

xskak: An extension to the package skak

Ulrike Fischer

January 2, 2015

Contents

1. Changes	2
2. Introduction	3
2.1. Warnings	5
2.2. Some history	6
2.3. Bugs and errors	7
2.4. Requirements	7
2.5. Installation	7
2.6. Loading and package options	7
2.7. Adaption to the package skak version 1.5	7
2.8. The example	8
3. Improved parsing of comments	9
4. The core of the package xskak: Saving games.	10
4.1. Initialization of the game	11
4.2. Continuing a game	15
4.3. The stored informations	17
4.3.1. Essential game related informations	17
4.3.2. Optional game informations: PGN-infos	19
4.3.3. The move related data	20
4.4. Exporting the games	23
4.5. The inner technique: how the data are stored	24
5. Retrieving the stored informations	27
5.1. Setting the variables	27
5.2. Getting the game informations	28
5.3. Getting the move informations	29
5.4. Using the data with \chessboard	30

5.5. Labeling a game	31
5.5.1. Retrieving the tagged values with \xskakget	32
5.5.2. The ref-keys	33
6. Looping	34
7. Printing	38
7.1. \longmoves revisitated	38
7.2. An user interface for styles	39
7.2.1. Defining style items	41
7.2.2. Defining style	42
7.3. Using the styles	46
7.4. The predefined styles	47
7.5. The new xskak printing command	47
7.6. Game titles and chessboard captions	48
8. PGN2LTX or How to sort the input	48
9. Compability issues	52
9.1. xskak and texmate	52
9.2. xskak and beamer	52
Index	54

1. Changes

2015-01-02 (Version 1.4)] Corrected a bug: After a promotion the promotionpiece and the fen position were wrong.

2014-04-19 (Version 1.3)] Corrected a bug: enpassant test didn't check the piece and so other figures than a pawn could make an enpassant move.

2012-08-24 (Version 1.2b) Corrected a small bug – \printchessgame doesn't wrap around a wrapfigure.

2012-08-24 (Version 1.2a) Corrected a small bug – \hidemoves printed comment signs.

2008-10-10 (Version 1.2) Made some changes to make the package xskak compatible to the version 1.5 of the package skak. See below for details.

Changed Code: As \longmoves doesn't work with \variation anyway \variation now use always \shortmoves.

Bug correction: the package skak forgets to store comments after castling moves. Code to correct this has been added to the package xskak.

Bug correction: the package xskak didn't print check and mate symbol after castling moves due to wrong position of braces.

2008-07-28 (Version 1.1) Bug correction: Added the code for the promotion piece to the SAN and LAN notation.

Bug correction: Remove some spurious spaces.

2. Introduction

The package xskak¹ is an extension for the package skak– and as the package xskak is an extension (and not a replacement) the main work is still done by the package skak so please read first the documentation of the package skak as I'm not explaining the skak commands in this documentation.

The package skak is a package from Torben Hoffmann which can parse and print chess games given in (a subset of) the PGN notation.

The main features of xskak are:

- xskak can handle an extended set of PGN.
- xskak can parse and store an arbitrary number of chess games in PGN. The games can be exported and imported and all positions and moves can be retrieved at anytime.
- xskak can handle an arbitrary number of printing styles to distinguish the different variations levels.

Extended PGN handling

When handling a PGN chess parsers at first have to split the notation in different chunks. I will call this step the *PGN parser*. As an example

```
15.Ne5!+ $3 {This breaks the black defense} (15.Re1? Kg8!)
```

contains as chunks a *move number* (15.), a *move* (Ne5), *short move comments* (!+), a *NAG* (\$3), a *comment* ({This breaks the black defense}) and a *variation* ((15.Re1? Kg8!)).

Neither skak nor xskak can handle all legal PGN's. The package skak can only handle PGN's which contains move numbers, moves and short comments. xskak can additionally handle NAG's and a special form of comments. None of the packages can currently parse PGN's with variations.






¹License: ~~TeX~~ Project Public License

Storing moves

After the splitting of the PGN the chunks are processed further. Depending on the type of the chunk different actions are taken. The most important subprocess is the *move parser*. It takes the move, retrieves from somewhere the current (FEN-)position of the game and tries to figure out what the move is exactly doing.

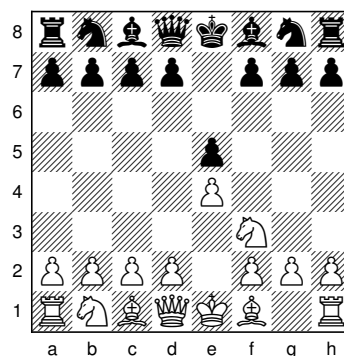
In xskak only the commands `\mainline` and `\hidemoves` calls the move parser, `\variation` uses only the PGN parser.

The main difference between xskak and skak is how they handle the informations gathered by the move parser: The package skak uses the results of the move parser mainly to update the board and after the move throws almost all other informations away – including previous positions. the package xskak saves all informations about the moves for later use. Here a simple example:

1. e4 e5 2. f3 c6 3. b5 a6 4. a4 f6

```
\newchessgame
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6}

\xskakset{moveid=2w}%
\chessboard[setfen=\xskakget{nextfen}][\lex]
Position after 2.\,\xskakget{lan}
```



Position after 2. g1–f3

So in short you start a game with `\newchessgame`, you enter moves with `\mainline` (or `\hidemoves`), you set the move for which you want to retrieve data with `\xskakset`, and then you retrieve the data with `\xskakget`.

The storing of the moves and positions is not restricted to the main game. You can store and retrieve as much variation levels you want.

Printing styles

The following example shows how to use different styles for different variation levels. But it also shows the problems: It can be quite tiresome to get a good printing result. You must pay attention to the levels and the spaces after commands, and for each variation you have to start a new game and resume older games if you go back to them.

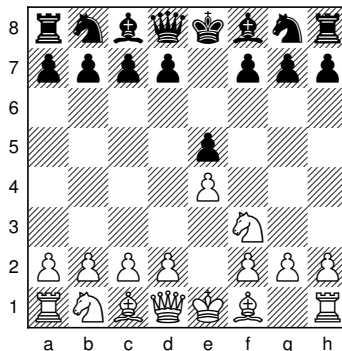
```

\longmoves \xskakset{style=UF}
\newchessgame[id=main]
\mainline{1.e4 e5}
%
(Alternatives are
\newchessgame[newvar=main,id=var1]%
\mainline[invar]{1... d5} which is answered
  by
\mainline{2. exd5 Qxd5}
%
  \newchessgame[newvar=var1,id=var1-1]
  (\mainline[invar]{2... Nf6})
  %
or
\newchessgame[newvar=main,id=var2]
\mainline[outvar]{1... e6 2. d4})
%
\resumechessgame[id=main]%
\mainline[outvar]{2.Nf3}

\chessboard

```

1. e2–e4 e7–e5 (Alternatives are 1... d7–d5 which is answered by 2. e4×d5 ♖d8×d5 (2... ♘g8–f6) or 1... e7–e6 2. d2–d4) **2. ♘g1–f3**



In the following sections I will describe the finer points. This includes

- how to handle more than one game at a time,
- how to retrieve the data faster,
- what data you can retrieve,
- how to loop through a game,
- how to export the data,
- how to improve the printing of the games,
- and some other changes/additions I made to skak.

2.1. Warnings

xskak and skak

The package xskak doesn't only define some new commands, it also changes some quite central internal commands of the package skak. This means

- it can break standard commands of the package skak,
- it will possibly not work together with future versions of the package skak,
- loading xskak.sty can lead to problems with packages like texmate which use internally skak.

Local and global commands

In the package chessboard almost every setting is local to the board. This was a quite natural decision as a board is a confined space. In xskak the situation is quite different: As a start skak already sets and saves some informations globally and the data saved by xskak during the parsing must be saved globally or they wouldn't never escape the `\mainline`. And secondly, as I wanted to use beamer and animate a lot of informations must be preserved from one frame/picture to the next. So I decided to use in most cases global commands unless – like in the cases of styles – I could see a real advantage in keeping a setting local.

So please be aware that settings made e.g. in a figure environment can affect the following text.

2.2. Some history

While the commands `\mainline` and `\hidemoves` parse a chess game the package skak updates an internal board and so can always show the current position. Sadly a lot of the informations gathered during the parsing are thrown away after skak has updated the board and has go on to the next move and so can not be used by the user. E.g. there is no way to get the field a piece came from or to store easily all positions of a game.

Some years ago I tried to find out how to print a game in long algebraic notation and I realized quite fast that I need some of these lost data to be able to do it, and that it would be necessary to patch some internal commands of the package skak to save the data. At the same time I also tried to insert some code that allows the parsing of standard PGN-comment signs. Due to lack of time the whole project didn't got very far but I had a working beta. Then a new version of the package skak came. It had even a – sadly not fully working (as can be seen in example 1) – command `\longmoves` (and the counterpart `\shortmoves`) to switch to the long algebraic notation. But I hadn't the time to test and adjust my package.

Example 1: Wrong pawn moves in the longmoves implementation of the package skak version 1.4

```
\documentclass[12pt]{article}
\usepackage{skak14}%= skak version 1.4
\begin{document}
\newgame\longmoves
\mainline{1.e4 e5 2.Nf3 Nf6}

\variation{2... d5}
\end{document}
```

```
1 e7-e4 e2-e5 2 ♖g1-f3 ♗g8-f6
2...g8-d5
```

Then in 2006 I wrote the package chessboard and during the tests also tried to make animated pdf's which show all positions of a game. That worked fine but the input was so tiresome that I restarted the xskak-project: Like in the case of the long algebraic notation the main problem was the loss of previous positions.

2.3. Bugs and errors

I'm quite sure that they are bugs and errors in the package.

If you have questions ask them in the newsgroups `comp.text.tex`, `de.comp.text.tex` or `fr.comp.text.tex` or at <http://tex.stackexchange.com/>. 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 `chess@nililand.de`. A bugreport should contain a complete, running, *minimal* example and the log-file of the pdf \LaTeX run (that means the engine that makes a pdf).

2.4. Requirements

The package `xskak` uses some primitives of e \TeX . It needs version 1.4 or version 1.5 of the package `skak`, a recent version of the package `chessfss` (chess font selection) and `xkeyval` (*key=value*-syntax), version 1.5 of `chessboard` (which in turns needs `pgf`) and `xifthen`.

2.5. Installation

Run `xskak.ins` through \LaTeX and then put the `*.sty` where \LaTeX finds them. (`<texmf>/tex/latex/xskak/` is a good place.) Update the filename database.

2.6. Loading and package options

Loading of the package is done with `\usepackage{xskak}`. Currently the package has only one option: `nopdfmatch`. The option will disable the use of the experimental `\pdfmatch` used in the processing of the style levels (see section 7.3). `xskak` will load the package `skak` without option (and won't pass options to it). So, if you want to load `skak` with options, you must do it before loading the package `xskak`.

2.7. Adaption to the package `skak` version 1.5

Above I warned that the package `xskak` can break with newer versions of the package `skak`. Some days ago the version 1.5 of `skak` appeared and promptly a lot of things broke so I had to adapt and change my code. Version 1.2 of the package `xskak` will hopefully work with both versions of the package `skak`.

In version 1.5 of the package `skak` some fundamental things have changed. At first `\longmoves` works now correctly, even as you can see for the `\variation` command:

Example 2: The new longmoves implementation of the package skak

```
\documentclass[12pt]{article}
\usepackage{skak15}%=skak version 1.5
\begin{document}
\newgame\longmoves
\mainline{1.e4 e5 2.Nf3 Nf6}

\variation{2... d5}
\end{document}
```

```
1 e2-e4 e7-e5 2 ♘g1-f3 ♘g8-f6
2...d7-d5
```

The more important change (from the point of my package) is that `\variation` now parse the moves too (that's why it can show the long algebraic notation). To achieve this the author of the package skak, Torben Hoffman, stores now the position before a move so that he can go back to this position and start the variation from there.

The main problem with this approach is that it imposes restrictions on the move numbers of variations: All variations must start with the move number of the last move in a `\mainline` command. So separating two moves of a variation by some text is no longer possible:

```
\documentclass{article}
\usepackage{skak15} %=skak version 1.5
\begin{document}
\newgame
\mainline{1.e4 e5 2. Nf3}
\variation{2. d3 d6} %works
\variation{2. c3} %works
and then
\variation{2...c6} %error
\end{document}
```

To tell the truth, I'm not sure that this a very good idea. In any case: the package xskak overwrites the responsible definitions of the package skak. With the package xskak `\variation` currently doesn't parse games (and so also can't show moves in long algebraic notation – if you want this you will have to start with the commands described later in this documentation a new game with another id). So the above example works fine with the package xskak.

2.8. The example

Almost all examples use as starting point a new game. But in some cases the following position is used, which allows to show almost all special cases (like enpassant, promotion, capture, castling, ambiguous moves). The position is stored under the name “example” and

so can be retrieved e.g. with the key `restorefen=example` from `chessboard` or the command `\restoregame{example}` of the package `skak`²:

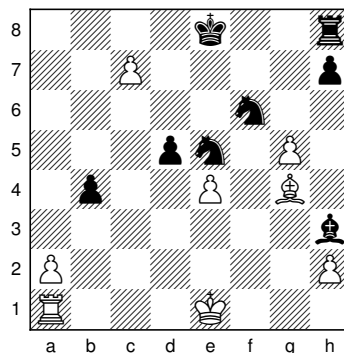
Example 3: The standard example position

4k2r/2P4p/5n2/3pn1P1/1p2P1B1/7b/P6P/R3K3 w Q- - 0 1

```
\newchessgame[id=example,
  castling=Q,
  setwhite={pa2,pc7,pg5,ph2,pe4,
    ke1,ra1,bg4},
  addblack={pb4,ph7,pd5,
    ke8,rh8,bh3,nf6,ne5},
  storefen=example]%

\mbox{\Xskakexampleinitfen}

\chessboard
```



3. Improved parsing of comments

The package `skak` can handle comments in the input if they follow directly the move, if the commands are robust and if the spaces that separate moves and numbers are not swallowed by a command.

Example 4: comments in the package `skak`

```
\documentclass[12pt]{article}
\usepackage{skak15} %=skak version 1.5
\begin{document}
\newgame
\newcommand\mycomment{ What else?}
\mainline{1.e4!\wbetter{}\mycomment{} e5
  2.Nf3 Nc6}
\end{document}
```

1 e4! \pm What else? e5 2 $\text{\textcircled{N}}\text{f3}$ $\text{\textcircled{N}}\text{c6}$

This is quite okay for a few small comments in a small number of games, but a bit tiresome if you want to handle a lot of games that you have annotated with a chess application. The PGN export of such games uses NAG's for short comments and curly braces for longer comments.

With the package `xskak` the commands `\mainline`, `\hidemoves` and `\variation` now can parse NAG's and longer comments if the input meet the following conditions:

- There must be spaces between the moves, each NAG and the comments.

²One could also use key tag to store the position and key `refen` to retrieve it.

- Text comments must be written as `\xskakcomment{<Text>}`³

While parsing the comments the spaces before and between them will be swallowed. If you want a space you must add it to the text. The NAG's (\$1, \$2 etc) are translated following the definitions in `xskak-nagdef.sty`. The list there is neither complete nor definite. For quite a lot NAG's I hadn't the faintest idea what do. I'm open for suggestions/corrections.

There exist two "fake" NAG's: \$d and \$D. If used in a `\mainline` or `\hidemoves` both will save the identification of the current move in an internal list and print whatever they have been defined to. (Currently \$D will print `\chessdiagramname` which is defined as `\ (Diagram)\ , $d is empty`). This can be used to refer to diagrams printed later.⁴

Example 5: NAG's, comments and diagram references in the input

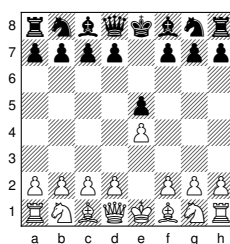
```
\newchessgame[id=GameA]
\mainline{1.e4! $2 e5 $D 2.Nf3 Nf6
  \xskakcomment{ What's that? }}

\renewcommand\xskakcomment[1]{\itshape #1}%
\mainline{3. Nxe5
  \xskakcomment{ I took a pawn! } $D }
\variation[invar]{3. Nc3 $4
  \xskakcomment{ (boring) }}

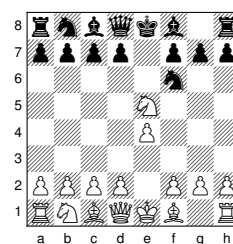
\makeatletter
\@for\mymoveid:=\XskakGameAdiagramlist\do{%
  \xskakset{moveid=\mymoveid}%
  \begin{tabular}{c}
  \chessboard[tinyboard,
    setfen=\xskakget{nextfen}]\
  After move \xskakget{opennr}\xskakget{lan}
  \end{tabular} }
```

1. e4!? e5 (Diagram) 2. ♖f3 ♖f6 What's that?

3. ♜×e5 I took a pawn! (Diagram) 3. ♜c3??
(boring)



After move 1... e7–e5



After move 3. ♖f3×e5

4. The core of the package xskak: Saving games.

The package `xskak` redefines some internal commands of the package `skak` to enable `\mainline` and `\hidemoves` to store quite a lot⁵ informations about the moves and the state of the game while parsing the game.

There are two sorts of data: Informations related to the game as a whole (e.g. the names of the players and the start position) and informations related to the moves. Both type of

³Braces are so frequent in T_EX that is simply not possible to identify the comments simply by looking for a brace – at least not without a lot of complicated hacks.

⁴It is not a accidental that there is a similar command "`\toD`" in `texmate`. That's where I got the idea from.

⁵**Attention:** When I say "a lot" I really mean it. Currently about 30 commands for each halfmove is saved plus about 30 commands for each game. A lot of games with individual game identifications can use a lot of memory. But in practice that shouldn't be a problem as it is seldom necessary to use a lot of different game identifications.

informations use a $\langle GameId \rangle$ in the names of the commands to identify the game they are referring to, the move data use additionally a $\langle MoveId \rangle$.

Exkurs 1

It is important to understand one difference between the use of $\langle GameId \rangle$ and $\langle MoveId \rangle$: There is only *one* command that holds the current $\langle GameId \rangle$. The $\langle GameId \rangle$ is a concept specific to the package xskak. The current value $\langle GameId \rangle$ is stored in a command and will be used (and can be changed) by all the commands that store informations *and* by all the commands that retrieve informations. So if you are parsing a game in bits you should be careful to use the same $\langle GameId \rangle$ for each part.

On the other sides there are *three* different places where move informations are set or used: the package skak uses in `\mainline` and `\hidemoves` a counter called “move” and a boolean to keep track of the current move and player. The package xskak has two internal commands which are used to set up the start moves of games and to retrieve the stored move data, and `\chessboard` has two keys that set move informations in FEN’s. This can be sometimes a bit confusing. The package skak will not like it if its move counter is changed while parsing a game, but it is normally not a problem to change the xskak move commands e.g. to retrieve informations about previous moves – as long as you don’t use commands like `\newchessgame` that affects also the skak move counter.

Exkurs 2

In chess we speak quite as often about the last move and the last position than about the next move and the next position: A position can be described as the one *after* the *last* move, the result of a move. But it can also be described as the one *before* the *next* move.

The keys and commands described later reflect this: Given a move number you can access e.g. the position *before* the move through the key word `pastfen` and the position *after* the move through the key word `nextfen`. Given a *tag* – that is a label between two moves, you can refer to the move *before* the tag through keys with the prefix `past` and to the move *after* the tag through keys with the prefix `next`.

In short: `past` looks back and `next` looks forward.

The first position of a game and the first move have the prefix `init`. The last position (after the last move) and the last move have the prefix `last`.

4.1. Initialization of the game

`\newchessgame`

When using the package xskak you should start a new game with `\newchessgame [$\langle key=value \rangle$`

list)] instead of `\newgame`. It will set the identification $\langle GameId \rangle$ of the game, the start position, start move and quite a lot of other variables to sensible values.

In the optional argument you can use the following keys:

id= $\langle GameId \rangle$ This key sets the string that identifies the game and is used in the following parsing commands to store the data. It is possible to use commands and counters e.g. for automatic numbering of the games. If you don't use the key or if you use the key without argument (`\newchessgame[id]`) a default is used. The initial value of this default is `game`, but it can be changed with `\xskakset`. You can use the key `id` in each `\newchessgame` only once.

<code>\newchessgame[white=Bob]% \mainline{1. e4 e5}</code>	
<code>\newchessgame[id=MY,white=Bill]% \mainline{1. e4 e5}</code>	
<code>\newcounter{gamecounter}% \stepcounter{gamecounter}% \xskakset{defaultid=Game\Alph{gamecounter}}%</code>	1. e4 e5
<code>\newchessgame[white=Harry]% \mainline{1. e4 e5}</code>	1. e4 e5
<code>\stepcounter{gamecounter}% \newchessgame[white=Thomas]% \mainline{1. d4 d5}</code>	1. d4 d5
<code>\Xskakgamewhite, \XskakMYwhite, \XskakGameAwhite, \XskakGameBwhite%</code>	Bob, Bill, Harry, Thomas

moveid= $\langle number \rangle$, **player**= $\langle "w" \text{ or } "b" \rangle$, **moveid**= $\langle number + "w" \text{ or } "b" \rangle$ With this keys you can change the start move for the parsing commands `\mainline` and `\hidemoves`. Setting this keys will change the counter and boolean of `skak` *and* the internal commands of `xskak`. The changes are global. Default is `moveid=1w`, this default can be changed with `\xskakset`. Move number and player given through one of the FEN-keys of `\chessboard` will not affect the move counter of `skak`!

<code>\newchessgame \mainline{1. e4}</code>	
<code>\newchessgame[moveid=4w] \mainline{4. e4 e5}</code>	1. e4
<code>\xskakset{defaultmoveid=5b} \newchessgame \mainline{5... c4}</code>	4. e4 e5
<code>\xskakset{defaultmoveid=1w} %back to normal</code>	5... c4

newvar=<GameId> If you use this key `\newchessgame` will use the position and the *<moveid>* before the last move of the game *<GameId>* as start position. If you don't give a value the current meaning of *<GameId>* is used. This can be used to start variations. The new game inherits only the start position of the reference game!

ref...=<tag> The *<GameId>* and the start move can also be set with keys which use values saved with a tag. The keys are described in section 5.5.

keys to set PGN-infos With the keys `result`, `white`, `black`, `whiteelo`, `blackelo`, `site`, `date`, `event` and `round` you can store the content of the PGN-infos. Unused keys are set to default values. These defaults can be changed with `\xskakset`.

With `\xskaknewpgninfo[<default value>]{<keyname>}` you can define a own PGN-info key. This will also define a key `default<name>` to change the default value later.

xskaknewpgninfo

```
\xskaknewpgninfo[Tom]{commentary}
```

```
\newchessgame[white=Harry,date=2007]
\xskakgetgame{white}, \xskakgetgame{date},
\xskakgetgame{event},
\xskakgetgame{commentary}
```

Harry, 2007, , Tom

```
\xskakset{defaultevent=My open}
```

Mller, My open

```
\newchessgame[commentary=Mller]
\xskakgetgame{commentary},
\xskakgetgame{event}
```

using chessboard keys You can use any key that can be used in the optional argument of `\chessboard` from the package `chessboard`. `\newchessgame` will pass all keys to an internal `\chessboard`⁶. `\chessboard` will set a position and give it back as the start position for the game. All chessboard keys are processed normally. You can even print directly the board by using the key `print`. The only exceptions⁷ are the keys `maxfield` and `zero` that change the size of the board – `skak` doesn't like unusual sizes of the board – and the keys `mover` and `fullmove` which are overwritten by the (eventually default) value of key `moveid`. The castling possibilities and the enpassant field are set accordingly the values in the default fen, you only need to set them if you want to use other values. Read the documentation of `chessboard` for more informations.

⁶That's why you will get an error from `\chessboard` if you use an unknown key.

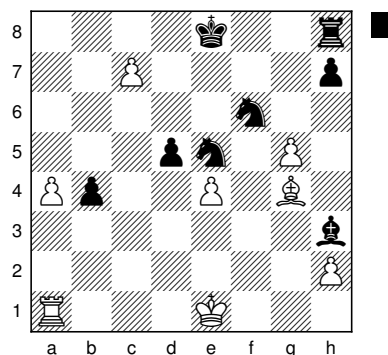
⁷They don't gives error but they will have no effect.

```

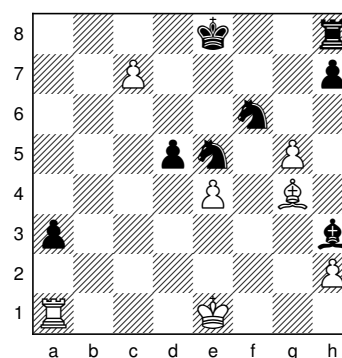
\newchessgame[
  moveid=4b,print,
  showmover,
  mover=w,% has no effect
  castling=Q,enpassant=a3,
  setwhite={pa4,pc7,pg5,ph2,pe4,ke1,ra1,bg4},
  addblack={pb4,ph7,pd5,ke8,rh8,bh3,nf6,ne5}]%

\mainline{4... bxa3}
\chessboard

```



4... bxa3



To use a position of another game as a start position you can use e.g. this sequence of keys (a shorter method that use a label/ref-method will be described in the section 5.5):

```

\newchessgame[id=<old id>, moveid=<old move id>,
  setfen=\xskakget{<nextfen|pastfen>},
  id=<new id>, moveid=<new initmove id>]

```

Attention: A new game is a *new* game. It doesn't inherit moves of another game – it only reuses a position. So if the new and the old game have identical *<GameId>* then the old game will be more or less lost.

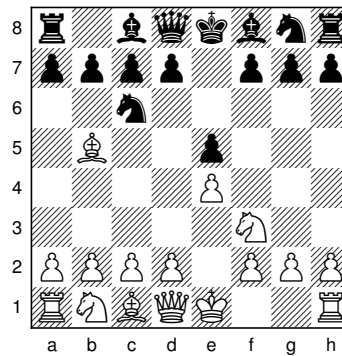
In case that you want to go one half move back and start a new game from there you can use the key newvar:

```

\newchessgame[newvar=<old id>, id=<new id>]

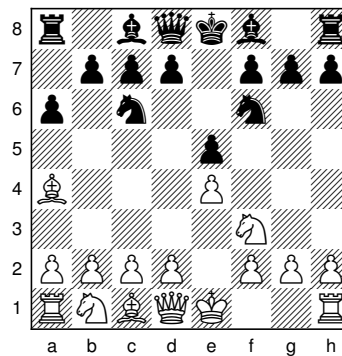
```

1. e4 e5 2. ♘f3 ♘c6 3. ♙b5 a6 4. ♙a4 ♘f6 5. O-O



1... ♘f6

1. e4 e5 2. ♘f3 ♘c6 3. ♙b5 a6 4. ♙a4 ♘f6 5. O-O



5. d3

```
\newchessgame[id=A]%
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4.
  Ba4 Nf6 5. O-O}

\newchessgame[
  id=A,moveid=3w,print,
  setfen=\xskakget{nextfen},
  %fen after move 3w
  id=B,moveid=1b]

\mainline{1... Nf6}

\newchessgame[id=C]
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4.
  Ba4 Nf6 5. O-O}

\newchessgame[print,
  id=C,
  % we retrieve the number of the last move
  % of game C and set it as first number
  % of the new game B at the same time:
  moveid=\xskakgetgame{lastmoveid},
  setfen=\xskakget{pastfen},
  id=B]

\mainline{5. d3}
```

4.2. Continuing a game

resumechessgame

The package skak doesn't have a problem if you split a game in pieces. You can use as many `\mainline` commands as you want. It even doesn't matter if you started the game in another group. But if you mix two games you must first reset the position and the move counter to the end position of the old game before you can continue it. This is possible with `\resumechessgame[⟨key=value list⟩]`. You can use `\chessboard-keys` in the optional argument but all the keys that tries to change the position will have no effect. It is also not possible to store the new moves under a new game identification or to renumber the moves.

The command accepts the following additional keys in the optional argument:

id=⟨GameId⟩ This key identifies the game that should be continued (as a default the currently active identification is used). The key changes `⟨GameId⟩` globally!

moveid=⟨number⟩, player=⟨"w" or "b"⟩, moveid⟨number + "w" or "b"⟩ This keys set the move from which on the game should be restarted. With the keys it is possible to go back and then continue the game along another variation (in this case the moves of

the first variation will be lost). The values describes the *next* move that means the position *before* the move is used. As a default `\resumechessgame` continues the game.

```
\newchessgame[id=main]
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5}
```

That reminds me another game
some years ago where I played

```
\newchessgame[id=exkurs]
\mainline{1. d4 Nf6 2. c4 e6}
\ldots\ lots of text \ldots\
```

Let's continue the second game with --
as a reminder -- the last move of
black

```
\resumechessgame[id=exkurs,moveid=2b]%
\mainline{2... e6 3. Nc3}
```

Let us now go back to the first game\\

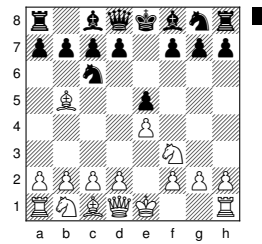
```
\resumechessgame[id=main,print,
    tinyboard,showmover]%
\mainline{3... a6}
```

1. e4 e5 2. ♘f3 ♘c6 3. ♗b5

That reminds me another game some years ago where I played **1. d4 ♘f6 2. c4 e6 ...** lots of text ...

Let's continue the second game with -- as a reminder -- the last move of black **2... e6 3. ♘c3**

Let us now go back to the first game



3... a6

newvar=⟨GameId⟩ This key moves you one halfmove back. This can be used to write down a lot of variations. With the (optional) value of the key you set the ⟨GameId⟩. If you don't give a value the current meaning of ⟨GameId⟩ is used.

1. e4 e5 2. ♘f3 ♗c6 3. ♙b5

1. d4 ♗f6 2. c4 e6

3. ♙c4

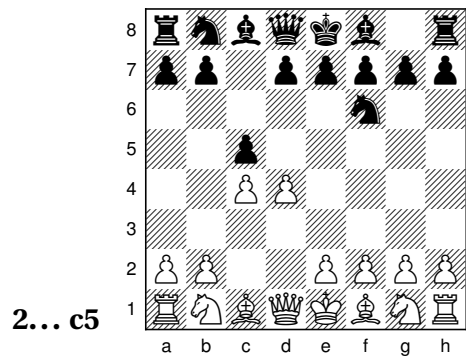
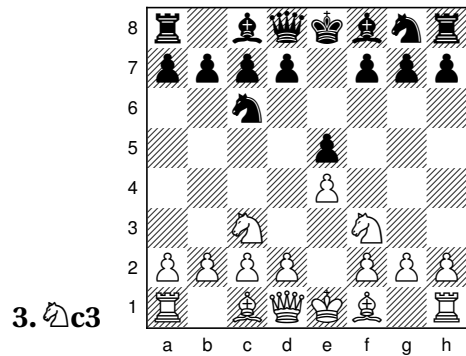
```
\newchessgame[id=main]
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5}
```

```
\newchessgame[id=exkurs]
\mainline{1. d4 Nf6 2. c4 e6}
```

```
\resumechessgame[newvar=main]
\mainline{3. Bc4}
```

```
\resumechessgame[newvar]
\mainline{3. Nc3}
\chessboard
```

```
\xskakset{id=exkurs}
\resumechessgame[newvar]
\mainline{2... c5}
\chessboard
```



ref...=<tag> The *<GameId>* and start move can also be set with this keys which use values saved with a tag. The keys are described in section 5.5.

4.3. The stored informations

The following lists describes all the types of information stored during the parsing of a game. As you will see some data are stored more than once with only slight differences. I did it because it is easier to store the redundant data than to have to convert them later.

There are also commands which contains the *<GameId>* and the *<MoveId>*. That looks a bit senseless as you need to know this values to be able to retrieve them but they allow easy access to the values without having to know or use the internal command names.

4.3.1. Essential game related informations

\Xskak<GameId>initfen The position before the first move. Set by `\newchessgame`.

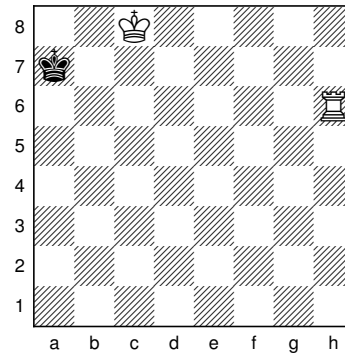
\Xskak<GameId>initplayer “w” or “b”. The color of the next (first) player, (must be identical to the color in the initfen). Also set by \newchessgame.

\Xskak<GameId>initmovenr The number of the first move (must be identical to the full-move number in the initfen). Also set by \newchessgame.

\Xskak<GameId>initmoveid Combination of initmovenr and initplayer.

```
\newchessgame[id=show,
  setfen=2K/k/7R,
  moveid=10b,
  print]

\mbox{\xskakgetgame{initfen}},\
\xskakgetgame{initplayer},
\xskakgetgame{initmovenr},
\xskakgetgame{initmoveid}
```



2K5/k7/7R/8/8/8/8/8 b KQkq - 0 10,
b, 10, 10b

\Xskak<GameId>lastfen The fen *after* the last move, set during parsing.

\Xskak<GameId>lastplayer “w” or “b”. The player which made the last move, set by \newchessgame, by \resumechessgame and during parsing. Identical to color in the fen *before* the last move! *Not* identical to the color in \Xskak<GameId>lastfen!

\Xskak<GameId>lastmovenr The number of the last move, set during parsing. Identical to fullmove in the fen *before* the last move! *Not* necessarily identical to the number in \Xskak<GameId>lastfen!

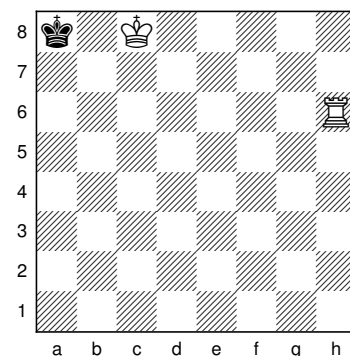
\Xskak<GameId>lastmoveid Combination of lastmovenr and lastplayer.

```
\newchessgame[id=show,
  setfen=2K/k/7R,
  castling={},
  % sets correct castling in the FEN
  moveid=10b]
\mainline{10... Ka8}

\chessboard

\mbox{\xskakgetgame{lastfen}},\
\xskakgetgame{lastplayer},
\xskakgetgame{lastmovenr},
\xskakgetgame{lastmoveid}
```

10... ♔a8



k1K5/8/7R/8/8/8/8/8 w - - 1 11,
b, 10, 10b

\Xskak<GameId>nextplayer “w” or “b”. The player which will make the next move, set by `\newchessgame`, by `\resumechessgame` and during parsing. Identical to the color in `\Xskak<GameId>lastfen`.

\Xskak<GameId>nextmovenr The number of the next move, set during parsing. Identical to `fullmove` in `\Xskak<GameId>lastfen`.

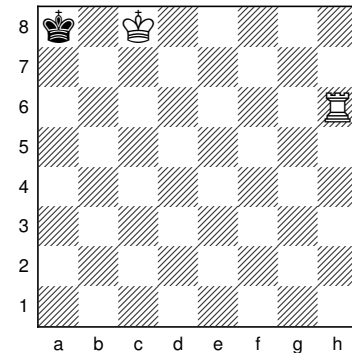
\Xskak<GameId>nextmoveid Combination of `nextmovenr` and `nextplayer`.

10... ♔a8

```
\newchessgame[id=show,
  setfen=2K/k/7R,
  moveid=10b]
\mainline{10... Ka8}

\chessboard

\xskakgetgame{nextplayer},
\xskakgetgame{nextmovenr},
\xskakgetgame{nextmoveid}
```



w, 11, 11w

\Xskak<GameId>diagramlist A comma separated list of moves (a combination of `movenr` and `player`), (e.g. 3b,20w). Records the moves with an \$d or \$D after the move. See section 3.

\Xskak<GameId>gameid This command holds the game identification (the value is equal to the value of `<GameId>`).

```
\newchessgame[id=game\arabic{section}]
The id of the current game is \xskakgetgame{gameid}
```

The id of the current game is game4

4.3.2. Optional game informations: PGN-infos

xskaknewpgninfo

The following commands store various optional informations which are normally stored in the so-called PGN-infos. You can set them in `\newchessgame` and retrieve them later. New PGN-infos can be defined with the `\xskaknewpgninfo[<default>]{<name>}`. With the exception of “result” all currently defined commands are empty by default.

\Xskak<GameId>result Default is *, value should be 1-0, or 1/2-1/2, 0-1 or something like that.

\Xskak<GameId>white The name of the white player.

\Xskak<GameId>black The name of the black player.

\Xskak<GameId>whiteelo The elo of the white player.

`\Xskak⟨GameId⟩blackelo` The elo of the black player.

`\Xskak⟨GameId⟩site` The site of the tournament.

`\Xskak⟨GameId⟩event` The name of the tournament.

`\Xskak⟨GameId⟩date` The date of the game.

`\Xskak⟨GameId⟩round` The round of game.

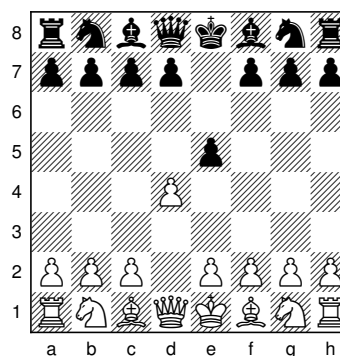
4.3.3. The move related data

The move data are stored during the parsing of a game with `\mainline` and/or `\hidemoves`. You can split the game in as much pieces as you want but **you must start a game with `\newchessgame` and you must use the same game identification for each piece** if you don't want to end with broken games.

All values depends on the parsing from the package `skak` which in turn depends heavily on the correct input: the package `skak` knows a bit about the position of the game (it needs it e.g. to find out where a piece came from) but doesn't analyze it thoroughly. The main work is done by analyzing the *notation* e.g. the package `skak` will recognize a check by the +-sign. Quite often the package `skak` doesn't recognize illegal moves or wrong input syntax (e.g. 0-0 for casting instead the correct O-O) but fails some moves later or shows a wrong position:

```
\newchessgame
\mainline{1.d4 e5 2.Bc4}
\chessboard
```

1. d4 e5 2. ♖c4



a). Piece data The following types store the piece which was moved and the piece that was perhaps captured. The variants differ in the storing of black pieces and pawns:

pgnpiece K,Q,R,B or N. Is empty for pawn and castling moves.

piecechar K,Q,R,B,N or p. Is empty for castling.

piece K,Q,R,B,N or P or k,q,r,b,n or p (for black). Is empty for castling.

pgnlostpiece In case of a capture move: K,Q,R,B, or N.

lostpiecechar In case of a capture move: K,Q,R,B,N or p, empty else.

lostpiece In case of a capture move it stores the lost piece as K,Q,R,B,N or P or k,q,r,b,n or p.

b). Move data

movefrom The field were the piece came from. Two fields in case of castling moves e.g. “e1,h1”.

pgnmovefrom This contains the additional “movefrom” information in the PGN for ambiguous moves or pawn captures. E.g. In case of **exd5** the value is “e”, in case of ♖**bd2** it is “b”.

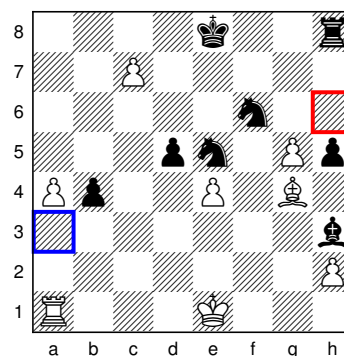
moveto The field were the piece move to. Two fields in case of castling moves e.g. “g1,f1”.

enpassant Sets the boolean `xskakboolenpassant` to true if the move was an enpassant capture.

enpassantsquare A field, after e.g. e4 the value is e3. Indicates if the next pawn move can be an enpassant capture.

```
\newchessgame[restorefen=example]
\mainline{1. a4 h5}
\chessboard[lastmoveid,
  pgfstyle=border,
  color=red,
  markfield=\xskakget{enpassantsquare},
  moveid=1w, color=blue,
  markfield=\xskakget{enpassantsquare}]
```

1. a4 h5



castling Sets the boolean `xskakboolcastling`, is true for short *and* long castling!

longcastling Sets the boolean `xskakboollongcastling`.

capture Sets the boolean `xskakboolcapture`.

promotion Sets the boolean `xskakboolpromotion`.

promotionpiece Q,R,B or N or q,r,b or n or empty.

promotionpiecechar Q,R,B or N or empty.

```

\newcommand\testpromotion{%
  \xskakget{promotion}%
  \ifthenelse{\boolean{xskakboolpromotion}}
  {A pawn promoted to
  \figsymbol{\xskakget{promotionpiecechar}}}
  {No promotion in this move}}
\newchessgame[restorefen=example]
\mainline{1. c8=Q+ Ke7}

\xskakset{moveid=1w}
\testpromotion

\xskakset{moveid=1b}
\testpromotion

```

1. c8♙+♚e7

A pawn promoted to ♙

No promotion in this move

c). Other data/combinations

addpieces contains the (list of) piece(s) that should be put on the board.

E.g. {kg8, rf8} in case of black short castling.

clearfields The list of fields that should be emptied on the board (equal to the move from field with the exception of enpassant moves).

nextfen The fen position *after* the current move.

pastfen The fen position *before* the current move (identical to the fen *after* the previous move).

san The standard algebraic notation of the move. It use chess commands: E.g.

\textsymfigsymbol{R}a2 for ♖a2

e\capturesymbol d5 for e×d5,

\castlingchar\castlinghyphen\castlingchar for O-O

It contains also the signs for chess and mate, but not the comments.

lan Long algebraic notation of the move, similar to san.

```

\newchessgame
\mainline{1.f3 e5 2. g4?? Qh4+#!}

\xskakset{moveid=2w}%
\xskakget{san}, \xskakget{lan}

\xskakset{moveid=2b}%
\xskakget{san}, \xskakget{lan}

```

1. f3 e5 2. g4?? ♙h4+#!

g4, g2–g4

♙h4+#!, ♙d8–h4+#!

opennr This contains the code to print the number like the package skak would do it if you start or continue a game with this move. The style must be set independantly either with a \styleX of skak or – if you use the new interface of xskak to set a style – with \xskakset and \mainlinestyle or \variationstyle.

```

\newchessgame
\mainline{1.e4 e5}

\xskakset{moveid=1w}%
\xskakget{opennr}\xskakget{lan},
{\styleB\mainlinestyle
\xskakget{opennr}\xskakget{lan}}

\xskakset{moveid=1b}%
\xskakget{opennr}\xskakget{lan}
\xskakset{style=styleA,level=1}
{\mainlinestyle
\xskakget{opennr}\xskakget{lan}}

\xskakset{style=UF}%return to my style

```

1. e4 e5
1. e2–e4, **1. e2–e4**
1... e7–e5 **1. -, e7–e5**

comments contains the short comments after a move (!,? → etc)

```

\newchessgame
\mainline{1.f3 e5 2. g4?? Qh4+#!}

\xskakset{moveid=2w}%
\xskakget{comments}

\xskakset{moveid=2b}%
\xskakget{comments}

```

1. f3 e5 2. g4?? ♖h4+#!
??
!

nag NAG's and comments inputed with \xskakcomment.

```

\newchessgame
\mainline{1.f3 e5 2. g4
          $4 \xskakcomment{ Why this? } Qh4+#!}

\xskakset{moveid=2w}%
\xskakget{nag}

```

1. f3 e5 2. g4?? Why this?
♖h4+#!
?? Why this?

4.4. Exporting the games

xskakexportgames

With \xskakexportgames[*<key=value list>*] you can export games to a file. As default the command exports the game data of the game with the default *<GameId>* to the file `xskakgames.xsk`⁸.

The command accept the following keys:

file=*<string>* This changes the name of the file. The extension can not be changed.

games={*<comma separated list of games>*} With this key you can chose the games that should be exported.

⁸I didn't find any other application that use this extension. So I think it is quite safe.

You can then input the games in other document and use there the game data without have to do all the parsing with `\mainline` again (which can be quite slow). The export file is also quite handy to debug the game parsing and saving.

As an example I exported in another document the following two games which I will use later in this documentation:

```
\newchessgame[id=export,white=Deep Blue, black=Kasparov,result=1-0]
\mainline{1.e4 c6 2.d4 d5 3.Nc3 dxe4 4.Nxe4 Nd7 5.Ng5 Ngf6 6.Bd3 e6
7.N1f3 h6 8.Nxe6 Qe7 9.O-O fxe6 10.Bg6+ Kd8 \xskakcomment{Kasparov
schüttelt kurz den Kopf} 11.Bf4 b5 12.a4 Bb7 13.Re1 Nd5 14.Bg3 Kc8
15.axb5 cxb5 16.Qd3 Bc6 17.Bf5 exf5 18.Rxe7 Bxe7 19.c4}

\newchessgame[white=none,black=none,id=exportb]
\mainline{1. e4 e5}

\xskakexportgames[games={export,exportb}]
```

`\xskakendgamedata`
`\xskakcurrentgameid`

The export consists more or less of a long list of definitions with one notable exception: At the end of each game there is the command `\xskakendgamedata`. As a default the command does nothing but if you redefine it e.g. to `\printchessgame` (see section 7.5), it can be used to print all games during the input. `\xskakcurrentgameid` holds the *GameId* of the previous game.

```
\renewcommand\xskakendgamedata
{\xskakset{id=\xskakcurrentgameid}%
\xskakgetgame{white}--\xskakgetgame{black}:
\printchessgame[style=styleB,level=1]
\par\bigskip}
\input{xskakgames.xsk}
```

Deep Blue–Kasparov: 1 e4 c6 2 d4 d5 3 ♘c3 dxe4
4 ♗xe4 ♘d7 5 ♘g5 ♘gf6 6 ♕d3 e6 7 ♘1f3 h6 8
♗xe6 ♚e7 9 O-O fxe6 10 ♜g6+ ♚d8 11 ♜f4 b5
12 a4 ♜b7 13 ♚e1 ♘d5 14 ♜g3 ♙c8 15 axb5
cxb5 16 ♚d3 ♜c6 17 ♜f5 exf5 18 ♚xe7 ♜xe7 19
c4

none–none: 1 e4 e5

4.5. The inner technique: how the data are stored

The game related informations are stored in commands with names of the following structure:

`Xskak`*<GameId>**<type of Information>*

If *<GameId>* is a simple string of characters you can use the command by putting a backslash before the name, in other case you will have to use `\csname... \endcsname`.

The move related informations are stored in commands with names that have the following structure:

`Xskak`.*<GameId>*.*<movenr>*.*<player>*.*<type of Information>*

As the names contain points and a number it is not possible to execute the commands by putting a backslash before the name. If you want to use them you will have to surround the name by `\curname... \endcurname` – something that probably will not be necessary in normal cases as there are user commands that allows “normal” access.

During storing the data $\langle \text{movenr} \rangle$ and $\langle \text{player} \rangle$ are taken from the current move as processed by the package skak, that is `\mainline{15... Nf3}` will store the data in `Xskak.<GameId>.15.b.<type of Information>`. Renumbering the moves during storing is not possible⁹.

Example 6 on page 26 shows an example that use the internal commands. As you can see, the `\curname–\endcurname`-input is quite powerful: You can construct a command with the help of other (expandable) commands or use the value of counters.

Warning!

`\curname whatever \endcurname` doesn't give an error if the command isn't defined. It simply defines `\whatever` as `\relax` and then executes it. So it is easy to overlook typing errors – and then to wonder why one doesn't get the expected output.

⁹But it would be probably not difficult to implement. So if you need it badly, sent me a feature request.

Example 6: Using the internal commands

Game 1

1. e4 e5 2. ♘f3 ♘c6 3. ♙b5 ♘f6 4. O-O
 ♜×e4 5. ♙×c6 b×c6

Game 2

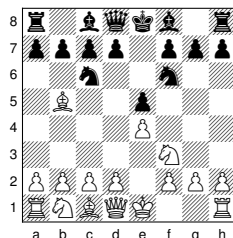
15. c8♚+ ♔e7 16. ♖d1

Comments

In the first game, white moved at first a ♘, then a ♘ and a ♙. In his last move black took a ♙, white took a ♘.

In the second game the second move of white was ♚a1–d1.

The position in the first game after the third move of black was:



The fourth move of white was a castling move.

```
\begin{multicols}{2}
\minisec{Game 1}\newchessgame[id=gameA]%
\mainline{1. e4 e5 2. Nf3 Nc6 3. Bb5 Nf6 4. O-O Nxe4 5.Bxc6 bxc6}

\minisec{Game 2}\newchessgame[id=game,restorefen=example,moveid=15w]%
\mainline{15. c8=Q+ Ke7 16. Rd1}

\minisec{Comments}\setcounter{enumi}{1}%
In the first game, white moved at first a
\figsymbol{\csname Xskak.gameA.\the\value{enumi}.w.piecechar\endcsname},
then a \figsymbol{%
\csname Xskak.gameA.\the\numexpr\the\value{enumi}+1\relax.w.piecechar\endcsname}
and a \figsymbol{%
\csname Xskak.gameA.\the\numexpr\the\value{enumi}+2\relax.w.piecechar\endcsname}.
In his last move black took a
\figsymbol{\csname Xskak.gameA.\XskakgameAlastmovenr.b.lostpiecechar\endcsname},
white took a \figsymbol{\csname
Xskak.gameA.5.w.lostpiecechar\endcsname}.

In the second game the second move of white was
\figsymbol{%
\csname Xskak.game.\the\numexpr \Xskakgameinitmovenr +1\relax.w.piece\endcsname}%
\csname Xskak.game.16.w.movefrom\endcsname--%
\csname Xskak.game.16.w.moveto\endcsname.

The position in the first game after the third move of black was:

\chessboard[tinyboard, setfen=\csname Xskak.gameA.3.b.nextfen\endcsname]
The fourth move of white was\csname
Xskak.gameA.4.w.castling\endcsname
\ifthenelse{\boolean{xskakboolcastling}}{\not} a castling move.
\end{multicols}
```

5. Retrieving the stored informations

5.1. Setting the variables

\xskakset

It should be clear that to retrieve the informations you must tell xskak the game and the move for which you want to get data. This is done with `\xskakset {⟨key=value list⟩}` which will set the variables `⟨GameId⟩` and/or `⟨MoveId⟩` to the values listed in the `key=value list`.

Some possible¹⁰ keys are:

id=⟨name⟩ The key will set/change globally the active game identification, the `⟨GameId⟩`. This will also affect the storing of the following moves! So if you change the `⟨GameId⟩` in the middle of the parsing of a game, don't forget to reset it to the old value before continuing!

movenr=⟨number⟩, player=⟨“w” or “b”⟩, moveid=⟨number + “w” or “b”⟩ The keys will set the move number and the player. The values are global but the values of `movenr` and `player` are not expanded directly while the value of `moveid` is. The keys change only the internal commands that are used to retrieve informations, the move counter and the boolean of `\mainline` are not affected by this keys.

stepmoveid=⟨number⟩ `stepmoveid` picks up the current value of the move variables of xskak and “adds” the given number of halfmoves. When no number is given the value 1 is used and this gives the next move. E.g. if the current move is 10w and you use `stepmoveid=3` then you get 11b. Negative numbers can be used, if this results in a move number smaller than 1, a warning is given but not an error: This gives you the possibility e.g. in loops to handle the case yourself.

lastmoveid=⟨GameId⟩ This key sets the xskak move identification commands to the number/color of the last move of the game `⟨GameId⟩`. `⟨GameId⟩` is optional. `lastmoveid=A` is equivalent to `id=A, lastmoveid`.

tag=⟨name⟩, ref...=⟨name⟩ `tag` saves the current values of `⟨GameId⟩` and the move in internal commands. They can then later be retrieved by the keys of type `ref`.

For more information on the labeling and referencing see section 5.5.

defaultid=⟨name⟩ With this key you can change the default game identification. The change is global.

¹⁰More keys are described later.

```
\setcounter{gamecounter}{4}
\xskakset{defaultid=game\Alph{gamecounter}}
```

```
\newchessgame[white=John]
\mainline{1. e4 e5}
```

1. e4 e5

```
\stepcounter{gamecounter}
\newchessgame[white=Jack]
\mainline{1. d4 d5}
```

1. d4 d5

The white players were John and Jack.

```
The white players were \XskakgameDwhite\ and
\XskakgameEwhite.
```

defaultmovenr=*<number>*, **defaultplayer**=*<“w” or “b”>*,

defaultmoveid=*<number><“w” or “b”>* With this keys you can change the default start move for the subsequent games. The change is global.

defaultfen=*<FEN>* As a default `\newchessgame` sets the start position to a new game. You can change this default with this key. The change is global.

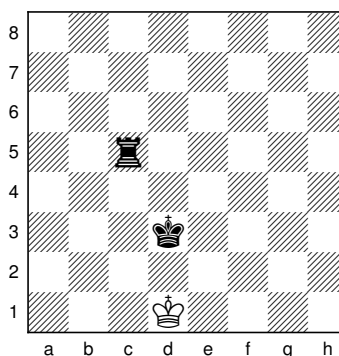
5.2. Getting the game informations

`\xskakgetgame` With `\xskakgetgame{<type>}` you can retrieve game informations for the currently active *<GameId>* (which can be set/changed with the commands `\newchessgame`, `\resumechessgame` and `\xskakset`).

```
\newchessgame[id=A,white=John White,
               black=Jack Black,
               result=1--0]
\minisec{\xskakgetgame{white}--
         \xskakgetgame{black}}
\mainline{1.e4 e5 2.Bc4 Nc6 3. Qh5 Nf6 4.Qxf7#}
\mbox{\xskakgetgame{result}}
```

John White– Jack Black

1. e4 e5 2. ♖c4 ♜c6 3. ♕h5 ♞f6 4. ♕xf7#
1-0



```
\newchessgame[id=B,
               white=Harry Red,
               black=Tom Green,
               result=0--1,
               setwhite={kd1},
               addblack={kd3,rc5}]
\chessboard[setfen={\xskakgetgame{initfen}}]
\minisec{\xskakgetgame{white} --
         \xskakgetgame{black}}
\mainline{1. Ke1 Rf5 2. Kd1 Rf1#}
\mbox{\xskakgetgame{result}}
```

Harry Red – Tom Green

1. ♔e1 ♖f5 2. ♔d1 ♖f1# 0-1

It is also possible to get the stored information by simply calling the command `\xskak<gameid><type>` (if *<gameid>* consists of letters).

```

\newchessgame[id=A,white=John White,
               black=Jack Black, result=1--0]
\minisec{\XskakAwhite\ -- \XskakAblack}
\mainline{1.e4 e5 2.Bc4 Nc6 3. Qh5 Nf6 4.Qxf7#}
\mbox{\XskakAresult}
\newchessgame[id=B,white=Harry Red,
               black=Tom Green, result=0--1]
\minisec{\XskakBwhite\ -- \XskakBblack}
\mainline{1.f3 e5 2.g4 Qh4#}
\mbox{\XskakBresult}

\minisec{List of Results}
\begin{tabular}[t]{l@{\,}--\,}lr}
\XskakAwhite&\XskakAblack&\XskakAresult\\
\XskakBwhite&\XskakBblack&\XskakBresult
\end{tabular}

\minisec{Final positions}
\chessboard[tinyboard,lastmoveid=A,
             setfen=\xskakget{nextfen}]
\chessboard[tinyboard,lastmoveid=B,
             setfen=\xskakget{nextfen}]

```

John White – Jack Black

1. e4 e5 2. ♖c4 ♗c6 3. ♕h5 ♖f6 4. ♕xf7#
1-0

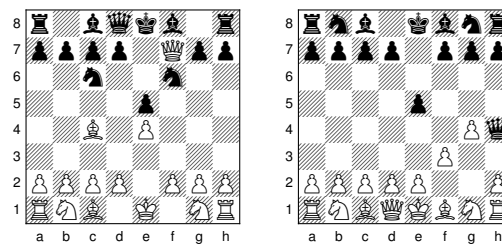
Harry Red – Tom Green

1. f3 e5 2. g4 ♕h4# 0-1

List of Results

John White – Jack Black 1-0
Harry Red – Tom Green 0-1

Final positions



5.3. Getting the move informations

Move informations are bit more difficult as they depend on more variables.

At first, you can naturally use a `\csname . . . \endcsname`:

```
\csname xskak.<gameid>.<movenr>.<player>.<type>\endcsname.
```

But normally you should use a `\xskakset/\xskakget`-combination¹¹.

`\xskakget` `\xskakget{<type>}` will pick up the current values of the variables for the `<GameId>` and the `<MoveId>` and then retrieve/print the content of type `<type>`.

`<type>` can be any of the types described in 4.3.3 and additionally the three virtual types `movenr`, `player` and `moveid` can be used which will output the current move identification. This allows easier retrieving of the current value of the internal variables e.g. in loops and tests.

Attention: The move variables are not changed by the parsing command of the package `skak`. And they are not stored like the other types – you can't retrieve them with a `\csname . . . \endcsname` construction.

¹¹I had to implement the retrieving as a two-step-method to be able to use the retrieving command in the `key=value list` of `\chessboard`. As the values are expanded during the processing complicated commands which use internally `\def` and `\setkeys` can not be used as values.

```
\newchessgame[moveid=10b]
\mainline{10... e5}
```

The game started with move `\xskakget{moveid}`.

```
\xskakset{moveid=5w}
```

Now the moveid has value `\xskakget{movenr}\xskakget{player}`.

10... e5

The game started with move 10b.

Now the moveid has value 5w.

5.4. Using the data with `\chessboard`

The move data can naturally be used with `\chessboard` e.g. to highlight the last move with an arrow. The $\langle MoveId \rangle$ you want to refer to can be set before the board with `\xskakset` but if this were the sole possibility this would mean that you couldn't get data from other moves. So I defined a bunch of new keys for `\chessboard` (and changed some existing keys) which allows to retrieve during building the board all available move datas. The most important ones are keys that tells `\chessboard` the $\langle GameId \rangle$ and the $\langle MoveId \rangle$ for which you want to retrieve datas:

id= $\langle name \rangle$ Like the other id-keys this one sets the variable that identifies the game. But setting this key will *not* affect the value of the game identification outside the board, any change is local to the board.

Attention: Setting this key will *not* change the default position shown by `\chessboard`. This is always the "current" game.

movenr= $\langle number \rangle$, player= $\langle "w" \text{ or } "b" \rangle$, moveid= $\langle number + "w" \text{ or } "b" \rangle$ This keys stores the number and player for which you want to retrieve move related informations. Again any setting is local to the board.

stepmoveid= $\langle number \rangle$ Like the similar key of `\xskakset` `stepmoveid` picks up the current value of the move variables of `xskak` and "adds" the given number of halfmoves. But again when you use the key in the argument of `\chessboard` the change is local to the board.

Attention: If you set the $\langle MoveId \rangle$ with the above keys to a value "outside" the game and then try to retrieve a data you will probably get horrible errors.

lastmoveid= $\langle gameid \rangle$ The usual case is probably that you want to use the (currently) last move of a game. This can be done with this key. The (optional) value lets you specify a game identification. `lastmoveid=A` is equivalent to `id=A, lastmoveid`. Again the setting is local to the board.

After you have set the move variables, you can use the command `\xskakget{\langle type \rangle}` described above used without problems¹² inside `\chessboard` as long as the value expands to something the keys can handle (`markmove` e.g. will not like it if you ask it to process a fen).

¹²I hope but I haven't tested any possible usage.

```

\newchessgame[id=A,white=John White,
               black=Jack Black, result=1--0]
\mainline{1.e4 e5 2.Bc4 Nc6 3. Qh5 Nf6 4.Qxf7#}

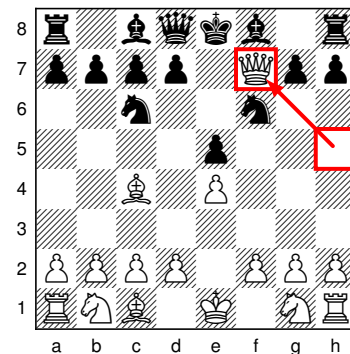
\newchessgame % a game between A and the board
\mainline{1.e4}

\setchessboard{shortenend=5pt,color=red}%
\chessboard[lastmoveid=A,setfen=\xskakget{nextfen},
             pgfstyle=border,color=red,
             markfields={\xskakget{moveto},\xskakget{movefrom}},
             pgfstyle=straightmove,
             markmove=\xskakget{movefrom}-\xskakget{moveto}]

```

1. e4 e5 2. ♖c4 ♘c6 3. ♙h5 ♗f6
4. ♙×f7#

1. e4



```

\newchessgame[id=A]
\mainline{1.e4 e5 2.Bc4 Nc6 3. Qh5 Nf6 4.Qxf7#}

\xskakget{moveid}% to show <MoveId> is still 1w

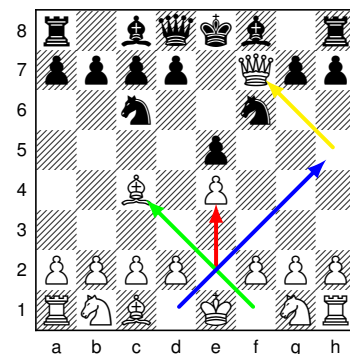
\setchessboard{shortenend=5pt,color=red}%
\chessboard[lastmoveid=A,setfen=\xskakget{nextfen},
             moveid=1w,
             pgfstyle=straightmove,
             markmove=\xskakget{movefrom}-\xskakget{moveto},
             stepmoveid=2,color=green,
             markmove=\xskakget{movefrom}-\xskakget{moveto},
             stepmoveid=2,color=blue,
             markmove=\xskakget{movefrom}-\xskakget{moveto},
             stepmoveid=2,color=yellow,
             markmove=\xskakget{movefrom}-\xskakget{moveto}]

\xskakget{moveid}% to show that <MoveId> is again 1w

```

1. e4 e5 2. ♖c4 ♘c6 3. ♙h5 ♗f6
4. ♙×f7#

1w



1w

5.5. Labeling a game

It is a bit tedious to have to remember the number and the color of the moves when you want to start or to continue a game, or for which you want to draw a diagram or retrieve another move information. So there is a sort of labeling system. In difference to the labeling in normal \LaTeX it doesn't write the values to auxiliary files. That means that you can't reference labels which come later in the document (at my opinion this would lead to chaos anyway).

To set a label you use the command `\xskakset` with the key `tag`¹³. It saves the current value of the move counter (the `skak` counter!), the value of the (`skak!`) boolean that holds the player and the `<GameId>`. If you haven't started any game yet (more precisely: if the move counter has value 0) then the package `xskak` will issue a warning and do nothing.

¹³The name "label" is used by `\chessboard` to set the labels of the boards.

You can set the tag between two `\mainline` (or `\hidemoves`) commands, but it works also if you put them directly in the argument of `\mainline`:

```
\newchessgame%
\mainline{1. e4 e5\xskakset{tag=A}
  2. Nf3\xskakset{tag=B} Nc6
  3. Bb5 Nf6
  4. O-O Nxe4
  5.Bxc6 bxc6}
```

**1. e4 e5 2. ♘f3 ♗c6 3. ♖b5 ♗f6 4. O-O ♗×e4
5. ♖×c6 b×c6**

The saved values can be retrieved with various keys of type `ref` (e.g. `refpastmoveid`, `refnextplayer`), or by the command `\xskakget`.

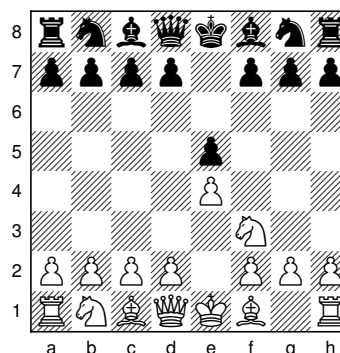
The label is set in most cases between two moves. There probably will be cases when you need the *MoveId* of the move *before* the tag e.g. to show the position or to comment on the last move, and there will be cases when you will need the *MoveId* of the move *after* the tag e.g. to continue the game. So there are `ref`-keys for both cases.

5.5.1. Retrieving the tagged values with `\xskakget`

Retrieving the *GameId* and/or *MoveId* values stored by a tag is again a more-than-one-step procedure: At first you must tell `xskak` the tag name you want to use, and then use a `\xskakget{<list of ref types>}`, where *ref type* is one of the following keywords: `refid`, `refpastmoveid`, `refnextmoveid`, `refpastmovenr`, `refnextmovenr`, `refpastplayer`, `refnextplayer`.

Setting the tag name is done with the key `reftag`, which you can use with all commands that accept the key `id`. If you use the key in the optional argument of `\chessboard` it will change the tag name locally for the board. In all other cases the tag name is set globally.

```
\newchessgame[id=new] % to set a new id
\chessboard[
  reftag=B,%from previous example
  id=\xskakget{refid},
  %retrieves and sets the <Gameid> from tag B
  moveid=\xskakget{refnextmoveid},
  %retrieves and sets the <Moveid> from tag B
  setfen=\xskakget{pastfen}]
```



5.5.2. The ref-keys

As the more-step-procedure to use the tags is a bit longish there exist keys that shorten the procedure. This keys too work in the arguments of all commands that can also handle the correspondent keys `id`, `moveid` etc. The effect will be identical to setting the values directly: Keys used in the optional argument of `\chessboard` will set their values only local to the board, in the other cases the values are set globally.

refid=*<tag>* This key will set the *<GameId>* to the value stored in the tag. It is a shortcut for `reftag=<tag>,id=\xskakgetverb+refid+`

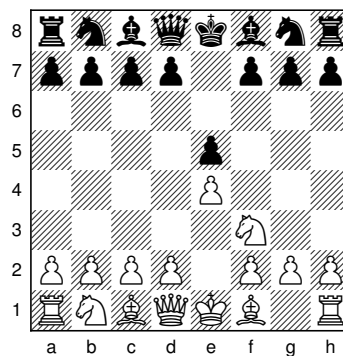
refpastmoveid=*<tag>*, **refpastmovenr**=*<tag>*, **refpastplayer**=*<tag>* This keys will set the respective move information to the value of the move before the tag. If the tag is before the move 1w, then the value 0b is used.

refnextmoveid=*<tag>*, **refnextmovenr**=*<tag>*, **refnextplayer**=*<tag>* This keys will set the respective move information to the value of the move after the tag.

refpast=*<tag>*, **refnext**=*<tag>* This keys will set the *<GameId>* and the *<MoveId>* in one go. E.g. `refpast` is a shortcut for `reftag=<tag>,id=\xskakgetverb+refid+,moveid=\xskakgetverb+refpastmoveid+`

reffen=*<tag>* This is a `\chessboard` key. It will add the position at the tag on the board, its effect is identical to using the key `addfen =<Position at tag>` described in the documentation of the package `chessboard`. So be aware that it really only places pieces on the board. It will not change the *<GameId>* nor the *<MoveId>* or the move counter of `skak`. So it is quite possible to use a position where black should move next to start a game with white to move.

```
\newchessgame[id=new]% to set a new id
\chessboard[reffen=B]
```



1. e4 e5 2. ♖f3 ♘c6 3. ♙b5 ♗f6 4. O-O ♘×e4
5. ♙×c6 b×c6

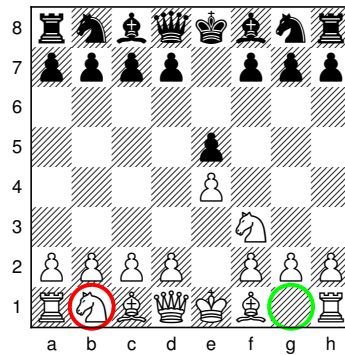
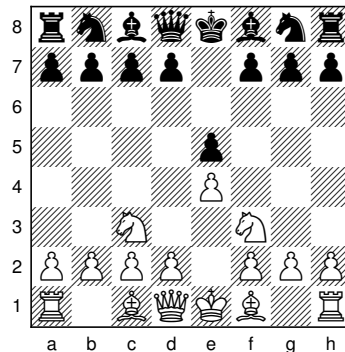
1. ♘c3

```
\newchessgame[id=A]%
\mainline{1.e4 e5 2. Nf3\xskakset{tag=C}
  Nc6
3. Bb5 Nf6 4. O-O Nxe4 5.Bxc6 bxc6}
```

```
\newchessgame[id=B,reffen=C]
\mainline{1. Nc3}
```

```
\chessboard
```

```
\chessboard[reffen=C,
  markstyle=circle,
  color=red,
  %from game B:
  markfield=\xskakget{movefrom},
  refpast=C,
  color=green,
  %from game A:
  markfield=\xskakget{movefrom}]
```



6. Looping

Looping with the `\whiledo` command of the package `ifthen` is not very difficult. But to adjust to the special numbering of chessmoves (1w, 1b, 2w, 2b, 3w ...) one needs some special commands to step through the moves and test for the end of the loop.

`\xskaktestmoveid`

Stepping can be done with the key `stepmoveid` described earlier.

`\xskaktestmoveid{<movenr>}{<player>}` is a test (defined with the help of package `xifthen`) which you can use in the first argument of `\whiledo`. It will give true if the move is in the current game.

```

\newchessgame[id=A]
\mainline{1. e4 e5 2. Nf3 Nf6}

\newchessgame[id=B]
(a second game)

Reprinting the first game:
\xskakset{id=A,
  moveid=\xskakgetgame{initmoveid}}
\whiledo{\xskaktestmoveid
  {\xskakget{movenr}}{\xskakget{player}}}
{\ifthenelse%
  {\equal{\xskakget{player}}{w}}
  { \xskakget{movenr}.\,}
  { }%
  \xskakget{lan}%
  \xskakset{stepmoveid}}

Reprinting only the black moves:
\xskakset{%
  id=A,
  moveid=\xskakgetgame{initmoveid},stepmoveid}
\whiledo{\xskaktestmoveid
  {\xskakget{movenr}}{\xskakget{player}}}
{ \xskakget{lan}%
  \xskakset{stepmoveid=2}}

```

1. e4 e5 2. ♖f3 ♗f6

(a second game)

Reprinting the first game: 1. e2–e4 e7–e5

2. ♗g1–f3 ♗g8–f6

Reprinting only the black moves: e7–e5 ♗g8–f6

`\xskakloop` `\xskakloop[⟨key=value list⟩]{⟨code⟩}` is a wrapper command for such a `\whiledo` loop.

The possible keys are:

id=⟨string⟩ Which sets the game identification. The setting is global.

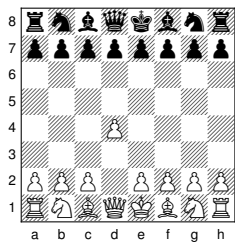
initmovenr=⟨number⟩, initplayer=⟨“w” or “b”⟩, initmoveid=⟨number + “w” or “b”⟩ This sets the move the loop should start with. Default is the first move of the game.

stopmovenr =⟨number⟩, stopplayer =⟨“w” or “b”⟩, stopmoveid=⟨number + “w” or “b”⟩ This sets the move where the loop should stop. Default is the last move of the game.

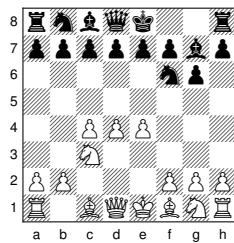
step=⟨number⟩ This sets the length of the steps of the loop. The number gives the number of halfmoves. Default is one.

showlast=⟨true|false⟩ When you are using the key `step` with a number unequal to one, it is possible that the last move (possibly set by `stopmoveid`) is left out. With the key `showlast` you can force `\xskakloop` to process the code for the last move.

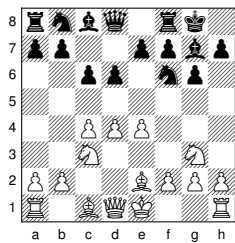
1. d4 Nf6 2. c4 g6 3. Nc3 Bg7 4. e4 d6 5. Nge2 O-O 6. Ng3 c6 7. Be2 a6 8. a4 a5 9. h4 h5
 10. Be3 Na6 11. f3 e5 12. d5 Nd7 13. Nf1 Ndc5 14. Nd2 Qb6 15. Qb1 Nb4 16. Nb3
 Ncd3+ 17. Kd2 Qxe3+



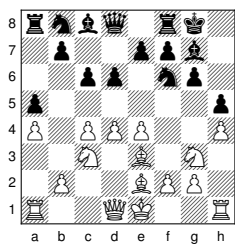
1. d2-d4



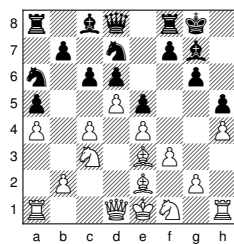
4. e2-e4



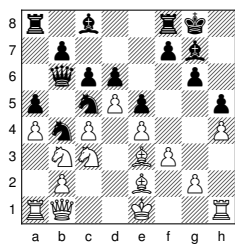
7. f1-e2



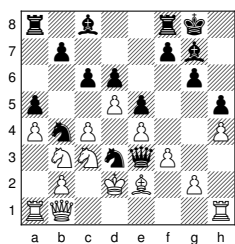
10. c1-e3



13. g3-f1



16. d2-b3



17... b6xe3+

```
\newchessgame
\mainline{%
1.d4 Nf6 2.c4 g6 3.Nc3 Bg7 4.e4 d6 5.Nge2 O-O 6.Ng3 c6 7.Be2 a6 8.a4
a5 9.h4 h5 10.Be3 Na6 11.f3 e5 12.d5 Nd7 13.Nf1 Ndc5 14.Nd2 Qb6
15.Qb1 Nb4 16.Nb3 Ncd3+ 17.Kd2 Qxe3+}

\xskakloop[step=6,showlast]{%
\begin{tabular}{c}
\chessboard[tinyboard,
setfen=\xskakget{nextfen}]
\\
\xskakget{opennr}\xskakget{lan}%
\end{tabular}\quad}%
```

With the help of the package `animate` and `\xskakloop` you can make animated boards:

The board was made with the following code:

```
\setchessboard{boardfontsize=0.8cm,shortenend=5pt, margin=false}%
\setboardfontencoding{LSBC4}%
\newskaklanguage{german}{KDTLSB}\skaklanguage[german]%
%
\newcommand\getmovestyle[1]{%
\ifthenelse
{\equal{#1}{N}}%knight move
{\def\mymovestyle{[clockwise=false,style=knight]curvemove}}
{\ifthenelse
{\equal{#1}{}}% castling
{\def\mymovestyle{curvemove}}
{\def\mymovestyle{straightmove}}}}%
%
\newchessgame[white=Paethz,black=Dirr,result=0-1,id=anim]%
\hidemoves{%
1.d4 Sf6 2.c4 g6 3.Sc3 Lg7 4.e4 d6 5.Sge2 0-0
6.Sg3 c6 7.Le2 a6 8.a4 a5 9.h4 h5 10.Le3 Sa6
11.f3 e5 12.d5 Sd7 13.Sf1 Sdc5 14.Sd2 Db6
15.Db1 Sb4 16.Sb3 Scd3+ 17.Kd2 Dxe3+}%
%
\unitlength0.8cm\relax
```

```

\begin{animateinline}[autoplay,loop,controls]{0.5}%
\begin{picture}(11,11)
\put(1.5,1.5){\chessboard[setfen=\xskakgetgame{initfen}]}%
\end{picture}
\newframe
\xskakloop{%
\getmovestyle{\xskakget{piecechar}}%
\begin{picture}(11,11)
\put(1.5,1.5){%
\chessboard[pgfstyle=\mymovestyle, color=blue,
pgfshortenend=0.3em,arrow=to,
markmoves=\xskakget{move},
coloremph,piececolor=blue,
emphfields={\xskakget{moveto}},
setfen=\xskakget{nextfen}]}%
\end{picture}%
\newframe}%
\begin{picture}(11,11)
\put(1.5,1.5){\chessboard[setfen=\xskakgetgame{lastfen}]}%
\end{picture}
\end{animateinline}

```

7. Printing

7.1. \longmoves revisited

In the example on page 1 I showed that the `\longmoves` of the package `skak` doesn't work correctly. This is due to the asynchronous parsing and printing of the `skak`-engine (`skak` – like `TEX` – has sometimes to look a bit ahead). With the package `xskak` the problem can be easily solved as it is now possible to retrieve informations about previous moves. So the package `xskak` redefines the internal commands to get `\longmoves` working.

Attention! The new `\longmoves` command works only correctly if the internal engine parses the moves – so *it doesn't work with \variation*. So starting with version 1.2 the package `xskak` forces `\variation` to use `\shortmoves`.

The redefinition of `\longmoves` is done by first defining a printing command and then (re-)defining `\longmoves` such that the internal `\printmove` now use the new command. The listing below shows the code¹⁴. You can take it as a template for your own special printing commands. `\WhiteToMove{<code for white>}{<code for black>}` is a command of

¹⁴The code doesn't show the complete truth. In the real code `\longmoves` also has to change the printing mode for `\printchessgame`. See 7.5

the package skak that executes the first argument if white is to move and the second if its black turn. The main trick in the new printing command is to go back one halfmove. The new printing command defines only the printing format for the move itself as the number is printed by the package skak at another place.

```
\def\xskak@do@printmove@algebraic{%
\csname Xskak.\xskak@val@gameid.%
\WhiteToMove{\the\numexpr\the\c@move-1\relax}{\the\c@move}.%
\WhiteToMove{b}{w}.lan\endcsname
\csname Xskak.\xskak@val@gameid.%
\WhiteToMove{\the\numexpr\the\c@move-1\relax}{\the\c@move}.%
\WhiteToMove{b}{w}.comments\endcsname{}}

\def\longmoves{%
\let\printmove=\xskak@do@printmove@algebraic}
```

xskakenpassanttext

The new printing command inserts a test for enpassant moves. You can suppress it by redefining `\xskakenpassanttext`.

```
\longmoves
\newchessgame
\mainline{1.e4 e5 2.Nf3 Nf6}

\newchessgame[restorefen=example]
\mainline{1. a4 bxa3!}

\renewcommand\xskakenpassanttext{}
\newchessgame[restorefen=example]
\mainline{1. a4 bxa3!}
```

1. e2–e4 e7–e5 2. ♖g1–f3 ♗g8–f6

1. a2–a4 b4×a3!

1. a2–a4 b4×a3!

7.2. An user interface for styles

To format a game the package skak puts during parsing commands before and between the moves and the counter. You can see the names of this commands in the following example (the style test is a simple local style predefined by xskak which contains different tests. In level 1 and 2 it contains style items to show commands. The optional argument [invar] force \variation to use level 2).

Example 7: The internal commands to format the output

```
\mainlinestyle \opencommands \beforenumber 1\whiteopen \beforewhite  
d4\afterwhite \beforeblack ♖f6\afterblack \beforenumber 2\whiteopen  
\beforewhite c4\closecommands
```

```
\mainlinestyle \opencommands \beforenumber 2\blackopen \beforeblack  
g6\afterblack \beforenumber 3\whiteopen \beforewhite ♗c3\afterwhite  
\beforeblack ♗g7\afterblack \beforenumber 4\whiteopen \beforewhite  
e4\closecommands
```

```
\mainlinestyle \opencommands \beforenumber 4\blackopen \beforeblack  
O-O\afterblack \beforenumber 5\whiteopen \beforewhite ♖f3\afterwhite  
\beforeblack ♗e8\closecommands
```

```
\mainlinestyle \opencommands \beforenumber 6\whiteopen \beforewhite  
♗e3\afterwhite \beforeblack e6\closecommands
```

```
\variationstyle \opencommands \beforenumber 6\blackopen \beforeblack  
d6\closecommands
```

```
\newchessgame  
\xskakset{%  
  style=test,  
  level=1}  
\color{blue}  
  
\mainline{1.d4 Nf6 2.c4}  
  
\mainline{2... g6 3.Nc3 Bg7 4.e4}  
  
\mainline{4... O-O 5. Nf3 Re8}  
  
\mainline{6. Be3 e6}  
  
\variation[invar]{6... d6}
```

By judiciously (re-)defining the commands skak inserts one can get almost every chess style one want. skak itself defines three examples that demonstrates the possibilities. These styles can be chosen by the commands `\styleA`, `\styleB` and `\styleC`. The first two are inline styles, the third uses for `\mainline` a tabbing environment and switches to an inline style for `\variation`. But this styles are really no more than examples. There is no user interface to change and define styles. And due to some dependencies between the predefined styles (`\styleB` is not only a stand-alone style but it is also used in the definition of `\styleC` for the variation level) adapting the styles is not easy.

So I decided to define a real user interface to define styles and switch from one to another. The user interface supports an unlimited number of levels: Each style is a bundle of *levels* where the look on each level is describe by a *style item* and some optional commands. Defining a style is therefore done in two steps:


```

% Define items:
\xskaknewstyleitem{A}{%
  beforenumber={},
  whiteopen={.\,},
  ...}
\xskaknewstyleitem{B}{...}

%Define style
\xskaknewstyle[%
  level=1,styleitem=A,
  font=\bfseries, %add special command to level 1.
  level=2,styleitem=A,
  level=3,styleitem=B,
  ...]{mystyle}

```

Attention! The (almost) indefinite number of levels comes at a price: `\mainline` and `\variation` no longer use automatically different styles. If a `\variation` should switch to a “variation style” you will have to use e.g. the option `[invar]` mentioned above (but you can define your own “variation” commands that do this switch). Please read section 7.3 for more informations.

7.2.1. Defining style items

`\xskaknewstyleitem`

With `\xskaknewstyleitem[⟨key=value list⟩]{⟨item⟩}` you define the style item `⟨item⟩`. You can use the following keys:

`⟨command name⟩=⟨commands⟩` where `⟨command name⟩` is one of `opencommands`, `closecommands`, `beforenumber`, `whiteopen`, `blackopen`, `beforewhite`, `beforeblack`, `afterwhite`, `afterblack`.

There are endless possibilities for the value `⟨commands⟩` – as there are endless possibilities to get more or less horrible errors. So I will give only some general advices for the definitions of style items:

- While processing `key=value lists` `xkeyval` strips spaces. So if you want to set one of the commands to a space use either commands like `\ , \space` and `~`, or protect the space by enough layers of braces.
- While processing `key=value lists` `xkeyval` strips also braces. If you need some grouping use `\begingroup` and `\endgroup` or protect the “real” braces by enough other braces.
- Use if possible switches (like `\bfseries`) instead of commands with arguments.

- Think about how you want to handle spaces at the start and the end of (bits of) a game or a variation. An input like `\mainline{1. e4}_\variation{1.e5}` will already insert a space so avoid to insert another one through e.g. `opencommands`.
- It is possible to open a group (with `\begingroup`) or a environment in one command and close it in another command. If you do this check carefully that you have balanced groups in all four combinations of white/black opening and ending the game! Check also if everything works if you have only one move.
- You don't have to give a definition for all keys/commands. Unused keys are set to empty.
- If you want to force a style (or a style item) to use the long or the short algebraic notation you can add `\longmoves/\shortmoves` to `opencommands`.

`<beforeNAG>=<commands>`, `<beforecomment>=<commands>` `xskak` puts around each NAG and comment given by `\xskakcomment` a group and before each NAG's and comment a formatting command. This commands are named `\xskak@beforeNAG` and `\xskak@beforecomment`. With the keys `beforeNAG` and `beforecomment` you can change this commands and so to some extent change the formatting of NAG and long comments (the styles of short comments like `+` or `!` are not affected!). The last command in `<commands>` can be a command that has one argument. NAG and the content of the comment will then be used as this argument. This makes is e.g. possible to gobble commands:

```
\providecommand\gobble[1]{}
\xskaknewstyleitem[%
  template=UF,%a predefined style item
  beforeNAG=\gobble,
  beforecomment=\gobble]{gobblecomments}

\xskakaddtostyle[%
  level=gobble,
  styleitem=gobblecomments]{test}

\variation[style=test,level=gobble]{1. e4! $1
  \xskakcomment{expected} e5 $2}
```

1. e4! e5

`template=<item>` With this key you can reuse the definitions from another style item.

7.2.2. Defining style

With `\xskaknewstyle[<key=value list>]{<style>}` you define the style `<style>`.

With `\xskakaddtostyle[<key=value list>]{<style>}` you can add more levels (or overwrite existing ones) to a style. Both commands can handle four keys: `level`, `styleitem`, `font` and `xfont`.

level=*number* With this key you set the number of the level for which you want to declared a “look” with the following keys. The “highest” level number is 1. Normally it should contain the description of the style for the main game. Variation styles should have larger numbers. But the *number* don’t need to be a number, strings are also possible.

If you want to keep everything simple: Use only (positive) numbers with levels. Then the levels have a well-defined order which is easy to handle. Key `invar` will add 1 to a level number, key `outvar` will subtract 1 and if you set the level to a number for which no style item as been declared `xskak` will simply subtract 1 from this number until it finds a level with a style item it can use.

But you can also use level names with chars e.g. to define variants for a specific variation level. So you could e.g. define levels `lopen`, `lclose`, `lmiddle`, `lbetteris`.

In this case it depends on your compiler how the keys `invar` and `outvar` and levels without declared style items are handled:

Case 1: You have an old pdf \TeX where the (experimental) primitive `\pdfmatch` is not defined (or you did use the package option `nopdfmatch`).

Then the keys `invar` and `outvar` will not work when the current level contains chars or other non-numerical tokens. They will issue more or less bewildering errors.

Levels without a declared style item will issue errors too.

Case 2: You have a recent pdf \TeX where the (experimental) primitive `\pdfmatch` is defined (and you didn’t use the package option `nopdfmatch`).

Case a) The level name doesn’t contain any number, e.g. `level=gobble`:

Then you will get an error if you try to set the level and you haven’t declared a style item for this level.

If the current level is `gobble` then the keys `invar` and `outvar` will issue a warning and let the level unchanged.

Case b) The level name do contain numbers, e.g. `level=a12b567`:

Then `xskak` will use the following substitution if you try to set the level and there is no style item declared: `xskak` will extract the first number from the name (in this case “12”), subtract one from this number and try the name with the new number in it. If the first number is 1, level 1 is used (it is always defined). If the first number is 0 you will get an error.

In the example `xskak` will try the levels in the following order until it finds one that it can use: `a12b567` \rightarrow `a11b567` \rightarrow `a10b567` \rightarrow ... `a2b567` \rightarrow `a1b567` \rightarrow 1

styleitem=<item> With this key you set the style item used by the current level *and all following levels* until another styleitem is set.

font=<commands> With this key you can add some specific commands to the current level (*and only to the current level*). While the key is meant to add font switches to a level it can also be used to overwrite some commands of a style item.

```
\xskaknewstyleitem[%
  template=UF,%a predefined style item
  beforenumber=\color{red},%
  whiteopen=.\,\color{black},
  blackopen=\ldots\,\color{black}]{rednumbers}

%% add to existing style test
\xskakaddtostyle[%
  level=10,styleitem=rednumbers,
  level=12,
  font=\def\beforenumber{\color{green}}\bfseries]
  {test}

\newchessgame
\mainline[style=test,level=12]{1. Nf3 Nf6 2. d4}

\variation[level=10]{2. Nc3}

\mainline[level=11]{2... d5}
```

1. ♖f3 ♖f6 2. d4
2. ♗c3
2... d5

```
\xskakset{style=test}%
\xskaknewstyleitem[%
  template=UF,%a predefined style item
  beforeNAG=\color{blue},
  beforecomment=\mdseries]{mycomments}

\xskakaddtostyle[level=13,styleitem=mycomments,
  font=\bfseries]{test}

\variation[level=13]{1. e4 $1 \xskakcomment{ as expected} e5}

\makeatletter
\xskakaddtostyle[%
  level=gobbleB,styleitem=gobblecomments,
  font=\renewcommand\xskak@beforeNAG{\color{red}}]%
  {test}
\makeatother

\variation[level=gobbleB]{1. e4! $1 \xskakcomment{ I
  expected it} e5 $2}
```

1. e4! as expected e5
1. e4!! e5?

xfont=<commands> With this key you can add some specific commands to all levels with the same first number in their name. It needs a recent pdfTeX with the command \pdfmatch.

As an example lets add three levels to the test style used above: the first (main) level should use a bold font, the second `\mdseries` and the third an italic font. For each level we will define a variant which puts a `\betteris` (\triangle) before the first move.

The example also demonstrates how to retrieve the current level. Please note the braces around some of the `\variations` which keep the level change local.

Example 8: “betteris” style

```
\xskaknewstyleitem[%
  template=UF,
  opencommands=(\betteris,
  closecommands=)]{betteris}
```

```
\xskakaddtostyle
[level=20 ,styleitem=UF,xfont=\bfseries,
 level=20bis,styleitem=betteris,
 level=21 ,xfont=\mdseries,
 level=22 ,xfont=\itshape]{test}
```

```
\xskakset{style=test}
```

```
\variation[level=20]{1. e4 e5} (level \xskakget{level})
\variation[level=20bis]{1... e6} (level \xskakget{level})
```

```
\variation[level=21]{1. e4 e5} (level \xskakget{level})
\variation[level=21bis]{1... e6} (level \xskakget{level})
```

```
\variation[level=22]{1. e4 e5} (level \xskakget{level})
\variation[level=22bis]{1... e6} (level \xskakget{level})
```

```
\variation[level=20]{1. e4 e5}
{\variation[level=\xskakget{level}bis]{1... e6}
 \variation[invar]{1... e6} \variation[invar]{1... e6}}
```

```
\variation[invar]{1. e4 e5} %to level 21
{\variation[level=\xskakget{level}bis]{1... e6}}
```

```
\variation[invar]{1. e4 e5} %to level 22
{\variation[level=\xskakget{level}bis]{1... e6}}
```

1. e4 e5 (level 20)
 \triangle **1... e6** (level 20bis)
 1. e4 e5 (level 21)
 \triangle 1... e6 (level 21bis)
1. e4 e5 (level 22)
 \triangle *1... e6* (level 22bis)
1. e4 e5 (\triangle **1... e6**)
 \triangle 1... e6 (\triangle 1... e6)
 1. e4 e5 (\triangle 1... e6)
1. e4 e5 (\triangle *1... e6*)

The previous examples demonstrate some important points to consider when defining styles:

- You can define as much levels as you want.
- The number of the levels don't have to be consecutive (they even need to be numbers). If there is no definition for a level (in the first example level 11), the style item of a previous level (in this case level 10) is used if `xskak` is able to figure out what the previous level could be.

- While it is often sensible to store the settings for the main level in level 1, and for the first variation level in level 2 and so on, you can also chose other orders or use more than one level for each logical variation level. If you e.g. want sometimes to add braces through opencommands you could define variants *na*, *nb* etc of the involved variation level.
- While it is easy to define style items and styles it is not easy sort and bundle them in a sensible way.

7.3. Using the styles

The names of the commands `\mainline` and `\variation` give the impression that you should use the first for the main game and the second for the variations. But at my opinion this impression is wrong: You should use

- `\mainline` if you want to parse *and* print chess moves,
- `\variation` if you want only to print chess moves,
- `\hidemoves` if you want only to parse moves and
- `\printchessgame` (described later) if you want to (re)print a previously parsed game.

This means that each of the printing commands can be used for the main game and for variations. And so it should be possible for each of the printing commands to set and use every style level.

In consequence this means that if you use the new interface – you don't have to – `\variation` will no longer switch to another style.

You can set the style and the level either with `\xskakset` or in the optional argument of the three printing commands. The keys you can use in both cases are:

style=*<style name>* This key sets the name of the style the following printing commands should use. The setting is always local to the current group (even with `\xskakset`¹⁵).

gstyle=*<style name>* This key sets the name of the style globally.

level=*<number>* This key sets the level the following printing commands should use. The setting is always local to the current group.

glevel=*<number>* This key sets the level globally.

invar This will add 1 to the first number in the current level (so it goes towards “higher” (deeper) levels). Again the change is local to the group. This key can break if the current value of the level is not a number.

¹⁵At first `\xskakset` did the settings globally. I changed it after the third time I had to correct the documentation because I forgot to reset a style after an example.

ginvar This is the global variant of invar.

outvar This will subtract 1 to the first number in the current level (so it goes towards level 1 as the main level). This key can break if the current value of the level is not a number.

goutvar This is the global variant of outvar.

7.4. The predefined styles

The package xskak predefines five styles: styleA, styleB and styleC imitate the predefined styles of skak with the addition that they change to italic in the third level. Style UF is my style and style test is a large container for all sorts of tests.

```
\newchessgame
\mainline[style=styleC,level=1]{%
1. e4 e6 2. d4 d5 3. e5 c5 4. c3 Qb6}
(\variation[invar]{4... Ne7 5. Nf3 Nec6}
(\variation[invar]{5... Nf5 6. h4 Be7
7. Bd3 cxd4 8. cxd4 Nc6 9. Bxf5 exf5 10. Bg5 $14})
\variation[outvar]{6. Bd3}
(\variation[invar]{6. h4 Nd7 7. h5 f6 $1 $132}))
\mainline[outvar,outvar]{ 5. Nf3 Bd7}
```

1	e4	e6
2	d4	d5
3	e5	c5
4	c3	♖b6

(4... ♗e7 5 ♗f3 ♗ec6 (5... ♗f5 6 h4
♙e7 7 ♙d3 cxd4 8 cxd4 ♗c6 9 ♙xf5
exf5 10 ♙g5±) 6 ♙d3 (6 h4 ♗d7 7 h5
f6!⇒))

5	♗f3	♙d7
---	-----	-----

7.5. The new xskak printing command

\printchessgame

As one of the aims of xskak is to separate the parsing from the printing of a game there is naturally also a printing command. With `\printchessgame[⟨key=value list⟩]` you can print a previously parsed and saved game. Without any options `\printchessgame` will print the game with the currently active `⟨GameId⟩` from start to end.

As keys you can use

id=⟨GameId⟩, **refid=⟨tag⟩** This keys change only for the printing the current `⟨GameId⟩`. Afterwards the old `⟨GameId⟩` is restored.

initmovenr=⟨number⟩, **initplayer=⟨“w” or “b”⟩**, **initmoveid=⟨number + “w” or “b”⟩**
This keys sets the move the printing should start with. If the move isn't in the game a warning is issued and the first move of the game is used instead.

stopmovenr=⟨number⟩, **stopplayer=⟨“w” or “b”⟩**, **stopmoveid=⟨number + “w” or “b”⟩**
This keys sets the move the printing should end with. If the move isn't in the game a warning is issued and the last move of the game is used instead.

reftag=*<tag>* This sets the tag name in case that you want to use tagged moves as start or end of the printing. There are no keys like `initrefpastmoveid` (I thought so long names are no longer manageable), so you will have to use e.g. `reftag=<tag>`, `initmovenr=\xskakget{refpast}`.

style, level, keyinvar and outvar `\printchessgame` use the same formatting commands as `skak`, so the style can be change through the interface defined and described in section 7.2.

To set the level and the style you can use the keys described in section 7.3 in the optional argument of `\printchessgame`. Like with `\mainline` and `\variation` the keys sets the values locally for the current group.

To print the actual moves, `\printchessgame` doesn't use the command `\printmove` from `skaksty` but an internal command named `\xskak@do@printmove`. So commands like `\longmoves` must set this command appropriately.

```
\input{xskakgames.xsk}
\printchessgame[style=UF,level=1,id=export,
  initmoveid=7w,stopmoveid=12b]
```

7. ♖1f3 h6 8. ♜×e6 ♚e7 9. O-O f×e6 10. ♘g6+
♙d8 11. ♘f4 b5 12. a4 ♘b7

7.6. Game titles and chessboard captions

There are no pre-made commands. You will have to define them with the help of the stored game data/move data. For titles I advise you to use a sort of sectioning command – this will prevent page breaks between the title and the game. Look e.g. at the definition of `\minisec` in one of the classes from the KOMA-bundle. For captions: look at the package `caption`.

8. PGN2LTX or How to sort the input

While `skak/xskak` can handle games in PGN-notation they can't handle PGN-files: neither the PGN-infos (which is a small nuisances) nor the variations and the comments in the game notation (which makes it tedious to print large games with a lot of annotations).

There exists a small number of perl scripts and applications to help to convert PGN to \LaTeX but none is really satisfactory. One of the problems that prevents better solutions is that there is no clear description what should be the result of a conversion from PGN to \LaTeX . In this section I will make some remarks about what I think would be a good output from such a converter for `xskak`¹⁶ and where I see problems. The description should also help you to enter or convert complicated games manually.

¹⁶`texmate` needs another output.

The PGN-infos

The “tag pair” section of a chess game in PGN shouldn’t pose problems to a converter. The tags should be simply converted to the equivalent keys of a `\newchessgame`:

```
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]
```

```
\newchessgame[id=<GameId>, %see below
    round={29},
    white={Fischer, Robert J.},
    black={Spassky, Boris V.},
    result={12-12}]
```

If the game use a `SetUp` and a `FEN` tag to set an alternative starting position, the converter will have to extract the start move from the FEN and set it with the key `moveid`.

The `<GameId>`

While it is probably possible to give each game an unique `<GameId>` I don’t think that it is sensible or really necessary. I would suggest, that each game gets uses `\newchessgame[id=main\xskakaddtoid,...]` where `\xskakaddtoid` is empty by default but can be redefined to allow some automatic numbering.

The parsing command

As you will probably sometimes use `\mainline` and sometimes `\hidemoves` it is probably best if the converter use an general `\parsechessgame` which can be `\let` to both commands.

NAG’s

Most NAG’s don’t need any special handling as `xskak` can parse them.

But some chess applications seems to put NAG’s *after* a move to denote a symbol that should go *before* the move. E.g. `1. e5 $142` should become `∩1.e5`.

Putting such symbols before a move number is in most cases only possible at the start of a `\mainline` or `\variation` so the converter will have probably to split the input and add the symbol either directly or through a style. A converter will need a list of all this special cases and good rules how to handle them.

Long comments

Longer commands (in the PGN in curly braces) could be handled either by adding a `\xskakcomment` at the begin of the comment or by ending the parsing and starting it again after the comment. The first method has the advantage that one can use styles to format to some extent the comments (or gobble them) but can cause problems if the comment contains something that isn't allowed in the argument of a command.

I think some real examples are needed to decide what is the best way.

The content of long comments

I don't think that is possible to handle all possible contents automatically and get a suitable result for \LaTeX . You will either have to follow some rules while inputing the comments or tidy them up later.

But a converter will probably be able to identify moves like 4. Nf3 and to surround them by suitable `\variation[style=...,level=...]{...}`.

Diagrams

Some chess applications let you put markers at places where you would like to print a diagram. E.g. chessbase put `Diagramm #17` in a comment.

A converter should also be able to recognize such markers for diagrams. But it is not easy to decide what should be done with this markers. In some cases you would probably not want to print boards at all. Sometimes the game should be interrupted and the diagram printed directly. Sometimes it should float. And I recall that someone once tried to put all diagrams in the order there were mentioned in the margin. If such markers are used it is probably the best if the converter sets a tag and sets one of the fake NAG's `$D` and `$d`. Then `\printchessgame` and the list of diagrams could be used to print the moves and the diagrams in the intended order (but it will probably need some coding to get the right results). The tag and the NAG should be move outside of the comment as the comment isn't processed in all cases.

Example 9: Input and suggested output for diagram markers

```
10. c3 e5 $1 {Diagramm # Die energische und konsequente
Fortsetzung, Schwarz öffnet das Zentrum, um seine Figuren zu
aktivieren.} 11. h3
```

```
\mainline{10. c3 e5\xskakset{tag=dial} $d $1
\xskakcomment{ Die energische und konsequente Fortsetzung,
Schwarz öffnet das Zentrum, um seine Figuren zu aktivieren. }
11. h3}
```

¹⁷I don't know if "Diagramm" is language dependent. Probably yes.

Variations

Here starts the real problems. Even if we forget for the moment the parsing of moves and diagrams it is not easy to describe a suitable output. The main problems come from the fact that styles and good typesetting are involved: E.g. PGN code two variations in a row as (1... e5) (1... e6). But you would probably print this as (**1... e5**; **1... e6**).

So the converter will have to look ahead and use suitable “open”, “middle”, “close” and “single” styles in the different combinations.

If you make long analyzes you will probably want to use numbered list for the variations – but not every variation is so important that it need its own number. In this case there must be a marker that tells the converter to handle this variation differently.

I haven't solved all this problems but I think that the following code could be used as starting point for further discussion about a \LaTeX -chess input¹⁸:

```
\newchessgame[id=main, ...]
\mainline{...}
  %new var level
  \newchessgame[newvar=main,id=var1a]%
  \mainline[level=1open]{....}
  %same var level
  \newchessgame[newvar=main,id=var1b]%
  \mainline[level=1close]{....}
  %new var level
  \newchessgame[newvar=var1b,id=var2a]%
  \mainline[level=2single]{....}
%% return to var1b:
\resumechessgame[id=var1b]
\mainline[level=1open]{....}
```

Initialization and title

Until now I have only discussed the game itself. But you would probably want to print titles, make table of contents and other lists. So one should also add some generic commands or environments like `\begin{chessgame}/\end{chessgame}` and `{\chessgameheader}` to each game.

¹⁸If one really wants to parse all variations a better naming for the nodes is needed.

9. Compability issues

9.1. xskak and texmate

It will probably work fine to use both packages in one document.

`\newchessgame` and `\resumechessgame` will both internally issue a `\fenposition` command that will setup the games also for `texmate`. This naturally means in return that both commands interfere with `texmate`!


You can't benefit of the extended input possibilities for NAG's and comments. `texmate` has its own parsing method.

Btw: If you are using `texmate` you should always indent your input as the package doesn't recognize new lines as spaces.

9.2. xskak and beamer

When making presentations with `beamer` you should remember two things: First you can't use verbatim material or other commands that relies on catcode changes in the frame-environment or `-command`. That means for the package `skak` that you can't use `#` in a `\mainline` but should replace it by `\mate`.

Second, if you are using overlays the content of the frame is processed more than once. That means that you should be careful to reset counters if you don't want to run into never ending loops.

The following listing shows an example how to print a chessgame with `beamer`. If your pdf-reader can handle annotations you can see the result in the attached file:  It uses overlays (which worked fine) and also shows how you can handle games in another language:

```
\documentclass{beamer}
\usepackage[LSBC4,T1]{fontenc}
\usepackage{chessboard}
\usepackage{xskak}

\newcommand\getmovestyle[1]{%
\ifthenelse
{\equal{#1}{N}}%knight move
{\def\mymovestyle{[clockwise=false,style=knight]curvemove}}
{\ifthenelse
{\equal{#1}{}}% castling
{\def\mymovestyle{curvemove}}
{\def\mymovestyle{straightmove}}}}
```

```

\setboardfontencoding{LSBC4}
\newskaklanguage{german}{KDTLSB}
\skaklanguage[german]

\begin{document}
\newchessgame[white=Paethz,black=Dirr,result=0-1]
\hidemoves{%
1.d4 Sf6 2.c4 g6 3.Sc3 Lg7 4.e4 d6 5.Sge2 0-0 6.Sg3 c6 7.Le2 a6 8.a4
a5 9.h4 h5 10.Le3 Sa6 11.f3 e5 12.d5 Sd7 13.Sf1 Sdc5 14.Sd2 Db6
15.Db1 Sb4 16.Sb3 Scd3+ 17.Kd2 Dxe3+}

\newcounter{chessmoves}

\begin{frame}{\xskakgetgame{white} -- \xskakgetgame{black}}

\begin{columns}[T]
\column[T]{0.5\textwidth}
\setcounter{chessmoves}{0}%
\xskakloop{%
\getmovestyle{\xskakget{piecechar}}}%
\refstepcounter{chessmoves}%
\only<\arabic{chessmoves}>{%
\chessboard[%
margin=false,showmover=false,inverse,
pgfstyle=\mymovestyle,color=blue,
linewidth=0.1em,pgfshortenend=0.4em,
arrow=to,markmoves=\xskakget{move},
setfen=\xskakget{nextfen},
coloremph,piececolor=blue,
emphfields={\xskakget{moveto}}
]\par
}%
}

\column[T]{0.45\textwidth}%
\renewcommand\baselinestretch{1.2}\footnotesize
\setcounter{chessmoves}{0}%
\xskakloop{%
\refstepcounter{chessmoves}%
{ \color<\arabic{chessmoves}>{blue}%
\uncover<\arabic{chessmoves}>->{%
\ifthenelse{\equal{\xskakget{player}}{w}}

```

```

{\xskakget{opennr}}{\xskakget{lan}}}}
\xskakgetgame{result}
\end{columns}
\end{frame}
\end{document}

```

Index

A	
addpieces (move data type)	22
afterblack (\xskaknewstyleitem key)	41
afterwhite (\xskaknewstyleitem key)	41
B	
beamer	52
beforeblack (\xskaknewstyleitem key)	41
beforenumber (\xskaknewstyleitem key)	41
beforewhite (\xskaknewstyleitem key)	41
black (\newchessgame key)	13
blackelo (\newchessgame key)	13
blackopen (\xskaknewstyleitem key)	41
C	
capture (move data type)	21
castling (move data type)	21
\chessboard	
id (key)	30
lastmoveid (key)	30
moveid (key)	30
movenr (key)	30
player (key)	30
reffen (key)	33
refid (key)	33
refnext (key)	33
refnextmoveid (key)	33
refnextmovenr (key)	33
refnextplayer (key)	33
refpast (key)	33
refpastmoveid (key)	33
refpastmovenr (key)	33
refpastplayer (key)	33
reftag (key)	32
clearfields (move data type)	22
closecommands (\xskaknewstyleitem key) 41	
comments (move data type)	23
D	
date (\newchessgame key)	13
defaultfen (\xskakset key)	28
defaultid (\xskakset key)	27
defaultmoveid (\xskakset key)	28
defaultmovenr (\xskakset key)	28
defaultplayer (\xskakset key)	28
E	
enpassant (move data type)	21
enpassantsquare (move data type)	21
event (\newchessgame key)	13
F	
file (\xskakexportgames key)	23
font (\xskaknewstyle key)	44
G	
game data	
\xskak<GameId>diagramlist	19
\xskak<GameId>gameid	19
\xskak<GameId>initfen	17
\xskak<GameId>initmoveid	18
\xskak<GameId>initmovenr	18
\xskak<GameId>initplayer	18
\xskak<GameId>lastfen	18
\xskak<GameId>lastmoveid	18
\xskak<GameId>lastmovenr	18
\xskak<GameId>lastplayer	18
\xskak<GameId>nextmoveid	19
\xskak<GameId>nextmovenr	19
\xskak<GameId>nextplayer	19
games (\xskakexportgames key)	23
ginvar (\mainline key)	47

ginvar (\printchessgame key)	47	M	
ginvar (\variation key)	47	\mainline	
ginvar (\xskakset key)	47	ginvar (key)	47
glevel (\mainline key)	46	glevel (key)	46
glevel (\printchessgame key)	46	goutvar (key)	47
glevel (\variation key)	46	gstyle (key)	46
glevel (\xskakset key)	46	invar (key)	46
goutvar (\mainline key)	47	level (key)	46
goutvar (\printchessgame key)	47	outvar (key)	47
goutvar (\variation key)	47	style (key)	46
goutvar (\xskakset key)	47	Move data	
gstyle (\mainline key)	46	addpieces	22
gstyle (\printchessgame key)	46	capture	21
gstyle (\variation key)	46	castling	21
gstyle (\xskakset key)	46	clearfields	22
		comments	23
I		enpassant	21
id (\newchessgame key)	12	enpassantsquare	21
id (\printchessgame key)	47	lan	22
id (\resumechessgame key)	15	longcastling	21
id (\xskakloop key)	35	lostpiece	21
id (\xskakset key)	27	lostpiecechar	20
initmoveid (\printchessgame key)	47	movefrom	21
initmoveid (\xskakloop key)	35	moveid (virtual type)	29
initmovenr (\printchessgame key)	47	movenr (virtual type)	29
initmovenr (\xskakloop key)	35	moveto	21
initplayer (\printchessgame key)	47	nag	23
initplayer (\xskakloop key)	35	nextfen	22
invar (\mainline key)	46	opennr	22
invar (\printchessgame key)	46, 48	pastfen	22
invar (\variation key)	46	pgnlostpiece	20
invar (\xskakset key)	46	pgnmovefrom	21
		pgnpiece	20
L		piece	20
lan (move data type)	22	piecechar	20
lastmoveid (\chessboard key)	30	player (virtual type)	29
lastmoveid (\xskakset key)	27	promotion	21
level (\mainline key)	46	promotionpiece	21
level (\printchessgame key)	46, 48	promotionpiecechar	21
level (\variation key)	46	san	22
level (\xskaknewstyle key)	43	movefrom (move data type)	21
level (\xskakset key)	46	moveid (\chessboard key)	30
longcastling (move data type)	21	moveid (\newchessgame key)	12
\longmoves	6, 38, 42	moveid (\resumechessgame key)	15
lostpiece (move data type)	21	moveid (\xskakset key)	27
lostpiecechar (move data type)	20	moveid (virtual move data type)	29

movenr (\chessboard key)	30	outvar (\variation key)	47
movenr (\newchessgame key)	12	outvar (\xskakset key)	47
movenr (\resumechessgame key)	15		
movenr (\xskakset key)	27	P	
movenr (virtual move data type)	29	pastfen (move data type)	22
moveto (move data type)	21	PGN-info data	
N		\Xskak<GameId>black	19
nag (move data type)	23	\Xskak<GameId>blackelo	20
\newchessgame		\Xskak<GameId>date	20
black (key)	13	\Xskak<GameId>event	20
blackelo (key)	13	\Xskak<GameId>result	19
date (key)	13	\Xskak<GameId>round	20
event (key)	13	\Xskak<GameId>site	20
id (key)	12	\Xskak<GameId>white	19
moveid (key)	12	\Xskak<GameId>whiteelo	19
movenr (key)	12	PGN-infos	<i>see</i> (\newchessgame keys)13
newvar (key)	13	pgnlostpiece (move data type)	20
player (key)	12	pgnmovefrom (move data type)	21
reffen (key)	33	pgnpiece (move data type)	20
refid (key)	33	piece (move data type)	20
refnext (key)	33	piecechar (move data type)	20
refnextmoveid (key)	33	player (\chessboard key)	30
refnextmovenr (key)	33	player (\newchessgame key)	12
refnextplayer (key)	33	player (\resumechessgame key)	15
refpast (key)	33	player (\xskakset key)	27
refpastmoveid (key)	33	player (virtual move data type)	29
refpastmovenr (key)	33	\printchessgame	
refpastplayer (key)	33	ginvar (key)	47
reftag (key)	32	glevel (key)	46
result (key)	13	goutvar (key)	47
round (key)	13	gstyle (key)	46
site (key)	13	id (key)	47
using chessboard keys	13	initmoveid (key)	47
white (key)	13	initmovenr (key)	47
whiteelo (key)	13	initplayer (key)	47
\newchessgame – usage	11	invar (key)	46, 48
newvar (\newchessgame key)	13	level (key)	46, 48
newvar (\resumechessgame key)	16	outvar (key)	47, 48
nextfen (move data type)	22	refid (key)	47
nopdfmatch (package option)	7	reftag (key)	48
O		stopmoveid (key)	47
opencommands (\xskaknewstyleitem key) 41		stopmovenr (key)	47
opennr (move data type)	22	stopplayer (key)	47
outvar (\mainline key)	47	style (key)	46, 48
outvar (\printchessgame key)	47, 48	\printchessgame – usage	47
		\xskakloop	
		reftag (key)	32

promotion (move data type)	21	reftag (\printchessgame key)	32, 48
promotionpiece (move data type)	21	reftag (\resumechessgame key)	32
promotionpiecechar (move data type)	21	reftag (\xskakloop key)	32
		reftag (\xskakset key)	32
		result (\newchessgame key)	13
		\resumechessgame	
		id (key)	15
		moveid (key)	15
		movenr (key)	15
		newvar (key)	16
		player (key)	15
		refid (key)	33
		refnext (key)	33
		refnextmoveid (key)	33
		refnextmovenr (key)	33
		refnextplayer (key)	33
		refpast (key)	33
		refpastmoveid (key)	33
		refpastmovenr (key)	33
		refpastplayer (key)	33
		reftag (key)	32
		\resumechessgame – usage	15
		round (\newchessgame key)	13
		S	
		san (move data type)	22
		\shortmoves	6, 42
		showlast (\xskakloop key)	35
		site (\newchessgame key)	13
		step (\xskakloop key)	35
		stepmoveid (\chessboard key)	30
		stepmoveid (\xskakset key)	27
		stopmoveid (\printchessgame key)	47
		stopmoveid (\xskakloop key)	35
		stopmovenr (\printchessgame key)	47
		stopmovenr (\xskakloop key)	35
		stopplayer (\printchessgame key)	47
		stopplayer (\xskakloop key)	35
		style (\mainline key)	46
		style (\printchessgame key)	46, 48
		style (\variation key)	46
		style (\xskakset key)	46
		style item	40
		styleitem (\xskaknewstyle key)	44
		T	
		tag (\xskakset key)	27, 31

template (\xskaknewstyleitem key)	42	\xskakloop	
texmate	52	id (key)	35
		initmoveid (key)	35
		initmovenr (key)	35
		initplayer (key)	35
		refid (key)	33
		reftag (key)	32
		showlast (key)	35
		step (key)	35
		stopmoveid (key)	35
		stopmovenr (key)	35
		stopplayer (key)	35
		\xskakloop – usage	35
		\xskaknewpgninfo – usage	13, 19
		\xskaknewstyle	
		font (key)	44
		level (key)	43
		styleitem (key)	44
		xfont (key)	44
		\xskaknewstyle – usage	42
		\xskaknewstyleitem	
		afterblack (key)	41
		afterwhite (key)	41
		beforeblack (key)	41
		beforenumber (key)	41
		beforewhite (key)	41
		blackopen (key)	41
		closecommands (key)	41
		opencommands (key)	41
		template (key)	42
		whiteopen (key)	41
		\xskaknewstyleitem – usage	41
		\Xskak<GameId>nextmoveid	19
		\Xskak<GameId>nextmovenr	19
		\Xskak<GameId>nextplayer	19
		\Xskak<GameId>result	19
		\Xskak<GameId>round	20
		\chessboard	
		stepmoveid (key)	30
		\xskakset	
		defaultfen (key)	28
		defaultid (key)	27
		defaultmoveid (key)	28
		defaultmovenr (key)	28
		defaultplayer (key)	28
		ginvar (key)	47
template (\xskaknewstyleitem key)	42		
texmate	52		
		V	
\variation			
ginvar (key)	47		
glevel (key)	46		
goutvar (key)	47		
gstyle (key)	46		
invar (key)	46		
level (key)	46		
outvar (key)	47		
style (key)	46		
		W	
white (\newchessgame key)	13		
whiteelo (\newchessgame key)	13		
whiteopen (\xskaknewstyleitem key)	41		
		X	
xfont (\xskaknewstyle key)	44		
\xskakaddtostyle – usage	42		
\Xskak<GameId>black	19		
\Xskak<GameId>blackelo	20		
\xskakcurrentgameid – usage	24		
\Xskak<GameId>date	20		
\Xskak<GameId>diagramlist	19		
\xskakendgamedata – usage	24		
\xskakenpassanttext – usage	39		
\Xskak<GameId>event	20		
\xskakexportgames			
file (key)	23		
games (key)	23		
\xskakexportgames – usage	23		
\Xskak<GameId>gameid	19		
\xskakget (with \chessboard)	30		
\xskakget – usage	29		
\xskakgetgame – usage	28		
\Xskak<GameId>initfen	17		
\Xskak<GameId>initmoveid	18		
\Xskak<GameId>initmovenr	18		
\Xskak<GameId>initplayer	18		
\Xskak<GameId>lastfen	18		
\Xskak<GameId>lastmoveid	18		
\Xskak<GameId>lastmovenr	18		
\Xskak<GameId>lastplayer	18		

glevel (key)	46	refnextplayer (key)	27, 33
goutvar (key)	47	refpast (key)	27, 33
gstyle (key)	46	refpastmoveid (key)	33
id (key)	27	refpastmoveid(key)	27
invar (key)	46	refpastmovenr (key)	27, 33
lastmoveid (key)	27	refpastplayer (key)	27, 33
level (key)	46	reftag (key)	32
moveid (key)	27	stepmoveid (key)	27
movenr (key)	27	style (key)	46
outvar (key)	47	tag (key)	27, 31
player (key)	27	\xskakset – usage	27
refid (key)	27, 33	\Xskak<GameId>site	20
refnext (key)	27, 33	\xskaktestmoveid – usage	34
refnextmoveid (key)	33	\Xskak<GameId>white	19
refnextmoveid(key)	27	\Xskak<GameId>whiteelo	19
refnextmovenr (key)	27, 33		