiteonblackpiecemaskcolor 28
whiteonwhitepiecemaskcolor 28
whitepiececolor 29
whitepiecemaskcolor 28

Z

zero 6, 11