# The songs package[*]

Kevin W. Hamlen

January 31, 2009

**Abstract**

The songs package produces books of songs that contain lyrics and chords but not sheet music. Its primary contribution is to allow lyric books, chord books, and books of overhead slides to all be maintained and generated from a single LaTeX source document. Additionally, one can automatically extract a subset of songs in a specified order to create handouts, automatically transpose chords to new keys, and manually create guitar tablature diagrams.

## 1 Introduction

The songs LaTeX package is designed to produce books of songs that contain lyrics and (optionally) chords, but not sheet music. By changing only one line of the LaTeX source document, one can generate a lyric book for singers, a chord book for musicians, or a book of overhead slides for corporate singing. In addition, for each one of these book styles, a one-line change to the source document can be used to extract only certain songs from the book in a specified order. This allows easy creation of handouts or slide sets from a larger master document.

Religious worship styles are becoming increasingly independent and self-driven in modern times, and with this trend have come difficult challenges for creating and maintaining printed material suitable for these venues. Christian denominations, for example, have seen the rise of the so-called "home church" movement, in which worshippers meet on a small scale in many different locales that vary from week to week. This has resulted in worship settings where instrumental accompaniment, if any, often consists solely of portable instruments like guitars, which typically play chords rather than full sheet music. In addition, sacred music has become more contemporary and more fluid than was typical of past eras. Congregations are less willing to accept a fixed book of songs like a hymnal, and rather prefer to have a constantly changing repertoire of music to which they can add and remove songs over time.

Typesetting material suitable for these settings is a challenging endeavor. Rather than producing a single book that remains static, worship coordinators must be able to create and maintain evolving collections of music that can be quickly arranged for specific events or services. Licensing restrictions and printing costs also make it desirable for these collections to simultaneously exist in multiple

[*]This document corresponds to songs v2.7, dated 2009/01/08, © 2009 Kevin W. Hamlen, and distributed under version 2 the GNU General Public License as published by the Free Software Foundation.

forms—as lyric books, as chord books, and as overhead slides—all of which must be maintained over time to be consistent with one another.

The songs LaTeX package is one attempt at meeting these demands. The LaTeX document publishing system allows beautiful documents to be generated mostly automatically according complex style rules, such as those demanded by poetry and music. The songs package facilitates the use of LaTeX to generate song books by providing an extensive set of LaTeX macros that handle many of the difficult aspects of arranging songs on a page. These macros ensure that chords remain placed above appropriate syllables as songs shift position in the book, that songs will continue to be placed in appropriate locations and with aesthetically pleasing spacing as song ordering changes, and that book indexes remain updated as new songs get introduced. In addition, simple facilities for automatically transposing songs or indexing songs by scripture reference are also provided.

## 2 Terms of Use

`http://songs.sourceforge.net`

## 3 Sample Document

The following sections of this document provide a detailed explanation of the songs package, its usage, and its implementation. However, for those who would like to start making song books quickly, the following is a sample document that yields a simple song book with one song and one title index. Starting from this template, you can begin to add songs and customizations to create a larger book. Instructions for compiling this sample song book follow the listing.

```
\documentclass{article}
\usepackage[chorded]{songs}

\newindex{titleidx}{titleidx}
\noversenumbers

\begin{document}
\showindex{Complete Index of Songs}{titleidx}
\songsection{Worship Songs}
```

```
\begin{songs}{titleidx}
\beginsong{Doxology}[by={Louis Bourgeois and Thomas Ken},
                     sr={Revelation 5:13},
                     cr={Public domain.},
                     index={Praise God, from Whom all blessings flow}]
\beginverse
\[G]Praise God, \[D]from \[Em]Whom \[Bm]all \[Em]bless\[D]ings \[G]flow;
\[G]Praise Him, all \[D]crea\[Em]tures \[C]here \[G]be\[D]low;
\[Em]Praise \[D]Him \[G]a\[D]bove, \[G]ye \[C]heav'n\[D]ly \[Em]host;
\[G]Praise Fa\[Em]ther, \[D]Son, \[Am]and \[G/B G/C]Ho\[D]ly \[G]Ghost.
\[C]A\[G]men.
\endverse
\endsong
\end{songs}

\end{document}
```

To compile this book, you would need to execute three commands. First, use LaTeX (`pdflatex` is recommended) to compile the document:

```
pdflatex mybook.tex
```

(where `mybook.tex` is the name of the source document above). Next, use the `songidx` program provided with this distribution to generate the indexes:

```
songidx titleidx.sxd titleidx.sbx
```

Finally, regenerate the document using LaTeX so that the newly generated index data will be included:

```
pdflatex mybook.tex
```

The final document will be named `mybook.pdf` if you use `pdflatex` or `mybook.dvi` if you use regular `latex`.

A copy of the first page of a sample song section is shown in Figure 1. The page shown in that figure is from a chorded version of the book. When generating a lyric version, the chords would be omitted. See §4 for information on how to generate different versions of the same book.

## 4 Initialization and Options

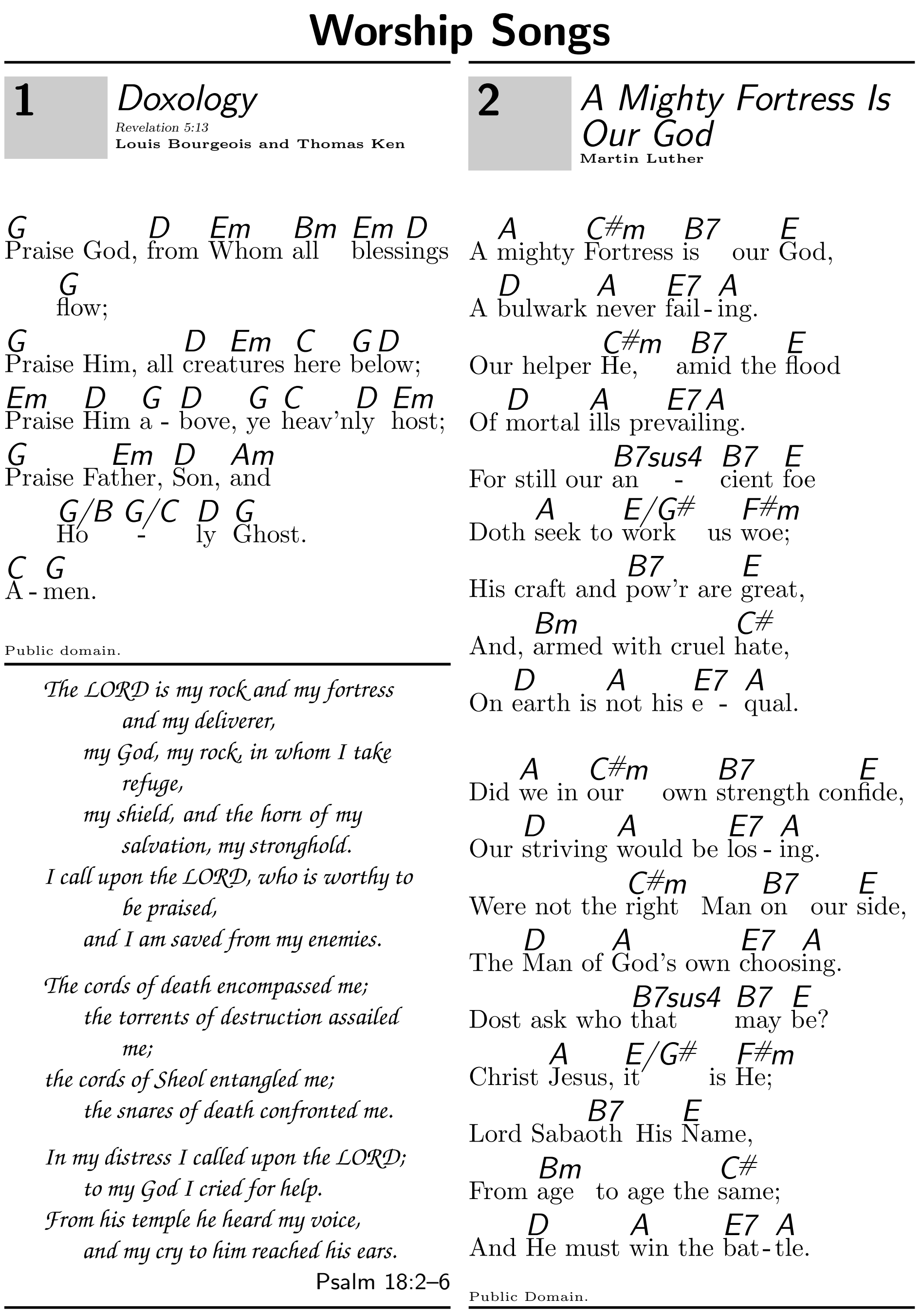
Each LaTeX document that uses the songs package should contain a line like the following near the top of the document:

`\usepackage[`⟨*options*⟩`]{songs}`

Supported ⟨*options*⟩ include the following:

`lyric` `chorded` `slides` `rawtext`

**Output Type.** The songs package can produce four kinds of books: lyric books, chord books, books of overhead slides, and raw text output. You can specify which kind of book is to be produced by specifying one of `lyric`, `chorded`, `slides`, or `rawtext` as an option. If none of these are specified, `chorded` is the default.



# Worship Songs

## 1 Doxology

*Revelation 5:13*

**Louis Bourgeois and Thomas Ken**

```
G            D     Em    Bm  Em D
Praise God, from Whom all    blessings
     G
     flow;
G                D   Em   C   G D
Praise Him, all creatures here below;
Em   D    G  D     G  C    D  Em
Praise Him a - bove, ye heav'nly  host;
G        Em  D    Am
Praise Father, Son, and
     G/B G/C  D  G
     Ho    -  ly Ghost.
C  G
A - men.
```

Public domain.

*The LORD is my rock and my fortress*
*and my deliverer,*
*my God, my rock, in whom I take refuge,*
*my shield, and the horn of my salvation, my stronghold.*
*I call upon the LORD, who is worthy to be praised,*
*and I am saved from my enemies.*

*The cords of death encompassed me;*
*the torrents of destruction assailed me;*
*the cords of Sheol entangled me;*
*the snares of death confronted me.*

*In my distress I called upon the LORD;*
*to my God I cried for help.*
*From his temple he heard my voice,*
*and my cry to him reached his ears.*

Psalm 18:2–6

## 2 A Mighty Fortress Is Our God

**Martin Luther**

```
 A       C#m     B7     E
A mighty Fortress is   our God,
 D       A     E7  A
A bulwark never fail-ing.
             C#m   B7     E
Our helper He,    amid the flood
  D      A      E7 A
Of mortal ills prevailing.
               B7sus4   B7   E
For still our an    -    cient foe
     A      E/G#      F#m
Doth seek to work   us woe;
               B7        E
His craft and pow'r are great,
     Bm             C#
And, armed with cruel hate,
   D       A       E7  A
On earth is not his e  -  qual.

   A     C#m       B7         E
Did we in our    own strength confide,
  D       A          E7  A
Our striving would be los-ing.
                C#m       B7     E
Were not the right  Man on  our side,
  D        A          E7   A
The Man of God's own choosing.
                 B7sus4  B7  E
Dost ask who that       may be?
      A     E/G#     F#m
Christ Jesus, it     is He;
         B7     E
Lord Sabaoth  His Name,
   Bm             C#
From age   to age the same;
   D        A        E7  A
And He must win the bat-tle.
```

Public Domain.

Figure 1: Sample page from a chord book

Lyric books omit all chords, whereas chord books include chords and additional information for musicians (specified using `\musicnote`). Books of overhead slides omit all chords like lyric books, but they typeset one song per page in a large font, centered.

Raw text output doesn't produce songs in the output document at all. Instead, when raw text output is selected, an ascii text file named ⟨*jobname*⟩`.txt` (where ⟨*jobname*⟩ is the filename given by `\jobname`) will be generated in the style of a lyric book. This can be useful for importing song books into another program, such as a spell-checker.

`\chordson` `\chordsoff` In addition to using the `lyric` and `chorded` options to turn chords on or off at the beginning of the document, chords can also be turned on or off anywhere in the middle of the document by using the `\chordson` or `\chordsoff` macros.

`\slides` In addition to using the `slides` option to produce an entire book of overhead slides, one can also activate slides mode using the `\slides` command. For best results, this should typically only be done in the document preamble or at the beginning of a fresh page.

`nomeasures` `showmeasures` `\measureson` `\measuresoff` **Measure Bars.** Even though the songs package does not support generation of full sheet music, it does include a facility for placing measure bars in addition to chords in chord books. To omit these measure bars, use the `nomeasures` option. To display measure bars, use the `showmeasures` option. (This is the default.) Measure bars can also be turned on or off in the middle of the document by using the `\measureson` or `\measuresoff` macros.

`transposecapos` **Transposition.** The `transposecapos` option changes the effect of the `\capo` macro. Normally, using `\capo{`⟨$n$⟩`}` within a song environment produces a textual note in chord books that suggests the use of a guitar capo on fret ⟨$n$⟩. However, when the `transposecapos` option is active, these textual notes will be omitted and instead the effect of `\capo{`⟨$n$⟩`}` will be the same as for `\transpose{`⟨$n$⟩`}`. That is, chords between the `\capo` macro and the end of the song will be automatically transposed up by ⟨$n$⟩ half-steps. This can be useful for adapting a chord book for guitarists to one that can be used by pianists, who don't have the luxury of using a capo. See §7.6 and §10 for more information on the `\capo` and `\transpose` macros.

`noindexes` `\indexeson` `\indexesoff` **Indexes.** The `noindexes` option suppresses the typesetting of any in-document indexes. Display of indexes can also be turned on or off using the `\indexeson` and `\indexesoff` macros. If indexes are off by the time the `\begin{document}` line is reached, then not only are in-document indexes not displayed, the auxiliary data files that are used to create them will not be generated either.

`nopdfindex` The `nopdfindex` option suppresses the creation of the pdf bookmark index that is normally included in `.pdf` files. If not generating a `.pdf` file, this option has no effect.

`onesongcolumn` `twosongcolumns` `\songcolumns` **Columns.** By default, songs in a `songs` environment will be typeset in two columns per page. To force one column per page, you can use the `onesongcolumn` option. To force the default of two columns per page, use the `twosongcolumns` option. The `\songcolumns{`⟨$n$⟩`}` macro can be used anywhere outside of `songs` environments to cause songs to be typeset in ⟨$n$⟩ columns per page (where ⟨$n$⟩

is any positive integer). Setting the number of columns to 1 will cause indexes to be typeset in a single column as well; otherwise indexes will be typeset in the index-default number of columns.

`noscripture` **Scripture Quotations.** The `noscripture` option omits scripture quotations (see §8) from the output.

`\scriptureon` `\scriptureoff` You can also turn scripture quotations on or off in the middle of the document by using `\scriptureon` or `\scriptureoff`, respectively.

`noshading` **Shaded Boxes.** The `noshading` option causes all shaded boxes, such as those that surround song numbers and textual notes, to be omitted. You might want to use this option if printing such shaded boxes causes problems for your printer or uses too much ink.

`\includeonlysongs` **Partial Song Sets.** Often it is useful to be able to extract a subset of songs from the master document—e.g. to create a handout or set of overhead slides for a specific worship service. To do this, you can type `\includeonlysongs{`⟨*songlist*⟩`}` in the document preamble (i.e. before the `\begin{document}` line), where ⟨*songlist*⟩ is a comma-separated list of the song numbers to include in the resulting document. For example, suppose your song book contains three song sections, one in which the songs are numbered with regular arabic numbers, one in which songs are numbered H1, H2, etc., and one in which songs are numbered C1, C2, etc. Then if you put

```
\includeonlysongs{37,H2,2,C4,H1}
```

in the preamble of your document, the first song section of the resulting document would contain only songs 37 and 2 (in that order), the second section would have only songs H2 and H1 (in that order), and the final section would have only song C4.

Partial books generated with `\includeonlysongs` will omit all scripture quotations (§8), and will ignore uses of the `\nextcol` macro. To force a column-break at a specific point in a partial book, add the word `nextcol` at the corresponding point in the ⟨*songlist*⟩ argument. To force a page-break, use consecutive column-breaks.

The `\includeonlysongs` macro cannot be used in conjunction with the `rawtext` document option.

## 5 Book Sections

`\songsection` **Section Titles.** Section titles in a song book can be produced with

`\songsection{`⟨*title*⟩`}`

which acts like LaTeX's `\section` command except that it centers the ⟨*title*⟩ text in sans serif font and omits the section number without excluding the section from indexes or tables of contents. Authors can redefine the `\songsection` command to affect the titles of index sections (see below).

`\songchapter` When using the `book` document class, you can use `\songchapter` instead of `\songsection` to start a new chapter. Likewise, you can redefine `\songchapter` instead of `\songsection` to affect the titles of indexes (see below).

\newauthorindex
\newindex
\newscripindex

**Indexes.** The songs in song sections can be itemized in indexes whose contents are generated automatically. To generate an index, first declare the index in the document preamble (i.e. before the `\begin{document}` line) with one of the following:

```
\newindex{⟨id⟩}{⟨filename⟩}
\newauthorindex{⟨id⟩}{⟨filename⟩}
\newscripindex{⟨id⟩}{⟨filename⟩}
```

which declare an index that will be sorted by song title, an index that will be sorted by author, or an index that will be sorted by scripture references, respectively. ⟨*id*⟩ should be an alphabetic identifier that will be used to identify the index in other macros that reference it. ⟨*filename*⟩ should be a string that, when appended with an extension, constitutes a valid filename on the system. Auxiliary files named ⟨*filename*⟩`.sxd` and ⟨*filename*⟩`.sbx` will be generated during the automatic index generation process.

\showindex

To display an index that was declared in the preamble, use:

```
\showindex{⟨title⟩}{⟨id⟩}
```

where ⟨*id*⟩ is the same identifier used in the `\newindex`, `\newauthorindex`, or `\newscripindex` command, and where ⟨*title*⟩ is the title of the index, which should consist only of simple text suitable for inclusion in the pdf bookmark index. This will display the complete index starting on a fresh page, including its automatically generated contents.

## 6 Compiling

As with a typical LaTeX document, compiling a song book document requires three steps. First, use LaTeX (`pdflatex` is recommended) to generate auxiliary files from the `.tex` file:

```
pdflatex mybook.tex
```

Second, use the `songidx` program to generate an index for each index that you declared with `\newindex`, `\newauthorindex`, or `\newscripindex`. The syntax of the `songidx` command is:

```
songidx [-b ⟨canon⟩.can] ⟨filename⟩.sxd ⟨filename⟩.sbx
```

where ⟨*filename*⟩ is the same ⟨*filename*⟩ that was used in the `\newindex`, `\newauthorindex`, or `\newscripindex` macro. If the index was declared with `\newscripindex`, then the `-b` option is used to specify which version of the bible you wish to use as a basis for sorting your scripture index. The ⟨*canon*⟩ part can be any of the `.can` files provided with the `songidx` distribution. If you are using a Protestant, Catholic, or Greek Orthodox Christian bible with book names in English, then the `bible.can` canon file should work well. If you are using a Jewish Tanakh, use `tanakh.can`. For other bibles, you should create your own `.can` file by copying and modifying one of the existing `.can` files.

For example, if your song book `.tex` file contained the lines

```
\newindex{titleidx}{titlfile}
\newauthorindex{authidx}{authfile}
\newscripindex{scripidx}{scrpfile}
```

then the commands to generate indexes sorted according to a Christian English bible would be:

```
songidx titlfile.sxd titlfile.sbx
songidx authfile.sxd authfile.sbx
songidx -b bible.can scrpfile.sxd scrpfile.sbx
```

Once the indexes are generated, you can generate the final book by invoking LaTeX one more time:

```
pdflatex mybook.tex
```

## 7 Songs

`songs` **Song Sets.** Songs are contained within `songs` environments. Each such environment begins and ends with:

```
\begin{songs}{⟨indexes⟩}
⋮
\end{songs}
```

⟨*indexes*⟩ is a comma-separated list of index identifiers (the ⟨*id*⟩'s specified with `\newindex`)—one identifier for each index that is to include entries for songs in this song set. Between the `\begin{songs}` and `\end{songs}` lines of a song section can appear only songs (see below) or scripture quotations (see §8). No text in a `songs` environment can lie outside of a song or scripture block.

`\beginsong` `\endsong` **Songs.** A song begins and ends with:

```
\beginsong{⟨titles⟩}[⟨otherinfo⟩]
⋮
\endsong
```

Songs should appear within `songs` environments (see above). If they do not, the vertical material comprising the song will be output directly to the current vertical list, and it is up to the enclosing environment to provide suitable page-breaking and other formatting.

In the `\beginsong` line, ⟨*titles*⟩ can be either a single song title or multiple song titles separated by `\\`. If multiple titles are provided, the first is typeset normally atop the song and the rest are each typeset in parentheses on separate lines. An index entry will be generated for each of these song titles, and it will be added to each title index associated with the current `songs` environment.

The ⟨*otherinfo*⟩ is optional; it and its surrounding brackets (`[]`) can be omitted. If provided, it is a comma-separated list of key-value pairs (keyvals) of the form ⟨*key*⟩=⟨*value*⟩. Each keyval provides some information about the song. The possible keys and their values are:

| | |
|---|---|
| `by={⟨authors⟩}` | *cite authors, composers, and other contributors* |
| `cr={⟨copyright⟩}` | *provide copyright information* |
| `li={⟨license⟩}` | *provide licensing information* |
| `sr={⟨refs⟩}` | *list related scripture references* |
| `index={⟨lyrics⟩}` | *add an index entry consisting of a line of lyrics* |
| `ititle={⟨title⟩}` | *add an index entry for an alternate title* |

For example, a song that begins and ends with

```
\beginsong{Title1 \\ Title2}[by={Joe Smith}, sr={Job 3},
  cr={\copyright~2009 XYZ.}, li={Used with permission.}]
\endsong
```

will look like

**1** *Title1*
*(Title2)*
*Job 3*
**Joe Smith**

© 2009 XYZ. Used with permission.

The four keyvals used in the above example are described in detail in the remainder of this section; the final two are documented in §7.7. You can also create your own keyvals (see §12.2).

`by=` **Song Authors.** The authors of a song can be specified with the keyval `by={`⟨*authors*⟩`}`, where ⟨*authors*⟩ are one or more authors, composers, translators, etc. An entry will be added to each author index associated with the current `songs` environment for each contributor listed. Contributors are expected to be separated by commas, semicolons, or the word `and`. For example:

```
by={Fred Smith, John Doe, and Billy Bob}
```

`cr=` **Copyright Info.** Copyright info for a song is provided by `cr={`⟨*copyright*⟩`}`, where ⟨*copyright*⟩ is material that identifies the copyright-holder of the song, if any. This typically begins with the © symbol produced with `\copyright`. For example:

```
cr={\copyright~2000 ABC Songs, Inc.}
```

Note that licensing information that typically appears immediately after the copyright info is *not* usually included here. That information is typically set with the `li=` keyval (see below). Copyright information will be typeset in fine print at the bottom of the song text.

`li=` `\setlicense` **Licensing Info.** Licensing information citing the terms of your lawful use of a song is provided by `li={`⟨*license*⟩`}`, where ⟨*license*⟩ is typically material that a copyright administrator requires licensees to place near each song covered by the license. Licensing information will be displayed in fine print under the song just after the copyright information (if any). Writing `\setlicense{`⟨*license*⟩`}` anywhere between the `\beginsong` and `\endsong` lines is equivalent to using `li={`⟨*license*⟩`}` in the `\beginsong` line.

Since many songs in a book are often covered by the same license, it is usually convenient to create a macro to abbreviate the licensing information. For example, if your organization has a music license from Christian Copyright Licensing International with license number 1234567, you might define a macro like

```
\newcommand{\CCLI}{(CCLI \#1234567)}
```

Then you could write `li=\CCLI` in the `\beginsong` line of each song covered by CCLI.

$$\begin{array}{rcl}
\langle \textit{refs} \rangle & \longrightarrow & \langle \textit{nothing} \rangle \mid \langle \textit{ref} \rangle\,\texttt{;}\llcorner\!\lrcorner\langle \textit{ref} \rangle\,\texttt{;}\ldots\texttt{;}\llcorner\!\lrcorner\langle \textit{ref} \rangle \\
\langle \textit{ref} \rangle & \longrightarrow & \langle \textit{many-chptr-book} \rangle\llcorner\!\lrcorner\langle \textit{chapters} \rangle \mid \langle \textit{one-chptr-book} \rangle\llcorner\!\lrcorner\langle \textit{verses} \rangle \\
\langle \textit{many-chptr-book} \rangle & \longrightarrow & \texttt{Genesis} \mid \texttt{Exodus} \mid \texttt{Leviticus} \mid \texttt{Numbers} \mid \ldots \\
\langle \textit{one-chptr-book} \rangle & \longrightarrow & \texttt{Obadiah} \mid \texttt{Philemon} \mid \texttt{2 John} \mid \texttt{3 John} \mid \texttt{Jude} \\
\langle \textit{chapters} \rangle & \longrightarrow & \langle \textit{chref} \rangle\,\texttt{,}\llcorner\!\lrcorner\langle \textit{chref} \rangle\,\texttt{,}\ldots\texttt{,}\llcorner\!\lrcorner\langle \textit{chref} \rangle \\
\langle \textit{chref} \rangle & \longrightarrow & \langle \textit{chapter} \rangle \mid \langle \textit{chapter} \rangle\texttt{-}\langle \textit{chapter} \rangle \mid \langle \textit{chapter} \rangle\texttt{:}\langle \textit{verses} \rangle \mid \\
 & & \quad \langle \textit{chapter} \rangle\texttt{:}\langle \textit{verse} \rangle\texttt{-}\langle \textit{chapter} \rangle\texttt{:}\langle \textit{verse} \rangle \\
\langle \textit{verses} \rangle & \longrightarrow & \langle \textit{vref} \rangle\,\texttt{,}\langle \textit{vref} \rangle\,\texttt{,}\ldots\texttt{,}\langle \textit{vref} \rangle \\
\langle \textit{vref} \rangle & \longrightarrow & \langle \textit{verse} \rangle \mid \langle \textit{verse} \rangle\texttt{-}\langle \textit{verse} \rangle
\end{array}$$

Figure 2: Formal syntax rules for song scripture references

`sr=` **Scripture References.** The songs package has extensive support for scripture citations and indexes of scripture citations. To cite scripture references for the song, use the keyval `sr={`⟨*refs*⟩`}`, where ⟨*refs*⟩ is a list of scripture references. Index entries will be added to all scripture indexes associated with the current `songs` environment for each such reference. The `songidx` index generation program expects ⟨*refs*⟩ to be a list of references in which semicolons are used to separate references to different books, and commas are used to separate references to to different chapters and verses within the same book. For example, one valid scripture citation would be

```
sr={John 3:16,17, 4:1-5; Jude 3}
```

The full formal syntax of a valid ⟨*refs*⟩ argument is given in Figure 2. In those syntax rules, ⟨*chapter*⟩ and ⟨*verse*⟩ are arabic numbers denoting a valid chapter number for the given book, and a valid verse number for the given chapter, respectively. Note that when referencing a book that has only one chapter, one should list only its verses after the book name (rather than `1:`⟨*verses*⟩).

`\nextcol` **Column Breaks.** The `\nextcol` macro can be used within a `songs` environment to force a column break. It should only appear between songs or scripture quotations. If the set is being typeset in one column, `\nextcol` forces a page break instead of a column break. When a partial list of songs is being extracted with `\includeonlysongs`, all `\nextcol` macros will be ignored.

## 7.1 Verses and Choruses

`\beginverse` `\endverse` `\beginchorus` `\endchorus`

**Starting A Verse Or Chorus.** Between the `\beginsong` and `\endsong` lines of a song can appear any number of verses and choruses. A verse begins and ends with:

```
\beginverse
⋮
\endverse
```

and a chorus begins and ends with:

```
\beginchorus
⋮
\endchorus
```

Verses are numbered (assuming `\noversenumbers` has not been used to suppress verse numbering) whereas choruses have a vertical line placed to their left.

You can also begin a verse with `\beginverse*` instead of `\beginverse` to create an unnumbered verse. This is often used for things that aren't really verses but should be typeset like a verse (e.g. intros, endings, and the like). A verse that starts with `\beginverse*` should still end with `\endverse` (not `\endverse*`).

Within a verse or chorus you should enter one line of text for each line of lyrics. The environment of a verse or chorus behaves as though `\obeylines` is active, so a line break in the source document produces a line break in the resulting document. Lines that are too long to fit will be wrapped with a hanging indentation equal to `\parindent`.

`\repchoruses` **Repeating Choruses.** When making overhead slides, it is often convenient to repeat the song's chorus once on each page, so that the projector-operator need not flip back to the first slide each time the chorus is to be sung. You can say `\repchoruses` to automate this process. This will cause the first chorus in each song to be automatically repeated after the first verse on each subsequent page of the song (unless that verse is already immediately followed by a chorus). If the first chorus is part of a set of two or more consecutive choruses, then the whole set of choruses will be repeated. (A set of choruses is assumed to consist of things like pre-choruses that should always be repeated along with the chorus.) Choruses will not be automatically inserted immediately after unnumbered verses (i.e., verses that begin with `\beginverse*`). Unnumbered verses are assumed to be bridges or endings that aren't followed by a chorus.

`\norepchoruses` The above heuristics cover the common cases, but they obviously don't cover every case. Some songs have more complex forms that don't fit the typical verse, chorus, verse, chorus pattern. The `\repchoruses` feature will not always be able to automatically insert choruses properly in these unusual cases. The best alternative is usually a manual approach. Before a song with irregular form, say `\norepchoruses` to turn automatic chorus-repeating off. Then, at points within the song where you want a chorus to be repeated on the overhead slides, type a construction like,

```
\ifslides
\beginchorus
⋮
\endchorus
\fi
```

and copy and paste the desired chorus into the middle. This will insert a repeated chorus at that point when generating slides, but not when generating a lyric book or chord book. After the song is concluded, type

```
\ifslides\repchoruses\fi
```

to turn automatic chorus-repeating back on, if desired.

`\brk` **Line Breaking.** When lines of lyrics are too wide to fit in a single line, TeX will automatically choose a reasonable place to break the line, wrapping it onto the next physical line of the document. However, sometimes it is desirable to specifically choose where TeX will break a long line so as to make it easier to read and sing. By placing a `\brk` macro within a line of lyrics, you can determine where TeX will break and wrap that line if it is too wide to fit in a single line of the resulting song book document. For example,

```
\beginverse
This is a \brk short line.
But this is a particularly long line of lyrics \brk that will
need to be wrapped.
\endverse
```

would produce

```
This is a short line.
But this is a particularly long line of lyrics
    that will need to be wrapped.
```

**Column and Page Breaking.** The `\brk` macro can also be used on a line by itself within a verse or chorus to suggest a page or column breakpoint if the verse or chorus is too long to fit in a single column. By default, the songs package will avoid inserting column- or page-breaks into the middle of verses and will never insert one into the middle of a chorus that is typeset with a vertical bar. When such a break is unavoidable, the package code will try to break the verse or chorus at a line where `\brk` appears by itself. If there are no `\brk` lines in a long verse, it will be broken somewhere that a line does not wrap. (A wrapped line is never divided by a column break.) If there are no `\brk` lines in a long chorus, it will overflow the column, yielding an overfull vbox warning.

## 7.2 Chords

`\[` Between the `\beginverse` and `\endverse` lines, or between the `\beginchorus`
`#` and `\endchorus` lines, chords can be produced using the macro `\[`⟨*chordname*⟩`]`.
`&` Chords will only appear in chord books. The ⟨*chordname*⟩ can consist of arbitrary text. To produce sharp and flat symbols, use `#` and `&` respectively.

Any text that immediately follows the `\[]` macro with no intervening whitespace will assumed to be the word or syllable that is to be sung as the chord is struck, and will therefore be typeset directly under the chord. For example:

```
                                        E♭        Am
\[E&]peace and \[Am]joy       produces  peace and joy
```

If whitespace (a space or end-of-line) immediately follows, then the chord name will be typeset without any lyric text below it, denoting that the chord is to be struck between any surrounding words. For example:

```
                                        E♭        Am
\[E&]peace and \[Am] joy      produces  peace and    joy
```

If the lyric text that immediately follows the chord ends with another chord, and if the width of the chord name exceeds the width of the lyric text, then hyphenation is added automatically. For example:

```
                                        F#sus4  A
\[F#sus4]e\[A]ternal          produces  e   -   ternal
```

Sequences of chords that sit above a single word can be written back-to-back with no intervening space, or as a single chord:

`\[A]\[B]\[Em]joy` *produces*
A B Em
joy

`\[A B Em]joy` *produces*
A B Em
joy

The only difference between the two examples above is that the chords in the first example can later be replayed separately (see §7.3) whereas the chords in the second example can only be replayed as a group.

You can explicitly dictate how much of the text following a chord macro is to appear under the chord name by using braces. To exclude text that would normally be drawn under the chord, use a pair of braces that includes the chord macro. For example:

`{\[G A]e}ternal` *produces*
G A
e - ternal

(Without the braces, the syllables "ternal" would not be pushed out away from the chord.) This might be used to indicate that the chord transition occurs on the first syllable rather than as the second syllable is sung.

Contrastingly, braces that do not include the chord itself can be used to include text under a chord that would otherwise be excluded. For example:

`\[Gmaj7sus4]{th' eternal}` *produces*
Gmaj7sus4
th' eternal

Without the braces, the word "eternal" would be pushed out away from the chord so that the chord would appear only over the partial word "th'". But since in this case the words "the eternal" are supposed to be sung together as a single three-syllable word (as indicated by the apostrophe), it is proper for the chord to span both words together.

`\DeclareLyricChar` **Symbols Under Chords.** If you are typesetting songs in a language whose alphabet contains symbols that LaTeX treats as punctuation, you may find yourself typing a lot of braces to get those symbols to appear properly under chords. Fortunately, there is a short-cut that can make this a lot easier. The following command instructs the the songs package to treat a given macro or token as non-chord-ending, so that it will by default be included under chords just like an alphabetic character.

`\DeclareLyricChar{`⟨*token*⟩`}`

Here, ⟨*token*⟩ must be a single TeX macro control sequence, active character, letter (something TeX assigns catcode 11), or punctuation symbol (something TeX assigns catcode 12). For example, by default,

`\[Fmaj7]s\dag range` *produces*
Fmaj7
s - †range

because `\dag` is not recognized as an alphabetic symbol; but if you first type,

```
\DeclareLyricChar{\dag}
```

then instead you will get:

```
\[Fmaj7]s\dag range
```
*produces* Fmaj7 / s†range

`\DeclareNonLyric` Likewise, you can type

```
\DeclareNonLyric{⟨token⟩}
```

to reverse the above effect and force a token to be lyric-ending. Such tokens will be pushed out away from long chord names so that they never fall under a chord, and hyphenation will be added to the resulting gap.

`\DeclareNoHyphen` To declare a token to be lyric-ending but without the added hyphenation, use `\DeclareNoHyphen{⟨token⟩}` instead. Such tokens will be pushed out away from long chord names so that they never fall under the chord, and hyphenation will not be added to the resulting gap.

`\MultiwordChords` **Extending Chords Over Adjacent Words.** The `\MultiwordChords` macro forces multiple words to be squeezed under one chord by default. Normally a long chord that sits atop a short lyric pushes subsequent lyrics away to make room for the chord:

```
\[Gmaj7sus4]my life
```
*produces*
```
Gmaj7sus4
my            life
```

But if you first type `\MultiwordChords`, then instead you will get the more compact:

```
\[Gmaj7sus4]my life
```
*produces*
```
Gmaj7sus4
my life
```

Authors should exercise caution when using `\MultiwordChords` because including many words under a single chord can often produce output that is ambiguous or misleading to musicians. For example,

```
\[F G Am]me free
```
*produces*
```
F G Am
me free
```

This is probably not what the author intended. The three chords were all supposed to be played while singing the word "me", but the output makes it look like some chords fall on the following word "free". Liberal use of braces is therefore required to make `\MultiwordChords` produce good results, which is why it isn't the default behavior.

`\shrp` `\flt` **Accidentals Outside Chords.** Sharp and flat symbols can be produced with `#` and `&` when they appear explicitly in a chord name, but if you wish to produce those symbols in other parts of the document, you must use the `\shrp` and `\flt` macros. For example, to define a macro that produces a *C#* chord, use:

```
\newcommand{\Csharp}{C\shrp}
```

## 7.3 Replaying Chords

`^` Many songs consist of multiple verses that use the same chords. The songs package simplifies this common case by providing a means to replay the chord sequence seen in a previous verse without having to retype all the chords. To replay a chord from a previous verse, type a hat symbol (^) anywhere you would otherwise use a chord macro (`\[]`). For example,

```
\beginverse
\[G]This is the \[C]first \[G]verse.
\endverse
\beginverse
The ^second verse ^ has the same ^chords.
\endverse
```

would produce

```
G           C     G
This is the first verse.

    G            C             G
The second verse    has the same chords.
```

Normal chords can appear amidst replayed chords without disrupting the sequence of chords being replayed. Thus, a third verse could say,

```
\beginverse
The ^third verse ^has a \[Cm]new ^chord.
\endverse
```

to produce

```
    G           C     Cm   G
The third verse has a new  chord.
```

Replaying works particularly well in conjunction with automatic transposition. See §10 for an example.

`\memorize` By default, chords are replayed from the current song's first verse, but you can replay the chords of a different verse or chorus by saying `\memorize` at the beginning of any verse or chorus whose chords you want to later replay. This causes the chord sequence of the current verse or chorus to be memorized. Subsequent verses or choruses within the same song can use ^ to replay the new sequence.

**Selective Memorization.** It is also possible to inject unmemorized chords into a memorized verse so that they will not be replayed. To suppress memorization of a chord, begin the chord's name with a hat symbol. For example,

```
\beginverse\memorize
The \[G]third \[C]chord will \[^Cm]not be re\[G]played.
\endverse
\beginverse
When ^replaying, the ^unmemorized chord is ^skipped.
\endverse
```

would produce

```
    G     C          Cm        G
The third chord will not  be replayed.

     G              C                    G
When replaying, the unmemorized chord is skipped.
```

This is useful when the first verse of a song has something unique, like an intro that won't be repeated in subsequent verses, but has other chords that you wish to replay.

**Memorizing Multiple Chord Sequences.** By default, the songs package only memorizes one sequence of chords at a time; using `\memorize` to memorize a new sequence causes any previously memorized sequence to be forgotten. However, you can memorize and replay multiple independent sequences using the macros described in the following paragraphs.

`\newchords` Memorized or replayed chord sequences can be stored in specific chord-replay registers. To declare a new chord-replay register, type

```
\newchords{⟨regname⟩}
```

where ⟨*regname*⟩ is a unique alphabetic identifier.

Once you've declared a register, you can memorize into that register by providing the ⟨*regname*⟩ as an optional argument to `\memorize`:

```
\memorize[⟨regname⟩]
```

Memorizing into a non-empty register replaces the contents of that register with the new chord sequence.

`\replay` To begin replaying chords from a particular register, type

```
\replay[⟨regname⟩]
```

Chord sequences memorized into registers declared with `\newchords` are global, which means you can memorize a chord sequence from one song and replay it in subsequent songs. You can also use `\replay` multiple times in the same verse or chorus to replay a sequence more than once.

## 7.4 Echoes and Repeats

`\echo` **Echo Parts.** Some songs contain echo parts that should be typeset differently from normal lyrics. To typeset an echo part, use `\echo{`⟨*lyrics and chords*⟩`}`. Echo parts will be parenthesized and italicized. For example,

`Alle\[G]luia! \echo{Alle\[A]luia!}` *produces* Alle$\overset{G}{\text{l}}$uia! *(Alle$\overset{A}{\text{l}}$uia!)*

`\rep` **Repeated Lines.** In other cases you might want to indicate that a line should be sung multiple times by all singers. To do so, put `\rep{`⟨*n*⟩`}` at the end of the line, where ⟨*n*⟩ is the number of times the line is to be repeated. For example,

`Alleluia! \rep{4}` *produces* Alleluia! (×4)

`\lrep` `\rrep` To indicate exactly where repeated parts begin and end, use `\lrep` and `\rrep` to create begin- and end-repeat signs. For example,

`\lrep \[G]Alleluia!\rrep \rep{4}` *produces* ‖: $\overset{G}{\text{A}}$lleluia! :‖ (×4)

## 7.5 Measure Bars

`\measurebar` `|` Measure bars can be added to chord books in order to help musicians keep time when playing unfamiliar songs. To insert a measure bar, type either `\measurebar` or type the vertical pipe symbol ("`|`"). For example,

```
Alle|\[G]luia
```
*produces* Alle|luia (chord G)

In order for measure bars to be displayed, the `showmeasures` option must be enabled. Measure bars are only displayed by default in chord books.

`\meter` The first measure bar in a song will have meter numbers placed above it to indicate the time signature of the piece. By default, these numbers will be 4/4, denoting four quarter notes per measure. To change the default, type `\meter{`⟨*n*⟩`}{`⟨*d*⟩`}` somewhere after the `\beginsong` line of the song but before the first measure bar, to declare a time signature of ⟨*n*⟩ ⟨*d*⟩th notes per measure.

`\mbar` You can also change meters mid-song either by using `\meter` in the middle of the song or by typing `\mbar{`⟨*n*⟩`}{`⟨*d*⟩`}` to produce a measure bar with a time signature of ⟨*n*⟩/⟨*d*⟩. For example,

```
\meter{6}{8}
\beginverse
|Sing to the |heavens, ye \mbar{4}{4}saints of |old!
\endverse
```

would produce

$\frac{6}{8}$|Sing to the |heavens, ye $\frac{4}{4}$|saints of |old!

## 7.6 Textual Notes

`\textnote` `\musicnote` Aside from verses and choruses, songs can also contain textual notes that provide various helpful instructions to singers and musicians. To create a textual note that will be displayed in both lyric books and chord books, use:

`\textnote{`⟨*text*⟩`}`

To create a textual note that will be displayed only in chord books, use:

`\musicnote{`⟨*text*⟩`}`

Both of these will create a shaded box containing ⟨*text*⟩. For example,

```
\textnote{Sing as a two-part round.}
```

would produce

Sing as a two-part round.

Textual notes can be placed anywhere within a song, either within verses and choruses or between them.

`\capo` **Guitar Capos.** One special kind of textual note suggests to guitarists which fret they should put their capo on in order to put the song in a good key for singing. Macro `\capo{`⟨*n*⟩`}` should be used for this purpose. It normally has the same effect as `\musicnote{capo `⟨*n*⟩`}`; however, if the `transposecapos` option is active, then it will instead have the effect of `\transpose{`⟨*n*⟩`}`. See §10 for more information on automatic chord transposition.

## 7.7 Index Entries

Every song automatically gets entries in the current section's title index(es) for every title specified in the song's `\beginsong` line. However, you can also add extra index entries for a song to any index.

`index=` **Indexing Lyrics.** For example, title indexes often have entries for memorable lines of lyrics in a song in addition to the song's title. You can add an index entry for the current song to the section's title index(es) by adding `index={`⟨*lyrics*⟩`}` to the song's `\beginsong` line. For example,

```
\beginsong{Doxology}
          [index={Praise God from Whom all blessings flow}]
```

would cause the song to be indexed both as "*Doxology*" and as "Praise God from Whom all blessings flow" in the section's title index(es). You can use `index=` multiple times in a `\beginsong` line to produce multiple additional index entries. Index entries produced with `index={`⟨*lyrics*⟩`}` will be typeset in an upright font instead of in italics to distinguish them from song titles.

`ititle=` **Indexing Extra Song Titles.** To add a regular index entry typeset in italics to the title index(es), use:

`ititle={`⟨*title*⟩`}`

in the `\beginsong` line instead. Like `index=` keyvals, `ititle=` can be used multiple times to produce multiple additional index entries.

`\indexentry` `\indextitleentry` You can also create index entries by saying `\indexentry[`⟨*indexes*⟩`]{`⟨*lyrics*⟩`}` (which creates an entry like `index=`) or `\indextitleentry[`⟨*indexes*⟩`]{`⟨*title*⟩`}` (which creates an entry like `ititle=`). These two macros can be used anywhere between the song's `\beginsong` and `\endsong` lines, and can be used multiple times to produce multiple entries. Without the optional ⟨*indexes*⟩ argument, the new entry is added to all of the title indexes for the current `songs` environment. If specified, ⟨*indexes*⟩ is a comma-separated list of index identifiers.

## 7.8 Chords in Ligatures

This subsection covers an advanced topic and can probably be skipped by those creating song books for non-professional use.

The `\[` macro is the normal means by which chords should be inserted into a song; however, a special case occurs when a chord falls within a ligature. Ligatures are combinations of letters or symbols that TeX normally typesets as a single font character so as to produce cleaner-looking output. The only ligatures in English are: ﬀ, ﬁ, ﬂ, ﬃ, and ﬄ. Other languages have additional ligatures like æ and œ. Notice that in each of these cases, the letters are "squished" together to form a single composite symbol.

Normally, producing a ligature like "ﬃ" in TeX is easy: if you type "`difficult`" in your document, TeX will observe the letters `ffi` occurring in sequence, change them into a ligature, and produce "diﬃcult" in the resulting document. But when a chord falls within a ligature, that process breaks down. For example, if you type `\[Gsus4]dif\[G]ficult`, then TeX produces "difficult" instead of diﬃcult even in the unchorded lyric book. (The difference between the two is subtle, so you have to look closely to see it. Notice that there is a break between the f's in the first instance that isn't present in the second.)

`\ch` To place a chord within a ligature without breaking the ligature, use the `\ch` macro, which functions a lot like TeX's `\discretionary` macro does for hyphenation. The syntax is:

```
\ch{⟨chord⟩}{⟨pre⟩}{⟨post⟩}{⟨full⟩}
```

where ⟨*chord*⟩ is the chord text, ⟨*pre*⟩ is the text to appear before the hyphen if the ligature is broken by auto-hyphenation, ⟨*post*⟩ is the text to appear after the hyphen if the ligature is broken by auto-hyphenation, and ⟨*full*⟩ is the full ligature if it is not broken by hyphenation. If the ligature is broken by auto-hyphenation, the ⟨*pre*⟩ text falls before the chord and the ⟨*post*⟩ text falls under the chord. If the ligature is not broken by auto-hyphenation, the chord text appears over the middle of the ⟨*full*⟩ text.

So for example, to correctly typeset `\[Gsus4]dif\[G]ficult`, in which the *G* chord falls in the middle of the "ﬃ" ligature, one should use:

`di\ch{G}{f}{fi}{ffi}cult` *produces* di$\overset{G}{\text{ﬃ}}$cult

This causes the "ﬃ" ligature to appear intact yet still correctly places the *G* chord over the second f. To use the `\ch` macro with a replayed chord name (see §7.3), use `^` as the ⟨*chord*⟩.

`\mch` The `\mch` macro is exactly like the `\ch` macro except that it also places a measure bar into the ligature along with the chord. For example,

`di\mch{G}{f}{fi}{ffi}cult` *produces* di$\overset{|G}{\text{ﬃ}}$cult

places both a measure bar and a *G* chord after the first "f" in "difficult", yet correctly produces an unbroken "ﬃ" ligature in copies of the book in which measure bars are not displayed.

In the unusual case that a meter change is required within a ligature, this can be achieved with a construction like:

`\meter{6}{8}di\mch{G}{f}{fi}{ffi}cult` *produces* dif$\overset{\frac{6}{8}G}{|}$ficult

The `\meter` macro sets the new time signature, which appears above the next measure bar—in this case the measure bar produced by the `\mch` macro.

Chords and measure bars produced with `^` or `|` are safe to use in ligatures. So `dif|^ficult` requires no special treatment; it leaves the "ﬃ" ligature intact when measure bars are not being displayed.

## 8 Scripture Quotations

Aside from songs, `songs` environments (see §5) can also include scripture quotations.

`\beginscripture` `\endscripture` **Starting a Scripture Quotation.** A scripture quotation begins and ends with

```
\beginscripture{⟨ref⟩}
⋮
\endscripture
```

where ⟨*ref*⟩ is a scripture reference that will be typeset at the end of the quotation. The ⟨*ref*⟩ argument should conform to the same syntax rules as for the ⟨*ref*⟩ arguments passed to `\beginsong` macros (see §7).

Between the `\beginscripture` and `\endscripture` lines, the text of the scripture quote should follow, which will be parsed in normal paragraph mode. For example:

```
\beginscripture{James 5:13}
Is any one of you in trouble? He should pray. Is anyone happy?
Let him sing songs of praise.
\endscripture
```

would produce

*Is any one of you in trouble? He should pray. Is anyone happy? Let him sing songs of praise.* James 5:13

**Tuplets.** If you are typesetting biblical poetry instead of prose, some extra constructs are required to typeset the text the way it appears in most bibles. Biblical poetry consists of tuplets—usually couplets and occasionally a triplet. The first line of each tuplet, called the "A-colon", is typeset flush with the left margin, while each additional line of the tupet, called the "B-colon", "C-colon", etc., is indented from the left margin. Any lines too long to fit are wrapped with double-width hanging indentation.

`\Acolon` `\Bcolon` You can produce this style of output by beginning the first line of a tuplet with an `\Acolon` macro and each additional line with a `\Bcolon` macro. Each line of the tuplet will then appear on its own line in the resulting scripture quotation, with proper indentation and line wrapping. For example,

```
\beginscripture{Psalm 1:1}
\Acolon Blessed is the man
\Bcolon who does not walk in the counsel of the wicked
\Acolon or stand in the way of sinners
\Bcolon or sit in the seat of mockers.
\endscripture
```

would produce

*Blessed is the man*
    *who does not walk in the counsel*
        *of the wicked*
*or stand in the way of sinners*
    *or sit in the seat of mockers.*
Psalm 1:1

`\strophe` **Stanzas.** Biblical poetry is often grouped into stanzas or "strophes", each of which is separated from the next by a small vertical space. You can create that vertical space by typing `\strophe`. For example,

```
\beginscripture{Psalm 88:2-3}
\Acolon May my prayer come before you;
\Bcolon turn your ear to my cry.
\strophe
\Acolon For my soul is full of trouble
\Bcolon and my life draws near the grave.
\endscripture
```

would produce

*May my prayer come before you;*
*turn your ear to my cry.*

*For my soul is full of trouble*
*and my life draws near the grave.*
Psalm 88:2–3

`\scripindent` `\scripoutdent` **Indented Blocks.** Some bible passages, such as those that mix prose and poetry, contain indented blocks of text. You can increase the indentation level within a scripture quotation by using `\scripindent` and decrease it by using `\scripoutdent`. For example,

```
\beginscripture{Hebrews 10:17-18}
Then he adds:
\scripindent
\Acolon ``Their sins and lawless acts
\Bcolon I will remember no more.''
\scripoutdent
And where these have been forgiven, there is no longer any
sacrifice for sin.
\endscripture
```

would produce

*Then he adds:*
*“Their sins and lawless acts*
*I will remember no more.”*
*And where these have been forgiven,*
*there is no longer any sacrifice for sin.*
Hebrews 10:17–18

## 9 Tablature Diagrams

`\gtab` Guitar tablature diagrams can be created by using the construct

`\gtab{`⟨*chord*⟩`}{`⟨*fret*⟩`:`⟨*strings*⟩`:`⟨*fingering*⟩`}`

where the ⟨*fret*⟩ and ⟨*fingering*⟩ parts are both optional (and you can omit any colon that borders an omitted argument).

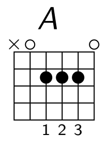
⟨*chord*⟩ is a chord name to be placed above the diagram.

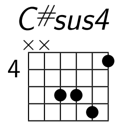
⟨*fret*⟩ is usually omitted, but if the top row of the diagram is intended to represent a fret other than the first one, then ⟨*fret*⟩ should be the number of the fret it represents (any number from 2 to 9).

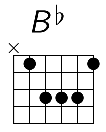
⟨*strings*⟩ should be a series of six symbols, one for each string of the guitar from lowest pitch to highest. Each symbol should be one of: `X` if that string is not to be played, `0` (zero or the letter O) if that string is to be played open, or one of `1` through `4` if that string is to be played on the given numbered fret. If `X` is used, that string will have an × placed above it in the tablature diagram. If `0` is used, that string will have an ∘ placed above it in the tablature diagram. If one of `1` through `4` is used, that string will have a • placed on it in the given numbered fret row of the diagram.

⟨*fingering*⟩ should either be empty if no fingering information is to be given, or it should likewise consist of a series of six symbols, one for each string of the guitar from lowest pitch to highest. Each symbol should be one of: `0` if no fingering information is to be displayed for that string (e.g., if the string is not being played or is being played open), or one of `1` through `4` to indicate that the given numbered finger is to be used to hold down that string. If ⟨*fingering*⟩ is provided, fingering numbers will be shown below each string of the resulting tablature diagram.

Here are some examples to illustrate:

`\gtab{A}{X02220:001230}` *produces* 

`\gtab{C#sus4}{4:XX3341}` *produces* 

`\gtab{B&}{X13331}` *produces* 

## 10 Automatic Transposition

`\transpose` You can automatically transpose some or all of the chords in a song up by ⟨*n*⟩ half-steps by adding the line

`\transpose{`⟨*n*⟩`}`

somewhere between the song's `\beginsong` line and the first chord to be transposed. For example, if a song's first chord is `\[D]`, and the line `\transpose{2}` appears before it, then the chord will appear as an *E* in the resulting document. Specifying a negative number for ⟨*n*⟩ will transpose subsequent chords down instead of up.

The `\transpose` macro will affect all chords appearing after it until the `\endsong` line. If two `\transpose` macros appear in the same song, their effects will be cumulative.

When the `transposecapos` option is active, the `\capo` macro acts like `\transpose`. See §7.6 for more information.

`\preferflats`
`\prefersharps`

**Enharmonics.** When using `\transpose` to automatically transpose the chords of a song, the songs package code will choose between enharmonically equivalent names for "black key" notes based on the first chord of the song. For example, if `\transpose{1}` is used, and if the first chord of the song is an *E*, then all *A* chords that appear in the song will be transcribed as $B^\flat$ chords rather than *A#* chords, since the key of *F*-major (*E* transposed up by one half-step) has a flatted key signature. Usually this guess will produce correct results, but if not, you can use either `\preferflats` or `\prefersharps` after the `\transpose` line to force all transcription to use flatted names or sharped names respectively, when resolving enharmonic equivalents.

**Modulated Verses.** Automatic transposition can be used in conjunction with chord-replaying (see §7.2) to produce modulated verses. For example,

```
\beginverse\memorize
\[F#]This is a \[B/F#]memorized \[F#]verse. \[E&7]
\endverse
\transpose{2}
\beginverse
^This verse is ^modulated up two ^half-steps.
\endverse
```

produces

```
F#        B/F#      F#   E♭7
This is a memorized verse.

A         D               A
This verse is modulated up two half-steps.
```

This works because memorization and replaying happen before transposition. That is, when memorizing and transposing chords at the same time, the chords are memorized as written, and then transposed chords are typeset. When replaying and transposing chords at the same time, transposition is applied to the untransposed chords that were memorized.

`\trchordformat`

**Multiple Keys.** By default, when chords are automatically transposed using `\transpose`, only the transposed chords are printed. However, in some cases you may wish to cause both the old chords and the transposed chords to be printed side-by-side so that musicians playing differently-tuned instruments can play from the same piece of music. This can be achieved by redefining the macro `\trchordformat{`⟨*old*⟩`}{`⟨*new*⟩`}`, where ⟨*old*⟩ is the old chord name and ⟨*new*⟩ is the transposed chord name. For example, to print the old chord above the new chord above each lyric, define

```
\renewcommand{\trchordformat}[2]{\vbox{\hbox{#1}\hbox{#2}}}
```

`\solfedge`
`\alphascale`

**Changing Note Names.** In many countries it is common to use the solfedge names for the notes of the scale (*LA, SI, DO, RE, MI, FA, SOL*) instead of the alphabetic names (*A, B, C, D, E, F, G*). By default, the transposition logic only understands alphabetic names, but you can tell it to look for solfedge names by typing `\solfedge`. To return to alphabetic names, type `\alphascale`.

`\notenames`

You can use other note names as well. To define your own note names, type

```
\notenames{⟨nameA⟩}{⟨nameB⟩}...{⟨nameG⟩}
```

where each of ⟨*nameA*⟩ through ⟨*nameG*⟩ must consist entirely of a sequence of one or more uppercase letters. For example, some solfedge musicians use *TI* instead of *SI* for the second note of the scale. To automatically transpose such music, one should type:

```
\notenames{LA}{TI}{DO}{RE}{MI}{FA}{SOL}
```

`\notenamesin` `\notenamesout` The songs package can also automatically convert one set of note names to another. For example, suppose you have a large song book in which chords have been typed using alphabetic note names, but you wish to produce a book that uses the equivalent solfedge names. You could achieve this by using the `\notenamesin` macro to tell the songs package which note names you typed in the input file, and then using `\notenamesout` to tell the songs package how you want it to typeset each note name in the output file. The final code would look like this:

```
\notenamesin{A}{B}{C}{D}{E}{F}{G}
\notenamesout{LA}{SI}{DO}{RE}{MI}{FA}{SOL}
```

The syntaxes of `\notenamesin` and `\notenamesout` are identical to that of `\notenames` (see above), except that the arguments of `\notenamesout` can consist of any LaTeX code that is legal in horizontal mode, not just capital letters.

To stop converting between note names, use `\alphascale`, `\solfedge`, or `\notenames` to reset all note names back to identical input and output scales.

`\transposehere` **Transposing Chords In Macros.** The automatic transposition logic won't find chord names that are hidden inside macro bodies. For example, if you abbreviate a chord by typing,

```
\newcommand{\mychord}{F\shrp sus4/C\shrp}
\transpose{4}
\[\mychord]
```

then the `\transpose` macro will fail to transpose it; the resulting chord will still be an *F#sus4/C#* chord. To fix the problem, you can use `\transposehere` in your macros to explicitly invoke the transposition logic on chord names embedded in macro bodies. The above example could be corrected by instead defining:

```
\newcommand{\mychord}{\transposehere{F\shrp sus4/C\shrp}}
```

## 11 Customizing the Book

The default appearance of a song book can be customized in a variety of ways, detailed below.

### 11.1 Song and Verse Numbering

Song numbering in each song section, and verse numbering in each song, are each controlled in similar ways:

songnum **Song Numbering.** The songnum counter defines the next song's number. It is set to 1 at the beginning of a songs environment and is increased by 1 after each \endsong. It can be redefined anywhere except within a song. For example,

```
\setcounter{songnum}{3}
```

would set the next song's number to be 3.

\songnumstyle You can change the song numbering style for a song section by redefining the \songnumstyle macro, which should accept a counter as its single argument. For example, to cause songs to be numbered in uppercase roman numerals, define

```
\renewcommand{\songnumstyle}[1]{\Roman{#1}}
```

The expansion of \songnumstyle must always produce plain text with no font formatting or unexpandable macro tokens. The text produced by \songnumstyle will be exported to auxiliary index generation files where it will be lexigraphically sorted and undergo other processing.

\printsongnum To change the formatting of song numbers as they appear at the beginning of each song, you should instead redefine the \printsongnum macro, which expects the text yielded by \songnumstyle as its only argument. For example, to typeset song numbers in italics, define

```
\renewcommand{\printsongnum}[1]{\it\LARGE#1}
```

Note that \printsongnum will *not* affect the typesetting style for song numbers displayed elsewhere, such as in indexes. It only affects how song numbers are rendered at the beginning of each song.

\songnumwidth The \songnumwidth length defines the width of the shaded boxes that contain song numbers at the beginning of each song. For example, to make each such box 2 centimeters wide, you could define

```
\setlength{\songnumwidth}{2cm}
```

versenum **Verse Numbering.** The versenum counter defines the next verse's number. It is set to 1 after each \beginsong line and is increased by 1 after each \endverse (except if the verse begins with \beginverse*). The versenum counter can be redefined anywhere within a song. For example,

```
\setcounter{versenum}{3}
```

would set the next verse's number to be 3.

\versenumstyle You can change the song numbering style by redefining the \versenumstyle macro, which should accept a counter as its single argument. For example, to cause verses to be numbered in uppercase roman numerals, define

```
\renewcommand{\versenumstyle}[1]{\Roman{#1}}
```

The expansion of \versenumstyle should always produce plain text with no font formatting or unexpandable macro tokens.

\printversenum To change the formatting of verse numbers as they appear at the beginning of each verse, you should instead redefine the \printversenum macro, which expects the text yielded by \versenumstyle as its only argument. For example, to typeset verse numbers in italics, define

```
\renewcommand{\printversenum}[1]{\it\LARGE#1.\ }
```

`\versenumwidth` The `\versenumwidth` length defines the horizontal space reserved for verse numbers to the left of each verse text. Verse text will be shifted right by this amount. For example, to reserve half a centimeter of space for verse numbers, define

```
\setlength{\versenumwidth}{0.5cm}
```

You can set `\versenumwidth` to a size less than the space taken up by some or all of the verse numbers. Doing so will cause the first line of the verse to be sufficiently indented to make room for the verse number, but the rest of the lines of the verse will only be indented by `\versenumwidth`.

`\noversenumbers` To turn off verse numbering entirely, use `\noversenumbers`. This is equivalent to saying

```
\renewcommand{\printversenum}[1]{}
\setlength{\versenumwidth}{0pt}
```

`\placeversenum` The horizontal placement of verse numbers within the first line of each verse is controlled by the `\placeversenum` macro. By default, each verse number is placed flush-left. For more information on this macro, recompile this documentation with the implementation section included.

## 11.2 Song Appearance

`\lyricfont` **Font Selection.** By default, lyrics will be typeset using the document-default font (`\normalfont`) and with the document-default point size (`\normalsize`). You can change these defaults, however, by redefining `\lyricfont`. For example, to cause lyrics to be typeset in small sans serif font, you could define

```
\renewcommand{\lyricfont}{\sffamily\small}
```

`\stitlefont` Song titles are typeset in a sans-serif, slanted font by default (sans-serif, upright if producing slides). You can change this default by redefining `\stitlefont`. For example, to cause lyrics to be typeset in a roman font, you could define

```
\renewcommand{\stitlefont}{\rmfont\Large}
```

`\versefont` `\chorusfont` You can apply additional font changes to verses and choruses by redefining `\versefont` and `\chorusfont`. For example, to typeset choruses in italics, you could define

```
\renewcommand{\chorusfont}{\it}
```

`\printchord` By default, chords will be typeset in sans serif oblique (slanted) font. You can customize chord appearance by redefining `\printchord`, which accepts the chord text as its sole argument. For example, to cause chords to be printed in roman boldface font, you could define

```
\renewcommand{\printchord}[1]{\rmfamily\bf#1}
```

`\everyverse` `\everychorus` **Verse and Chorus Titles.** The `\everyverse` macro is executed at the beginning of each verse, and `\everychorus` is executed at the beginning of each chorus. Thus, to begin each chorus with the word "Chorus:" one could type,

```
\renewcommand{\everychorus}{\textnote{Chorus:}}
```

`\versesep` **Spacing Options.** The vertical distance between song verses and song choruses is defined by the skip register `\versesep`. For example, to put 12 points of space between each pair of verses and choruses, with a flexibility of plus or minus 2 points, you could define

```
\versesep=12pt plus 2pt minus 2pt
```

`\baselineadj` The vertical distance between the baselines of consecutive lines of lyrics is computed by the `songs` package based on several factors including the lyric font size, the chord font size (if in `chorded` mode), and whether `slides` mode is currently active. You can adjust the results of this computation by redefining skip register `\baselineadj`. For example, to reduce the natural distance between baselines by 1 point but allow an additional 1 point of stretching when attempting to balance columns, you could define

```
\baselineadj=-1pt plus 1pt minus 0pt
```

`\cbarwidth` The width of the vertical line that appears to the left of choruses is controlled by the `\cbarwidth` length. To eliminate the line entirely (and the spacing around it), you can set `\cbarwidth` to `0pt`:

```
\setlength{\cbarwidth}{0pt}
```

`\sbarheight` The height of the horizontal line that appears between each pair of songs is controlled by the `\sbarheight` length. To eliminate the line entirely (and the spacing around it), you can set `\sbarheight` to `0pt`:

```
\setlength{\sbarheight}{0pt}
```

`\makeprelude` `\makepostlude` **Song Top and Bottom Material.** For complete control over the appearance of the header and footer material that precedes and concludes each song, you can redefine the macros `\makeprelude` and `\makepostlude`. When typesetting a song, the `songs` package code invokes both of these macros once (after processing all the material between the `\beginsong` and `\endsong` lines), placing the results within vboxes. The resulting vboxes are placed atop and below the song content. By default, `\makeprelude` displays the song's titles, authors, and scripture references to the right of a shaded box containing the song's number; and `\makepostlude` displays the song's copyright and licensing information in fine print.

To customize the default behavior, you can override these two macros with new definitions. Within the new definitions, use `\songtitle` to get the song's primary title and use `\nexttitle` or `\foreachtitle` to access any alternate titles. The song's authors, scripture references, and copyright information (if any) can be retrieved with `\songauthors`, `\songrefs`, and `\songcopyright`. The song's licensing information (if any) can be retrieved with `\songlicense`. More information about these macros can be found in §12.1. To get the song's number, use `\songnumstyle{songnum}` (see §11.1).

`\extendprelude` `\extendpostlude` Sometimes a less drastic addition to the song header or footer is desired. To use the existing song header and footer structures but change some material under the title in the header, or to change the material in the footer, redefine `\extendprelude` or `\extendpostlude`. For example, to print the words "Used with permission" at the end of every song's footer information, one could define

```
\renewcommand{\extendpostlude}{
  \songcopyright\ \songlicense\unskip
  \ Used with permission.
}
```

For an example of how to redefine `\extendprelude` see §12.2.

`\showauthors` `\showrefs` Within `\extendprelude` or `\extendpostlude`, one can use `\showauthors` and `\showrefs` to display the song author information and scripture reference citations (if any). See the `\newsongkey` macro in §12.2 for an example.

`\vvpenalty` `\ccpenalty` `\vcpenalty` `\cvpenalty` **Page- and Column-breaking.** Page-breaking and column-breaking within songs that are too large to fit in a single column/page is influenced by the values of several penalties. Penalties of value `\vvpenalty`, `\ccpenalty`, `\vcpenalty`, and `\cvpenalty` are inserted into each song between consecutive verses, between consecutive choruses, after a verse followed by a chorus, and after a chorus followed by a verse, respectively. The more negative the penalty, the more likely TeX is to place a page- or column-break at that penalty. If any are set to -10000 or lower, a break will be forced there. By default, all are set to -100 so that breaks between verses and choruses are preferred over breaks within choruses and verses, but are not forced.

`\sepverses` Saying `\sepverses` sets all of the above penalties to -10000 except for `\ccpenalty` which remains set to -100. This is useful in `slides` mode because it forces each verse and chorus to be typeset on a separate slide, except for consecutive choruses, which remain together when possible. (This default reflects an expectation that consecutive choruses typically consist of a pre-chorus and chorus that are always sung together.)

These defaults can be changed by changing the relevant penalty register directly. For example, to force a page- or column-break between consecutive choruses, type

```
\ccpenalty=-10000
```

`\spenalty` The value of `\spenalty` controls whether multiple songs are allowed to appear in a single column/page. Values higher than -10000 allow multiple songs; other values cause each song to be started on a fresh column/page. The default is 0, except when producing slides when the default is -10000.

`\versejustify` `\chorusjustify` **Text Justification.** Various parameters affecting the justification of verses and choruses are controlled by macros `\versejustify` and `\chorusjustify`, respectively. By default, both typeset paragraphs ragged-right with hanging indentation, and they introduce space at the left margin for verse numbers and the vertical bar shown to the left of choruses. In slides mode, they are redefined to center all lyrics.

`\justifyleft` `\justifycenter` To force verses or choruses to be left-justified or centered, set `\versejustify` or `\chorusjustify` equal to `\justifyleft` or `\justifycenter`, respectively. For example, to cause choruses to be centered, one could type:

```
\renewcommand{\chorusjustify}{\justifycenter}
```

`\notejustify` Justification of textual notes too long to fit on a single line is controlled by the `\notejustify` macro. By default, it sets up an environment that fully justifies the note (i.e., all but the last line of each paragraph extends all the way from the left to the right margin). For more information, recompile this documentation with the implementation section included.

`\placenote` A textual note that is shorter than a single line is placed flush-left by default, or is centered when in slides mode. This placement of textual notes is controlled by `\placenote`. For more information, recompile this documentation with the implementation section included.

### 11.3 Scripture Appearance

`\scripturefont` By default, scripture quotations will be typeset in Zaph Chancery font with the document-default point size (`\normalsize`). You can change these defaults by redefining `\scripturefont`. For example, to cause scripture quotations to be typeset in sans serif italics, you could define:

```
\renewcommand{\scripturefont}{\sffamily\it}
```

`\printscrcite` By default, the citation at the end of a scripture quotation will be typeset in sans serif font at the document-default point size (`\normalsize`). You can customize the appearance of the citation by redefining `\printscrcite`, which accepts the citation text as its sole argument. For example, to cause citations to be printed in roman italics font, you could define

```
\renewcommand{\printscrcite}[1]{\rmfamily\it#1}
```

### 11.4 Indexes

`\idxheadwidth` **Index Appearance.** The `\idxheadwidth` length defines the width of the shaded boxes that begin each alphabetic block of a large index. For example, to set the width of those boxes to 1 centimeter, you could define

```
\setlength{\idxheadwidth}{1cm}
```

`\idxcont` In a scripture index, when a column break separates a block of entries devoted to a book of the Bible, the new column is titled "⟨*bookname*⟩ (continued)" by default. You can change this default by redefining the `\idxcont` macro, which will receive the ⟨*bookname*⟩ as its single argument. For example, to typeset an index in German, one might define

```
\renewcommand{\idxcont}[1]{#1 (fortgefahren)}
```

`\titleprefixword` **Alphabetization Options.** In English, when a title begins with "The" or "A", it is traditional to move these words to the end of the title and sort the entry by the following word. So for example, "The Song Title" would be indexed as "Song Title, The". To change this default behavior, you can use `\titleprefixword` in the document preamble to define each word that will be moved to the end whenever it appears as the first word of a title index entry. For example, to cause the word "I" to be moved to the end of title index entries, one could say,

```
\titleprefixword{I}
```

The first use of `\titleprefixword` overrides the defaults, so if you also want to continue to move "The" and "A" to the end of entries, you must also say `\titleprefixword{The}` and `\titleprefixword{A}` explicitly. This macro may only be used in the document preamble.

`\authsepword` **Special Words In Song Info.** When parsing author index entries, the word "and" is recognized by the `songidx` program as a conjunctive that separates author names. To override this default and specify a different conjunctive, use the `\authsepword` macro one or more times in the document preamble. For example, to instead treat "und" as a conjunctive, you could say,

```
\authsepword{und}
```

The first use of `\authsepword` overrides the default, so if you also want to continue to treat "and" as a conjunctive, you must also say `\authsepword{and}` explicitly. The `\authsepword` macro may only be used in the document preamble.

`\authbyword` When parsing author index entries, the word "by" is recognized as a keyword signaling that the index entry should only include material in the current list item that follows the word "by". So for example, "Music by J.S. Bach" would be indexed as "Bach, J.S." rather than "Bach, Music by J.S." To recognize a different word instead of "by", you can use `\authbyword` in the document preamble. For example, to recognize "durch" instead, you could say

```
\authbyword{durch}
```

The first use of `\authbyword` overrides the default, so if you also want to continue to treat "by" as a keyword, you must also say `\authbyword{by}` explicitly. The `\authbyword` macro may only be used in the document preamble.

`\authignoreword` When parsing author index entries, if a list item contains the word "unknown", that item is ignored and is not indexed. This prevents items like "Composer unknown" from being indexed as names. To cause the indexer to recognize and ignore a different word, you can use the `\authignoreword` macro in the document preamble. For example, to ignore author index entries containing the word "unbekannt", you could say,

```
\authignoreword{unbekannt}
```

The first use of `\authignoreword` overrides the default, so if you also want to continue to ignore list items containing the word "unknown", you must also say `\authignoreword{unknown}` explicitly. The `\authignoreword` macro may only be used in the document preamble.

## 11.5 Other Customizations

`\colminheight` **Column Balancing.** Each column in a two-column song book is vertically stretched to be at least as high as the value of length `\colminheight`. In `lyric` books, `\colminheight` is set by default to be the value of `\textheight`. This has the effect of making every column exactly `\textheight` high. In `chorded` books, `\colminheight` defaults to `0pt`, causing each column to be typeset at its natural height without any stretching. You can change the value of `\colminheight` to force a different amount of column stretching. For example, if you set

```
\setlength{\colminheight}{9in}
\setlength{\textheight}{9.5in}
```

then each column will be at least 9 inches high with a possible extra 0.5 inches of space to accommodate columns that are slightly larger.

\songmark
\versemark
\chorusmark

**Page Headers and Footers.** If you want to add information associated with songs to page headings and footers, you can redefine `\songmark`, `\versemark`, or `\chorusmark` to add the necessary TeX marks to the current page whenever a new song, verse, or chorus begins. These macros expect no arguments; to access the current song's information including titles, use the macros documented in §12.1. To access the current song's number or the current verse's number, use `\songnumstyle{songnum}` or `\songnumstyle{versenum}` (see §11.1). For example, to include the song number in the page headings produced by LaTeX's `\pagestyle{myheadings}` feature, you could redefine `\songmark` as follows:

```
\renewcommand{\songmark}{
  \markboth{\songnumstyle{songnum}}{\songnumstyle{songnum}}
}
```

## 12 Programming

The `songs` package provides numerous macros that can be helpful when writing LaTeX code for a song book document. These are described below.

### 12.1 Accessing Song Info

The macros described in this section are typically used within `\makeprelude` or `\makepostlude` to typeset the various information provided in the `\beginsong` line or elsewhere within the song. However, they can also be used elsewhere within a song if desired.

\songauthors To access the current song's list of authors (if any) use `\songauthors`. This yields the value of the `by=` key used in the `\beginsong` line.

\songrefs To access the current song's list of scripture references (if any) use `\songrefs`. The result of `\songrefs` will not yield exactly what was used in the `sr=` keyval of the `\beginsong` line; some preprocessing is done first. In particular, hyphens have been changed to en-dashes and spaces that fall within a list of verse numbers have been changed to thin spaces. In addition, various penalties have been added to inhibit line breaks in strange places and encourage line breaks in others.

\songcopyright To access the current song's copyright info (if any), use `\songcopyright`. This yields the value of the `cr=` key used in the `\beginsong` line.

\songlicense To access the current song's licensing information (if any), use `\songlicense`. This yields the value of the `li=` key used in the `\beginsong` line, or whatever text was last declared with `\setlicense`.

Since songs can have any number of titles, accessing the current song's title requires something more sophisticated than a single macro. The following macros describe how to access each of a song's titles in turn.

\songtitle The `\songtitle` macro yields the current song's title. By default this is the first title provided in the `\beginsong` line. The `\nexttitle` and `\foreachtitle` macros (see below) cause it to be set to the current song's other titles, if any.

\resettitles To access the current song's primary title (i.e. the first title specified in the song's `\beginsong` line), execute `\resettitles`. This sets the `\songtitle` macro to be the song's primary title.

\nexttitle To access the song's next title, you can execute `\nexttitle`, which sets

`\songtitle` to be the next title in the song's list of titles (or sets `\songtitle` to `\relax` if there are no more titles).

`\foreachtitle` Using `\nexttitle` in a `\loop` construction suffices to access all of a song's titles, but in the common case that you just want to access all of of the titles in sequence, there is an easier way. The `\foreachtitle` macro accepts a chunk of LaTeX code as its single argument and executes it once for each (remaining) song title. Within the code chunk, use `\songtitle` to access the current title.

For example, the following code would generate a comma-separated list of all of the current song's titles:

```
\resettitles
\songtitle
\nexttitle
\foreachtitle{, \songtitle}
```

## 12.2 Defining New Beginsong Keyvals

`\newsongkey` The `\beginsong` macro supports several optional keyval parameters, including `by=`, `sr=`, and `cr=`, for declaring song information; but users can define their own additional keyvals as well. To do so, use the `\newsongkey` macro, which has the syntax

`\newsongkey{`⟨*keyname*⟩`}{`⟨*initcode*⟩`}[`⟨*default*⟩`]{`⟨*setcode*⟩`}`

Here, ⟨*keyname*⟩ is the name of the new key for the keyval, ⟨*initcode*⟩ is LaTeX code that will be executed at the start of each `\beginsong` line before the `\beginsong` arguments are processed, ⟨*default*⟩ (if specified) is the default value used for the keyval when ⟨*keyname*⟩ appears in `\beginsong` without a value, and ⟨*setcode*⟩ is macro code that will be executed whenever ⟨*key*⟩ is parsed as part of the `\beginsong` keyval arguments. In ⟨*setcode*⟩, `#1` expands to the value assigned by the user to the keyval (or to ⟨*default*⟩ if no value was given).

For example, to define a new song key called `arr` which stores its value in a macro called `\arranger`, one could write:

```
\newcommand{\noarranger}{}
\newcommand{\arranger}{}
\newsongkey{arr}{\let\arranger=\noarranger}{\def\arranger{#1}}
```

Then one could redefine `\extendprelude` to print the arranger below the other song header information:

```
\renewcommand{\extendprelude}{
  \showrefs\showauthors
  \ifx\arranger\noarranger\else
    {\bfseries Arranged by \arranger\par}
  \fi
}
```

A `\beginsong` line could then specify the song's arranger as follows:

```
\beginsong{The Title}[arr={S. Omebody}]
⋮
\endsong
```

| Type | Processed only if... |
|---|---|
| `chorded` | the `chorded` option is active |
| `lyric` | the `chorded` option is not active |
| `slides` | the `slides` option is active |
| `partiallist` | the `\includeonlysongs` macro is being used to extract a partial list of songs |
| `songindexes` | the `noindexes` option is not active |
| `measures` | the `nomeasures` option is not active |
| `pdfindex` | the `nopdfindex` option is not active |
| `rawtext` | the `rawtext` option is active |
| `transcapos` | the `transposecapos` option is active |
| `vnumbered` | the current verse is numbered (i.e., it was started with `\beginverse` instead of `\beginverse*`) |

Table 1: Conditional macros

This would produce

**1** *The Title*
**Arranged by S. Omebody**

For more detailed information about keyvals and how they work, the reader is advised to consult the documentation for David Carlisle's `keyval` package, which should come standard with most LaTeX $2_\varepsilon$ installations.

## 12.3 Conditionals

The songs package provides a variety of macros for creating conditional blocks of code. These are useful for including certain verses or textual notes only in certain kinds of books. For example, a musician's chord book might include extra verses with alternate chordings; these extra verses wouldn't be shown in the non-chorded version of the book. Conditional blocks can also be used in LaTeX code to customize parameters or redefine macros whose definitions should depend on the kind of book being generated.

`\if...` A conditional block begins with a macro named `\if`⟨*type*⟩, where ⟨*type*⟩ is one of the types listed in the first column of Table 1. The conditional block concludes with the macro `\fi`. Any material between the `\if`⟨*type*⟩ and the `\fi` will be processed only if the condition in the second column of the table is true. Between the `\if`⟨*type*⟩ and the `\fi` may also appear an `\else`. When `\else` is used, then the material between the `\else` and the `\fi` will only be processed if the condition in the second column is *false*.

For example, in the construction

```
\ifchorded
  ⟨A⟩
\else
  ⟨B⟩
\fi
```

material ⟨*A*⟩ is only included if the `chorded` option is active, and material ⟨*B*⟩ is only included if the `chorded` option is not active.

The conditional blocks described above will work fine most places, but if they begin within a verse or chorus, then problems can result. In that case TEX may complain of a "runaway argument" because end-of-line has a special meaning within verses and choruses (it is `\outer`) that TEX disallows within conditional blocks. (For a more thorough explanation, recompile this document with the implementation included.)

`\begin...only` `\end...only` To avoid this problem, start the conditional block with `\begin`⟨*type*⟩`only` instead of `\if`⟨*type*⟩. The conditional block must end with an `\end`⟨*type*⟩`only` (with the same ⟨*type*⟩) instead of a `\fi`. For example:

```
\beginchordedonly
⋮
\endchordedonly
```

produces a conditional block that will only be processed if the `chorded` option is in effect. These conditionals can be used anywhere, not just within verses or choruses, but they have the limitation that they cannot contain an `\else`. (Keep reading for a way to work around this limitation, though.)

`\begin...never` `\end...never` You can create inverses of all the conditionals listed in Table 1 by substituting `never` for `only` in the macro names described in the previous paragraph. For example,

```
\beginslidesnever
⋮
\endslidesnever
```

produces a conditional block that will only be processed if the `slides` option is *not* in effect. This means that you can simulate an `\if`...`\else`...`\fi` construction by using two consecutive blocks with opposite conditions. For example,

```
\beginslidesonly
⋮
\endslidesonly
\beginslidesnever
⋮
\endslidesnever
```

is the same as `\ifslides`...`\else`...`\fi`, but it is safe to use within a verse or chorus.

### 12.4 Partial Song Lists

`\songlist` When `\includeonlysongs` is used to extract a partial list of songs, the `\songlist` macro expands to the comma-separated list of songs that is being extracted. Redefining `\songlist` within the document preamble will alter the list of songs to be extracted. Redefining it after the preamble may have unpredictable results.

### 12.5 Font Kerning Corrections

`\shiftdblquotes` **Scripture Font Quotation Marks.** The Zaph Chancery font, used by default to typeset scripture quotations, seems to have some kerning problems with its double-quote ligatures. In particular, every left double-quote and right-double quote seems to have about 1.1pt and 2pt of extra space, respectively, to its left (to my eye). This causes left double-quotes appearing at the left margin to appear indented, and it causes right double-quotes to appear spaced out to the right of the quoted text they finish. Rather than redefine the font metrics (which would complicate the installation of this software), the `\shiftdblquotes` macro can be used to adjust the spacing around all double-quote ligatures until the current scoping group ends. The syntax is:

`\shiftdblquotes{`⟨*ldqleft*⟩`}{`⟨*ldqright*⟩`}{`⟨*rdqleft*⟩`}{`⟨*rdqright*⟩`}`

where all four parameters are lengths. The effect is to insert ⟨*ldqleft*⟩ and ⟨*ldqright*⟩ extra space to the left and right of all left double-quote ligatures, and insert ⟨*rdqleft*⟩ and ⟨*rdqright*⟩ extra space to the left and right of all right double-quote ligatures.

To correct the kerning of the double-quote ligatures in the Zaph Chancery font, the `\scripturefont` macro invokes

```
\shiftdblquotes{-1.1pt}{0pt}{-2pt}{0pt}
```

Unless you use other fonts that also have this problem, or you use an unusual point size for Zaph Chancery that necessitates a different kerning correction, the user probably shouldn't need to use this macro explicitly.

**Chord Overstriking.** In order to conserve space and keep songs readable, the songs package pushes chords down very close to the lyrics with which they are paired. Unfortunately, this can sometimes cause low-hanging characters in chord names to overstrike the lyrics they sit above. For example,

`\[(Gsus4/D)]Overstrike` *produces* (Gsus4/D) Overstrike

Note that the parentheses and slash symbols in the chord name have invaded the lyric that sits beneath them.

`\chordlocals` A little bit of overstriking is definitely preferable to raising chord names higher (which would make songs more difficult for musicians to read and play), but book-makers with a penchant for high-quality typesetting might desire a better solution. The best solution is to use a font for chord names that minimizes low-hanging symbols; but if you lack such a font, then the following trick works pretty well. Somewhere in the preamble of your document, you can write the following LaTeX code:

```
\renewcommand{\chordlocals}{\catcode`(\active
                            \catcode`)\active
                            \catcode`/\active}
\newcommand{\smraise}[1]{\raise2pt\hbox{\small#1}}
\newcommand{\myslash}{\smraise/}
\newcommand{\myopenparen}{\smraise(}
\newcommand{\mycloseparen}{\smraise)}
{\chordlocals
 \global\let(\myopenparen
 \global\let)\mycloseparen
 \global\let/\myslash}
```

This sets the `/`, `(`, and `)` symbols as active characters whenever they appear within chord names. (Recompile this documentation to include the implementation section for more information about the `\chordlocals` macro.) Each active character is defined so that it produces a smaller, raised version of the original symbol. The result is as follows:

```
\[(Gsus4/D)]Overstrike (fixed)
```

*produces* $\overset{\textit{(Gsus4/D)}}{\text{Overstrike}}$ (fixed)

As you can see, the low-hanging symbols have been elevated so that they sit above the baseline, correcting the overstrike problem.

## 13 Index Generation

The material in this section describes macros provided by the songs package that are used during the automatic generation of the song book indexes. Since index generation is automatic, document authors should not normally need to use any of these macros directly. The documentation in this section is therefore provided purely for completeness and for informational purposes. For instructions on how to automatically generate indexes when compiling a song book, see §6. For info on how to customize the appearance of indexes, see §11.4.

Automatic generation of song book indexes is a three stage process:

1. Each time a song book LaTeX file is compiled, an auxiliary file named ⟨*filename*⟩`.sxd` will be written out for each ⟨*filename*⟩ defined using `\newindex`, `\newauthorindex`, or `\newscripindex`. These `.sxd` files are simple ascii files that can be viewed using any standard text editor. They begin with a line identifying the type of index (title, author, or scripture) and then contain triples of lines, one triple for each song to appear in the index. The first line of a triple has the information by which the song is being indexed (a title, author, or scripture reference). The second line has the song's number in the book (yielded by `\songnumstyle`). The third line is an identifying label for the song used in hyperlinking.

2. Once the `.sxd` files have been generated, an external program is used to transform each `.sxd` file into a `.sbx` file. Most LaTeX documents use the `makeindex` program provided with LaTeX to produce indexes from data files, but `makeindex` is not powerful enough to deal with scripture references. Thus, distributions of songs package software should come with a specialized index generation program to do this.

3. The `.sbx` files produced by the index generator program are then read in by the `\showindex` macro next time the source document is compiled using LaTeX. These `.sbx` files consist of the macros and environments described below.

`idxblock` In indexes that are blocked off into sections, one for each letter of the alphabet, the ⟨*filename*⟩`.sbx` files generated for that index will consist of a series of `idxblock` environments, one for each such section. An `idxblock` environment begins with

`\begin{idxblock}{`⟨*letter*⟩`}`

where ⟨*letter*⟩ is the letter of the alphabet for that block. An `idxblock` environment ends with `\end{idxblock}`.

`\idxentry` `\idxaltentry` The ⟨*filename*⟩`.sbx` files generated for each index will contain a series of lines of the form

```
\idxentry{⟨leftside⟩}{⟨rightside⟩}
\indexaltentry{⟨leftside⟩}{⟨rightside⟩}
```

each of which creates an index entry with ⟨*leftside*⟩ on the left, followed by a series of dots, followed by ⟨*rightside*⟩ on the right. The `\indexentry` is used for "normal" entries (e.g., titles in a title index), and `\indexaltentry` is used for "alternate" entries (e.g., lyric lines in a title index).

Within ⟨*rightside*⟩, multiple items are separated with `\\` macros instead of commas. When used in an index `.sbx` file, the `\\` macro will produce a comma followed by some complex spacing that allows index lines to be broken suitably if they are too long to fit in one physical line.

## 14 Other Similar Packages

There are a number of other LaTeX packages available for typesetting songs, tablature diagrams, or song books. Probably the best of these is the Song♭ook package by Christopher Rath (`http://rath.ca/Misc/Songbook`). Most of the differences between that package and this one are intentional; the following is a summary of where I've adopted various differing design decisions and why.

**Ease of Song Entry.** In designing the songs package, I invested a lot of effort in making it easy to type chords. With most LaTeX song book packages, including the Song♭ook package, the user types chords using a standard LaTeX macro syntax like `\Ch{`⟨*chord*⟩`}{`⟨*lyric*⟩`}`. Although I originally wrote the songs package to use a similar syntax for chords, I switched to the less conventional `\[`⟨*chord*⟩`]`⟨*lyric*⟩ syntax for several significant reasons detailed below.

First, macros in the standard LaTeX package syntax take longer to type than macros in the songs package's syntax (eight extra keypresses including uses of the shift key). This can become become really taxing when typing up a large book. Chords often appear as frequently as one per syllable, especially in hymns, so keeping the syntax as brief as possible is desirable.

Second, the standard LaTeX macro syntax isn't really suited to typesetting chords because it is extremely difficult for the user to know what exactly should go in the ⟨*lyric*⟩ part of the macro. Chords don't always lie above entire words; they often lie above only a syllable of a word, or they might lie above both a word and the punctuation that follows it. This means that in order to type chords correctly in a standard LaTeX macro format, a user must conform to some very unintuitive and complex style rules. For example, in the Song♭ook package, typing `\Ch{C}{difficult!}` and `\Ch{C}{diffi}cult!` and `\Ch{C}{difficult}!` all produce different results, each of which are undesirable for different reasons in different situations. The user must learn when to use which, and often must experiment to discover which works and which doesn't in any given situation. In contrast, in the songs package the ⟨*lyric*⟩ argument is implicit (it isn't surrounded by opening and closing braces), which allows the package programming to automatically determine which part of the following lyrics should lie under the chord and which should not. This eliminates the need for the user to learn any complex style rules unless something unusual is desired.

Third and finally, proper hyphenation is a significant challenge when typesetting song books. Extra hyphenation must usually appear in chord books wherever a chord is wider than the syllable it sits above. Such hyphenation should be omitted in lyric-only versions of the book since those versions lack chords, so the hyphenation would be superfluous and odd-looking. Packages that use a standard LaTeX macro syntax for chords require the user to manually identify places where extra hyphenation will be necessary and type something special in those places to make chord books look right. This can be very taxing and difficult for the user because it isn't usually possible to predict the need for hyphenation in advance. The user must therefore proof the chord book very carefully to identify and correct any hyphenation errors. In contrast, the macro syntax used by the songs package allows its programming to detect and correct all the usual forms of hyphenation automatically, greatly simplifying the task of entering songs.

One final difference between the chord-entry syntax used by the Song♭ook package and that that used by the songs package involves flat accidentals. The Song♭ook package allows the user to use "`b`" in a ⟨*chord*⟩ to produce a flat symbol, whereas the songs package requires an "`&`" instead. Although using "`b`" is probably more intuitive for the casual user, I elected not to support that syntax for an important technical reason. When "`b`" is redefined to produce a flat symbol in a ⟨*chord*⟩, the user can no longer use "`b`" for any other purpose within a ⟨*chord*⟩, such as to produce a literal "b" or to type another macro name like `\hbox` that contains a "b". Consequently, the songs package uses the less obvious `&` symbol to produce flat symbols.

**Song Structure.** The Song♭ook package has more detailed support than the songs package for typesetting material that reflects the high-level structure of a song. There are special mechanisms for typesetting intros, bridges, brackets, endings, and the like. The songs package is comparatively simplistic in the sense that it provides a smaller number of primitives that can be combined to duplicate this functionality. Textual instructions are producible by document authors using `\textnote` (§7.6), `\musicnote` (§7.6), and `\beginchordedonly` (§12.3) as building blocks.

**Stylistic Differences.** The Song♭ook package is really designed to produce chord books with one song per page, whereas the songs package is designed to produce chord books with many songs per page, arranged in two columns. Either approach might be the better one depending on which sorts of uses your song book is to be put. If you like to make photocopies of individual pages and hand them out to musicians, a book with one song per page might be more convenient. On the other hand, if you are printing entire chord books, one song per page can become very expensive. With the songs package, I've elected to pack as many songs as possible per page by default, but include options for generating one song per page (in particular see `\spenalty` and `\songcolumns`) and options for generating a specified subset of songs in a specified order (for producing hand-outs that can be photocopied and given to musicians for a particular service).

Song appearance also differs between the Song♭ook and songs packages. Since the songs package was designed for multiple columns per page, it includes a number of features not available in the Song♭ook package like automatic column balancing, completely customizable song header and song footer blocks, and facilities for adding beautiful scripture quotations to fill in gaps between songs.

**Indexes.** The Song♭ook package has a facility for automatically generating an index sorted by key, but the songs package does not. On the other hand, the songs package has a facility for automatically generating indexes sorted by scripture references but the Song♭ook package does not. I adopted the latter approach because while I personally have not found much use for indexes sorted by key, I have found a scripture index invaluable for planning services around particular sermons or topics. Both packages can generate indexes sorted by author, by title, and by notable lines of lyrics.

**Automatic Transposition.** The songs package has a facility for automatically transposing songs, and even generating chord books that print the chords in multiple keys (e.g., so that a pianist and guitarist using a capo can play together from the same book). I am not aware of any other song typesetting packages that include automatic transposition.

The Song♭ook package and the songs package were developed entirely independently. I originally developed the set of LaTeX macros that eventually became the songs package in order to typeset a song book for the Graduate Christian Fellowship (GCF) at Cornell University, and the Cornell International Christian Fellowship (CICF). Once I had fine-tuned my package to be sufficiently versatile, I decided to release it for public use. At that time I noticed the Song♭ook package and wrote this analysis of the differences between that package and mine.

For information on more song-typesetting resources, I recommend consulting the documentation provided with the Song♭ook package. It includes an excellent list of other resources that might be of interest to creators of song books.

## 15 GNU General Public License

### Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately

publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,