The `guitar` package*

Martin Väth†‡

2009/03/17

Abstract

This package allows you to typeset guitar chords over song texts. This package requires the `toolbox` package. To typeset graphical chords, you need an additional package like e.g. `gchords.sty`, available from `http://www.damtp.cam.ac.uk/user/kp229/gchords/`

This is not the only package available for typesetting guitar songs. The following packages are also available (and possibly also others: I do not claim that this list is complete).

1. `gchords.sty`; `http://www.damtp.cam.ac.uk/user/kp229/gchords/`
2. `songbook.sty`; `http://www.rath.ca/Misc/Songbook/`
   (or on CTAN: `macros/latex/contrib/supported/songbook/`)
3. guitarTEX (which is not a 'pure' (La)TEXsolution):
   `http://rz-home.de/~jmiltz/guitartex/`

The packages 1. and 2. have somewhat different intentions. It should be possible to combine them with the current package without any severe problems.

Contents

*This file has version number 1.6, last revised 2009/03/17.

†`vaeth@mathematik.uni-wuerzburg.de`

‡The author thanks Donald Arseneau `<asnd@triumf.ca>`, Dan Luecking `<luecking@uark.edu>`, and a colleague who does not want to be named here.

1 Changes

v1.6 (*2009/03/17*) Do not redefine commands on new catcodes globally. Instead save and restore the original definition when changing catcodes. Thanks to Heiko Oberdiek for pointing out this bug.

v1.5 (*2001/08/27*) Included references to other packages in documentation; slightly modified the `\typeout` of the package name.

v1.4 (*2001/08/27*) Introduced `*` versions to avoid changing of catcodes in special situation.

v1.3 (*2001/08/24*) Improved technique of right-alignment thanks to valuable hints of Donald Arseneau `<asnd@triumf.ca>`. Introduced `\` and made `\guitarFirstFlush` the default.

v1.2 (*2001/08/21*) Right-alignment of broken lines. Added remarks about catcode-changes in the documentation.

v1.1 (*2001/08/19*) First release.

2 Installation

This package should run with plain TEX, LATEX2.09, and LATEX 2_ε. Actually, it should actually run with all TEX formats.

To use `guitar`, you have to put the file `guitar.sty` in a path where TEX looks for its input files. The TEX documents using `guitar` need the following modifications in their header:

- If you use LATEX 2_ε, put in the preamble the command

  ```
  \usepackage{guitar}
  ```

- If you use LATEX2.09, use `guitar` as a style option, e.g.

  ```
  \documentstyle[guitar]{article}
  ```

  or

  ```
  \documentstyle[guitar,12pt]{article}
  ```

- If you use some other (non-LATEX) format, you will probably have to insert a line like

  ```
  \catcode`\@=11\input guitar.sty\catcode`\@=12\relax
  ```

For TEX-insiders: The only LATEX-specific commands used in `guitar.sty` are:

- `\newenvironment`
- `\newcommand` (used only in the form `\newcommand{`⟨*command*⟩`}{}` to ensure that ⟨*command*⟩ was not defined before)
- `\newsavebox`
- `\newlength`
- `\RequirePackage`
- `\ProvidesPackage`
- `\typeout`

The above commands are used only if they are defined (otherwise, the plainTEX substitutes are used).

3 Example and Usage

`\guitarChord` The general usage is very simple:

```
\guitarChord{Bb}Really. \guitarChord{G#}Oh yes.
```

produces the output

B♭ G♯
Really. Oh yes.

(in the chord, `b` and `#` always translate into ♭ resp. ♯). When you switch into "magic" mode, you can even use `[...]` instead of `\guitarChord{...}`. E.g., in `guitar` LATEX, you can just write

```
\begin{guitar}
[Cm]This [Bb]is a [G#]very [Gm]simple song.
You can still use any [\TeX]macros in the chord text.
Actually, each accord reads a second argument over which it is [A]{centered}.
By default, the linefeeds in the source lead to linebreaks %
in the output, unless you put a percentage sign at the end of the line. %
If a single line gets too long, the wrapped lines are right-aligned.
To force the wrapping use \verb|\newline|\newline like here.
To stretch the first line, use \verb|\linebreak|\linebreak like here.

Two subsequent linefeeds in the source start a new verse.


Three or more subsequent linefeeds in the source start a new verse with a %
slightly larger space on top.
\end{guitar}
```

This will produce the output

Cm B♭ G♯ Gm
This is a very simple song.
You can still use any TEX macros in the chord text.
Actually, each accord reads a second argument over which it is A centered.
By default, the linefeeds in the source lead to linebreaks in the output, unless you put a percentage sign at the end of the line. If a single line gets too long, the wrapped lines are right-aligned.
To force the wrapping use `\newline` like here.
To stretch the first line, use `\linebreak` like here.

Two subsequent linefeeds in the source start a new verse.

Three or more subsequent linefeeds in the source start a new verse with a slightly larger space on top.

Unfortunately, if the chords get too long, they may overlap. For example, `[Ebm+7]O[G#]K` produces the unreadable output E♭mG♯+7OK. To avoid this, the following solutions are supported:

1. If the token after your chord is a space, like in "`a[Ebm+7] space`", this space is stretched: a E♭m+7 space. Note that this is different from the notation `a[Ebm+7]{ }space` which produces instead: a E♭m+7 space.

2. Inside words, you can put a `_` at the end like for `[Ebm+7_]O[G#]K`. This stretches the word: E♭m+7 G♯O____K.

3. Outside words, you can put a `|` at the end like for `[Ebm+7|]{OK}[G#|].[C#m]␣`: E♭m+7 G♯ C♯m
OK .

If the chord is short enough (compared to its argument) then the symbols `|` and `_` do not harm: `[E|]{long}` and `[E_]{long}er` produce E long and E longer.

`\guitarOn` `\guitarOff`

You may (and in plainTEX you must) replace `\begin{guitar}` and `\end{guitar}` by `\guitarOn` and `\guitarOff`, respectively.

If you only want to put the guitar chords as above but do not like the feature concerning the linefeeds, you can use the environment

```
\begin{guitarMagic}
\end{guitarMagic}
```

guitarMagic
\guitarMagicOn
\guitarMagicOff

or the respective commands \guitarMagicOn and \guitarMagicOff. Similarly, you can also use only the linefeed feature by the environment

```
\begin{guitarCr}
\end{guitarCr}
```

guitarCr
\guitarCrOn
\guitarCrOff

or the respective commands \guitarCrOn and \guitarCrOff.

As a matter of fact, you can also e.g. use \guitarCrOff *within* a {guitar} environment, but then you have to use \guitarCrOn before you end the environment to have a proper nesting of modes.

4 Customization

If you want that the first line in a broken phrase is always left-aligned (i.e. the spaces between words are not enlarged to fill the line), enter before your line the command

\guitarFirstLeft

```
\guitarFirstLeft
```

You can switch back into the default mode with

\guitarFirstLeft

```
\guitarFirstFlush
```

The output of the # and b tokens in the chords is generated by the two commands

\guitarSharp
\guitarFlat

```
\guitarSharp
\guitarFlat.
```

Thus, for, example the customization

```
\def\guitarSharp{$^{\sharp}$}
```

will make the chord [F#] look like $F^{\sharp}$ (instead of the default $F\sharp$).

In a similar way, the macros

\guitarEndLine
\guitarEndPar
\guitarEndDoublePar

```
\guitarEndLine
\guitarEndPar
\guitarEndDoubePar
```

are expanded when the song text contains one, two, or three (or more) linefeeds.

After the macro

\guitarNoChord

```
\guitarNoChord
```

has been used, no chords are output anymore. You can make the effect local by putting this command in a group. (\guitarNoChord redefines the macros which are responsible for the actual output of the chords – see the description at the end).

The command

\guitarPreAccord

```
\guitarPreAccord
```

is expanded before the actual chord is typeset: You can change this definition to e.g. modify the font used for the chord. By default, this command contains a `\strut` which ensures that the lines with chords usually all have the same height. The macro `\guitarPreAccord` may also use and modify the macro

`\guitarAccord` `\guitarAccord`

which contains the actual chord which is then output (you may use this feature to e.g. calculate a modulation in `\guitarPreAccord` or to replace the text of `\guitarAccord` by some graphic equivalent).

It is guaranteed that the macros `\guitarPreAccord` and `\guitarAccord` are expanded precisely once. In contrast, it might happen that the argument of the chord (i.e. the text over which the chord is put) is expanded twice. (The doubled expansion could have been avoided, but the price were that in some cases the TEX kerning mechanism would then fail if the argument is a syllable of some word which is continued afterwards).

Whenever the "guitar" mode or the "linefeed" mode is turned on or off, the corresponding macro

`\guitarMagicOnHook` `\guitarMagicOnHook`
`\guitarMagicOffHook` `\guitarMagicOffHook`
`\guitarCrOnHook` `\guitarCrOnHook`
`\guitarCrOffHook` `\guitarCrOffHook`

is expanded (for switching on, the macro is expanded even before the necessary catcode changes are done, and for switching off, the macro is expanded after the catcodes have been restored). In the current defaults, a group is opened and closed in `\guitarCrOnHook` and `\guitarCrOffHook`: This is the reason why these commands must be properly nested. (The reason for this group is that `\parindent` should be changed locally).

In the current implementation, TEX forgets everything following the magic tokens `|` and `_` in the chord (because these symbols are intended to be the last one in the chord). If you want to use these tokens within a chord: They loose their magic meaning if they occur in a braced context (i.e. between `{` and `}`). If you regularly have to use these tokens within chords, you can change the default choice of this tokens by the respective commands

`\guitarSplitDist` `\toolboxMakeSplit{`⟨*token*⟩`}{guitarSplitDist}`
`\guitarSplitMerge` `\toolboxMakeSplit{`⟨*token*⟩`}{guitarSplitMerge}`.

`\toolboxMakeSplit` (See the `toolbox` package for details of the command `\toolboxMakeSplit` used here). It should be pointed out that ⟨*token*⟩ may actually be a *sequence* of tokens, so that for the 'magic tokens' `|` and `_` you may instead use whole phrases if you prefer.

Some fine tuning of the spacing can be done by redefining

`\guitarCalcDim` `\guitarCalcDim`.

When this macro is called, the skip register

\guitarDim `\guitarDim`

contains the width of the chord which should be put over the text. After `\guitarCalcDim` has been expanded, it is ensured that at least `\guitarDim` (horizontal) space is reserved for the chord. Thus, if `\guitarCalcDim` increases e.g. `\guitarDim` by `2pt` (which is the default behavior of `\guitarCalcDim`), then `2pt` more space is reserved for the chord than its actual size: This ensures that two subsequent chords are always separated by some space.

Finally, you can modify the macros which do the actual setting of the chords: This setting is done by the commands

\guitarPut `\guitarPut`
\guitarPutOnSpace `\guitarPutOnSpace`
\guitarPutDist `\guitarPutDist`
\guitarPutMerge `\guitarPutMerge`

which with the exception of `\guitarPutOnSpace` all expect one argument (namely the text on which they should be put). The command `\guitarPutOnSpace` is called, if a space followed the chord argument. Similarly, `\guitarPutDist` and `\guitarPutMerge` are called if the chord argument ended with the magic symbol `|` respectively `_`.

Before one of the above four macros is called, the chord text is stored in the macro `\guitarAccord` (and has not been expanded to this time); also the argument has not been expanded to this time. The above four macros are also responsible for the appropriate expansion (also of the expansion of the macros `\guitarCalcDim` and `\guitarPreAccord`, if the latter should still keep their meaning). The above four macros may freely use the skip register `\guitarDim` and the box register `\guitarBox` (which are both not used or modified by the other macros of this package).

\guitarCalcDim
\guitarPreAccord
\guitarDim
\guitarBox

5 Possible Problems

The special treatment of `[ # b` and ⟨*linefeed*⟩ is realized by a changing of catcodes. This may cause essentially two sorts of problems:

1. In some cases, you will want to avoid the automatic replacement of `#` and `b`. For example, in `[\textbf{F#}]` or in `\guitarChord{\textbf{F#}}` the `b` of `\textbf` gets replaced which is of course not what you want. To avoid this replacement, there is a `*` version of the respective commands, i.e. the above example works in the intended way if you write instead `[*\textbf{F\guitarSharp}]` or `\guitarChord*{\textbf{F\guitarSharp}}` (you have to use the command `\guitarSharp`, because due to the `*` also the `#` would not be replaced).

2. There are some situations in which catcodes are set too early, e.g. if you use the macros within the argument of certain macros. In this case, these symbols keep their usual TEX meaning (which may either lead to an error

in case of `#` (and sometimes `[`) or just to a false output). If this happens to you, you have to replace at the corresponding place the above one-symbol shortcuts by their longer macro equivalents. In other words: You might have to replace in certain special situations some occurrences of the symbols according to the following table:

`[...]`	→	`\guitarChord{...}`
`#`	→	`\guitarSharp`
`b`	→	`\guitarFlat`
linefeed	→	`\guitarEndLine`
2 linefeeds	→	`\guitarEndPar`
3+ linefeeds	→	`\guitarEndDoublePar`