The songs package*

Kevin W. Hamlen

May 4, 2011

Abstract

The songs package produces songbooks that contain lyrics and chords (but not full sheet music). It allows lyric books, chord books, overhead slides, and digital projector slides to all be maintained and generated from a single LaTeX source document. Automatic transposition, guitar tablature diagrams, handout creation, and a variety of specialized song indexes are all supported.

1 Introduction

The songs LaTeX package produces books of songs that contain lyrics and (optionally) chords. A single source document yields a lyric book for singers, a chord book for musicians, and overhead or digital projector slides for corporate singing.

The software is especially well suited for churches and religious fellowships desiring to create their own books of worship songs. Rather than purchasing a fixed hymnal of songs, the songs package allows worship coordinators to maintain a constantly evolving repertoire of music to which they can add and remove songs over time. As the book content changes, the indexes, spacing, and other formatting details automatically adjust to stay consistent. Songs can also be quickly selected and arranged for specific events or services through the use of scripture indexes, automatic transposition, and handout and slide set creation features.

2 Terms of Use

*This document corresponds to songs v2.12, dated 2011/01/09,

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, 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 is 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 are 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.

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`) is 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`

Chords can also be turned on or off anywhere in the middle of the document by using the `\chordson` or `\chordsoff` macros.

`\slides`

Slides mode can be activated in the middle of the document by using the `\slides` macro. For best results, this should typically only be done in the document preamble or at the beginning of a fresh page.

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.

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.

Figure 1: Sample page from a chord book

nomeasures
showmeasures
\measureson
\measuresoff

Measure Bars. The songs package includes a facility for placing measure bars in chord books (see §7.7). To omit these measure bars, use the `nomeasures` option; to display them, use the `showmeasures` option (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{`$\langle n \rangle$`}` within a song environment produces a textual note in chord books that suggests the use of a guitar capo on fret $\langle n \rangle$. However, when the `transposecapos` option is active, these textual notes are omitted and instead the effect of `\capo{`$\langle n \rangle$`}` is the same as for `\transpose{`$\langle n \rangle$`}`. That is, chords between the `\capo` macro and the end of the song are automatically transposed up by $\langle n \rangle$ 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 capos. See §7.8 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 the auxiliary data files that are used to create indexes are not generated.

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

Columns. By default, songs in a `songs` environment are 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{`$\langle n \rangle$`}` macro can be used anywhere outside of `songs` environments to cause songs to be typeset in $\langle n \rangle$ columns per page (see §11.5).

noscripture
\scriptureon
\scriptureoff

Scripture Quotations. The `noscripture` option omits scripture quotations (see §8.2) from the output. 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{`$\langle$*songlist*$\rangle$`}` in the document preamble (i.e., before the `\begin{document}` line), where $\langle$*songlist*$\rangle$ is a comma-separated list of the song numbers to include. For example,

```
\includeonlysongs{37,50,2}
```

creates a document consisting only of songs 37, 50, and 2, in that order.

Partial books generated with `\includeonlysongs` omit all scripture quotations (§8.2), and ignore uses of `\nextcol`, `\brk`, `\sclearpage`, and `\scleardpage` between songs unless they are followed by a star (e.g., `\nextcol*`). To force a column- or page-break at a specific point in a partial book, add the word `nextcol`, `brk`, `sclearpage`, or `scleardpage` at the corresponding point in the ⟨*songlist*⟩ argument.

The `\includeonlysongs` macro only reorders songs within each `songs` environment (see §7), not between different `songs` environments. It also 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.

`\songchapter` When using the `book` document class, use `\songchapter` instead of the `\songsection` macro.

`\newindex` `\newauthorindex` `\newscripindex` **Indexes.** The songs package supports three kinds of indexes: indexes by title and/or notable lyrics, indexes by author, and indexes by scripture reference. 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⟩}
```

The ⟨*id*⟩ should be an alphabetic identifier that will be used to identify the index in other macros that reference it. The ⟨*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` are generated during the automatic index generation process. For example:

```
\newindex{mainindex}{idxfile}
```

creates a title index named "`mainindex`" whose data is stored in files named `idxfile.sxd` and `idxfile.sbx`.

`\showindex` To display the index in the document, use:

```
\showindex[⟨columns⟩]{⟨title⟩}{⟨id⟩}
```

where ⟨*id*⟩ is the same identifier used in the `\newindex`, `\newauthorindex`, or `\newscripindex` command, and where the ⟨*title*⟩ is the title of the index, which should consist only of simple text suitable for inclusion in the pdf bookmark index. (To change the formatting of the title, redefine the `\songsection` or `\songchapter` macro.) The `[`⟨*columns*⟩`]` part is optional, and if specified dictates the number of columns that should be used if the index can't fit in a single column.

For example:

```
\showindex{Index of Song Titles}{mainindex}
```

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. 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 contains the lines

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

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

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

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

```
pdflatex mybook.tex
```

7 Songs

7.1 Beginning a Song

songs **Song Sets.** Songs are contained within `songs` environments. Each `songs` 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 inter-song environments (see §8). No text in a `songs` environment may lie outside of these environments.

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

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

Songs should appear only within `songs` environments (see above). (If they do not, see §11.5 for how to create songs using your own page-builder.)

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.

The `[`⟨*otherinfo*⟩`]` part is optional. If provided, it consists of 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*⟩`}`	*authors, composers, and other contributors*
`cr={`⟨*copyright*⟩`}`	*copyright information*
`li={`⟨*license*⟩`}`	*licensing information*
`sr={`⟨*refs*⟩`}`	*related scripture references*
`index={`⟨*lyrics*⟩`}`	*an extra index entry for a line of lyrics*
`ititle={`⟨*title*⟩`}`	*an extra index entry for a hidden title*

For example, a song that begins and ends with

```
\beginsong{Title1 \\ Title2}[by={Joe Smith}, sr={Job 3},
  cr={\copyright~2011 XYZ.}, li={Used with permission.}]
\endsong
```

looks like

The four keyvals used in the above example are described in detail in the remainder of this section; the final two are documented in §7.9. You can also create your own keyvals (see §11.8).

by= **Song Authors.** The `by={`⟨*authors*⟩`}` keyval lists one or more authors, composers, translators, etc. An entry is 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}
```

Words separated by a macro-space (`\␣`) or tie (`~`) instead of a regular space are treated as single words by the indexer. For example, `The␣Vienna␣Boys'␣Choir` is indexed as "Choir, The Vienna Boys'" but `The␣Vienna\␣Boys'\␣Choir` is indexed as "Vienna Boys' Choir, The".

cr= **Copyright Info.** The `cr={`⟨*copyright*⟩`}` keyval specifies the copyright-holder of the song, if any. For example:

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

Copyright information is typeset in fine print at the bottom of the song.

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 is 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.

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

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 are 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 is

```
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*⟩).

7.2 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.

To create an unnumbered verse, begin the verse with `\beginverse*` instead of `\beginverse`. 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. Each line of the source document produces a separate line in the resulting document (like LaTeX's `\obeylines` macro). Lines that are too long to fit are wrapped with hanging indentation of width `\parindent`.

`\repchoruses` **Repeating Choruses.** When making overhead slides, it is often convenient to repeat the song's chorus after the first verse 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 causes 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 is repeated. (A set of choruses is assumed to consist of things like pre-choruses that should always be repeated along with the chorus.) Choruses are not 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 covers the common cases, but some songs have more complex forms that don't fit the typical verse, chorus, verse, chorus pattern. The `\repchoruses` feature is not always 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 inserts 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

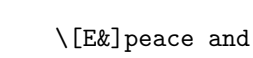
```
\ifslides\repchoruses\fi
```

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

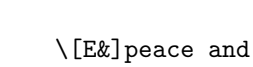
7.3 Chords

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

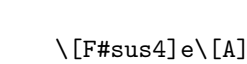
Any text that immediately follows the `\[]` macro with no intervening whitespace is assumed to be the word or syllable that is to be sung as the chord is struck, and is therefore typeset directly under the chord. For example:

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

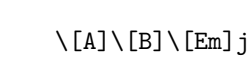
If whitespace (a space or end-of-line) immediately follows, then the chord name be typeset without any lyric text below it, indicating that the chord is to be struck between any surrounding words. For example:

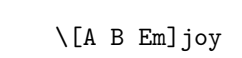
`\[E&]peace and \[Am] joy` *produces* 

If the lyric text that immediately follows the chord contains 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]e\[A]ternal` *produces* 

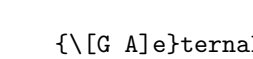
Sequences of chords that sit above a single word can be written back-to-back with no intervening space, or as a single chord:

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

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

The only difference between the two examples above is that the chords in the first example can later be replayed separately (see §7.4) 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* 

(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'".

`\nolyrics` **Chords Without Lyrics.** Sometimes you may want to write a line of chords with no lyrics in it at all, such as for an instrumental intro or solo. To make the chords in such a line sit on the baseline instead of raised above it, use the `\nolyrics` macro. For example:

`{\nolyrics Intro: \[G] \[A] \[D]}` *produces* Intro: G A D

Note the enclosing braces that determine how long the effect should last. Multiple lines can be included in the braces, or if the entire verse or chorus has no lyrics then the braces can be omitted. Instrumental solos should typically not appear in lyric books, so such lines should usually also be surrounded by `\ifchorded` and `\fi` (see §11.4).

`\DeclareLyricChar` **Symbols Under Chords.** If you are typesetting songs in a language whose alphabet contains symbols that LaTeX treats as punctuation, you can use the `\DeclareLyricChar` macro to instruct the **songs** package to treat the symbol as non-chord-ending, so that it is included under chords by default 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 are pushed out away from long chord names so that they never fall under a chord, and hyphenation is 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 are pushed out away from long chord names so that they never fall under the chord, but hyphenation is not 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 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 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 might be misleading to musicians if all three chords are intended to be played while singing the word "me." Liberal use of braces is therefore required to make \MultiwordChords produce good results, which is why it isn't the default.

\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}
```

7.4 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
```

produces

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 can be used in combination with automatic transposition to produce modulated verses. 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. Subsequent verses or choruses that use `^` replay chords from the most recently memorized verse or chorus.

Selective Memorization. It is also possible to inject unmemorized chords into a memorized verse so that they are not later 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
```

produces

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 and `^` replays chords from that most recently memorized sequence. However, you can memorize and replay multiple independent sequences using the macros described in the following paragraphs.

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

`\newchords{`⟨*regname*⟩`}`

where ⟨*regname*⟩ is any unique alphabetic name.

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 replay chord from a particular register, type

`\replay[`⟨*regname*⟩`]`

Subsequent uses of `^` reproduce chords from the sequence stored in register ⟨*regname*⟩.

Register contents are global, so you can memorize a chord sequence from one song and replay it in others. You can also use `\replay` multiple times in the same verse or chorus to replay a sequence more than once.

7.5 Line and Column Breaks

`\brk` **Line Breaking.** To cause a long line of lyrics to be broken in a particular place, put the `\brk` macro at that point in the line. This does not affect lines short enough to fit without breaking. 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
```

produces

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

Column Breaks Within Songs. To suggest a column break within a verse or chorus too long to fit in a single column, use `\brk` on a line by itself. If there are no `\brk` lines in a long verse, it is 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 overflows the column, yielding an overfull vbox warning.

`\nextcol` `\sclearpage` `\scleardpage` **Column Breaks Between Songs.** To force a column break between songs, use `\nextcol`, `\brk`, `\sclearpage`, or `\scleardpage` between songs. The `\nextcol` macro ends the column by leaving blank space at the bottom. The `\brk` macro ends the current column in lyric books by stretching the preceeding text so that the column ends flush with the bottom of the page. (In non-lyric books `\brk` is identical to `\nextcol`.) The `\sclearpage` macro is like `\nextcol` except that it shifts to the next blank page if the current page is nonempty. The `\scleardpage`

macro is like `\sclearpage` except that it shifts to the next blank even-numbered page in two-sided documents. Column breaks usually need to be in different places in different book types. To achieve this, use a conditional block from §11.4. For example,

```
\ifchorded\else\ifslides\else\brk\fi\fi
```

forces a column break only in lyric books but does not affect chord books or books of overhead slides.

When a partial list of songs is being extracted with `\includeonlysongs`, `\brk`, `\nextcol`, `\clearpage`, and `\cleardpage` macros between songs must be followed by a star to have any effect. To force a column-break at a specific point in a partial book, add the word `nextcol`, `brk`, `clearpage`, or `cleardpage` at the corresponding point in the argument to `\includeonlysongs`.

7.6 Echoes and Repeats

`\echo` **Echo Parts.** To typeset an echo part, use `\echo{`⟨*lyrics and chords*⟩`}`. Echo parts are parenthesized and italicized. For example,

`Alle\[G]luia! \echo{Alle\[A]luia!}` *produces* Alle^{G}luia! *(Alle^{A}luia!)*

`\rep` **Repeated Lines.** To indicate that a line should be sung multiple times by all singers, 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* ‖:^{G}Alleluia!:‖(×4)

7.7 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|^{G}luia

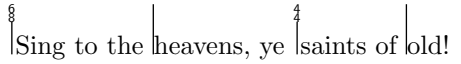
In order for measure bars to be displayed, the `showmeasures` option must be enabled. Measure bars are only displayed by default in chord books.

`\meter` The first measure bar in a song has meter numbers placed above it to indicate the time signature of the piece. By default, these numbers are 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
```

produces



7.8 Textual Notes

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

`\textnote{`⟨*text*⟩`}`

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

`\musicnote{`⟨*text*⟩`}`

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

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

produces

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 capos 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 instead has the effect of `\transpose{`⟨*n*⟩`}`. See §10 for more information on automatic chord transposition.

7.9 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}]
```

causes 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*⟩`}` are 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. If specified, ⟨*indexes*⟩ is a comma-separated list of the identifiers of indexes to which the entry should be added. Otherwise the new entry is added to all of the title indexes for the current `songs` environment.

7.10 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.

`\ch` When a chord macro falls inside a ligature, LATEX fails to compact the ligature into a single font character even in non-chorded versions of the book. To avoid this minor typograhpical error, use the `\ch` macro to typeset the chord:

`\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. 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*G*fficult

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 §7.4), 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*G*fficult

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* di§6 8*G*fficult

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. Thus, `dif|^ficult` requires no special treatment; it leaves the "ffi" ligature intact when measure bars are not being displayed.

8 Between Songs

Never put any material directly into the top level of a `songs` environment. Doing so will disrupt the page-builder, usually producing strange page breaks and blank pages. To safely put material between songs, use one of the environments described in this section.

8.1 Intersong Displays

`intersong` To put column-width material between the songs in a `songs` environment, use an `intersong` environment. Material contributed in an `intersong` environment is subject to the same column-breaking rules as songs (see §11.5), but all other formatting is up to you. For example, to put a LaTeX picture between two songs, you could write:

```
\begin{intersong}
  \begin{picture}(50,200)
  ⋮
  \end{picture}
\end{intersong}
```

By default, LaTeX inserts interline glue below the last line of an `intersong` environment. To suppress this, end the `intersong` content with `\par\nointerlineskip`.

`intersong*` To instead put page-width material above a song, use an `intersong*` environment. This starts a new page if the current page already has column-width material in it. For example, to put a page-width LaTeX picture atop the next song, write:

```
\begin{intersong*}
  \begin{picture}(100,200)
  ⋮
  \end{picture}
\end{intersong*}
```

`songgroup` By default, all intersong displays are omitted when generating a partial book with `\includeonlysongs`. You can force them to be included whenever a particular song is included by using a `songgroup` environment:

```
\begin{songgroup}
⋮
\end{songgroup}
```

Each `songgroup` environment may include any number of `intersong`, `intersong*`, or scripture quotations (see §8.2), but must include exactly one song. When using `\includeonlysongs`, the entire group is included in the book if the enclosed song is included; otherwise the entire group is omitted.

8.2 Scripture Quotations

`\beginscripture` `\endscripture` **Starting a Scripture Quotation.** A special form of intersong block typesets a scripture quotation. Scripture quotations begin and end with

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

where ⟨*ref*⟩ is a scripture reference that is 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).

The text of the scripture quotation between the `\beginscripture` and `\endscripture` lines are 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
```

produces

> *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", 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 `\Acolon` and each additional line with `\Bcolon`. Each line of the tuplet appears 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
```

produces

> *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
```

produces

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
```

produces

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 Guitar Tablatures

`\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 may omit any colon that borders an omitted argument).

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

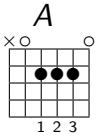
⟨*fret*⟩ is usually omitted, but if the top row of the diagram is intended to represent a fret other than the first one, then ⟨*fret*⟩ should be the number of the fret it represents (any number from 2 to 9).

⟨*strings*⟩ should be a series of 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 `9` if that string is to be played on the given numbered fret.

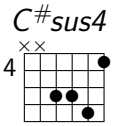
⟨*fingering*⟩ should either be empty if no fingering information is to be given, or it should consist of a series of digits, one for each string of the guitar from lowest pitch to highest. Each digit should be one of: `0` if no fingering information is to be displayed for that string (e.g., if the string 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.

Here are some examples to illustrate:

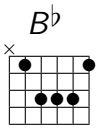
`\gtab{A}{X02220:001230}` *produces*



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



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



`\minfrets` By default, tablature diagrams always consist of at least 4 fret rows (more if the ⟨*strings*⟩ argument contains a number larger than 4). To change the minimum number of fret rows, change the value of `\minfrets`. For example, typing

```
\minfrets=1
```

causes tablature diagrams to have only as many rows are required to accommodate the largest digit appearing in the ⟨*strings*⟩ argument.

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{`$\langle n \rangle$`}`

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 appears as an E in the resulting document. Specifying a negative number for $\langle n \rangle$ transposes subsequent chords down instead of up.

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

When the `transposecapos` option is active, the `\capo` macro acts like `\transpose`. See §7.8 for more information.

`\preferflats` `\prefersharps`

Enharmonics. When using `\transpose` to automatically transpose the chords of a song, the **songs** package code chooses 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 are 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 produces 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 §7.3) 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

$F^\#$ $B/F^\#$ $F^\#$ $E^\flat 7$
This is a memorized verse.

$A^\flat$ $D^\flat/A^\flat$ $A^\flat$
This verse is modulated up two half-steps.

`\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 together so that musicians playing differently-tuned instruments can play from the same piece of music. This can be achieved by redefining the `\trchordformat`

macro, which expects two arguments—the original chord name and the transposed chord name, respectively. 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, use:

```
\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 looks 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 uppercase 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 fails to transpose it; the resulting chord is still 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}
```

sets the next song's number to be 3.

\thesongnum

You can change the song numbering style for a song section by redefining `\thesongnum`. For example, to cause songs to be numbered A1, A2, etc., in the current song section, type

```
\renewcommand{\thesongnum}{A\arabic{songnum}}
```

The expansion of `\thesongnum` must always produce plain text with no font formatting or unexpandable macro tokens, since that text is exported to auxiliary index generation files where it is sorted alphabetically.

Any `\ref` that refers to a `\label` that appears within a song (but outside a numbered verse) yields that song's number as typeset by `\thesongnum`.

\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 `\thesongnum` as its only argument. For example, to typeset song numbers in italics atop each song, define

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

\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}
```

If `\songnumwidth` is set to zero, song numbers are not shown at all.

\nosongnumbers

To turn off song numbering entirely, type `\nosongnumbers`. This inhibits the display of the song number atop each song (but song numbers are still be displayed elsewhere, such as in indexes). The same effect can be achieved by setting `\songnumwidth` to zero.

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}
```

sets the next verse's number to be 3.

\theversenum You can change the verse numbering style by redefining `\theversenum`. For example, to cause verses to be numbered in uppercase roman numerals, define

```
\renewcommand{\theversenum}{\Roman{versenum}}
```

Any `\ref` that refers to a `\label` that appears within a numbered verse yields that verse's number as typeset by `\theversenum`.

\printversenum To change the formatting of verse numbers as they appear at the beginning of each verse, you should redefine the `\printversenum` macro, which expects the text yielded by `\theversenum` 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 is shifted right by this amount. For example, to reserve half a centimeter of space for verse numbers, define

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

Verse numbers whose widths exceed `\versenumwidth` indent the first line of the verse an additional amount to make room, but subsequent lines of the verse are only 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. Authors interested in changing the placement of verse numbers should consult §16.2 of the implementation section for more information on this macro.

11.2 Song Appearance

\lyricfont **Font Selection.** By default, lyrics are typeset using the document-default font (`\normalfont`) and with the document-default point size (`\normalsize`). You can change these defaults 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 titles to be typeset in a roman font, you could define

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

\versefont
\chorusfont
\notefont

You can apply additional font changes to verses, choruses, and textual notes produced with `\textnote` and `\musicnote` by redefining `\versefont`, `\chorusfont`, and `\notefont`, respectively. For example, to typeset choruses in italics, you could define

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

\notebgcolor
\snumbgcolor

The colors of shaded boxes containing textual notes and song numbers can be changed by redefining the `\notebgcolor` and `\snumbgcolor` macros. For example:

```
\renewcommand{\notebgcolor}{red}
```

\printchord

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

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

\sharpsymbol
\flatsymbol

Accidental Symbols. By default, sharp and flat symbols are typeset using LaTeX's `\#` (#) and `\flat` (♭) macros. Users can change this by redefining `\sharpsymbol` and `\flatsymbol`. For example, to use `\sharp` (♯) instead of #, one could redefine `\sharpsymbol` as follows.

```
\renewcommand{\sharpsymbol}{\ensuremath{^\sharp}}
```

\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}
```

Song Top and Bottom Material. You can adjust the header and footer material that precedes and concludes each song by redefining \extendprelude and \extendpostlude.

\extendprelude \showauthors \showrefs By default, \extendprelude displays the song's authors and scripture references using the macros \showauthors and \showrefs. The following definition changes it to also print copyright info:

```
\renewcommand{\extendprelude}{
  \showrefs\showauthors
  {\bfseries\songcopyright\par}
}
```

\extendpostlude By default, \extendpostlude prints the song's copyright and licensing information as a single paragraph using \songcopyright and \songlicense. The following definition changes it to also print the words "Used with permission" at the end of every song's footer information:

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

In general, any macro documented in §12 can be used in \extendprelude and \extendpostlude to print song information, such as \songauthors, \songrefs, \songcopyright, and \songlicense. For convenience, the \showauthors and \showrefs macros display author and scripture reference information as a preformatted paragraph the way it appears in the default song header blocks.

See §11.8 for how to define new \beginsong keyvals and use them in \extendprelude.

\makeprelude \makepostlude 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.

`\vvpenalty` `\ccpenalty` `\vcpenalty` `\cvpenalty` `\brkpenalty`

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 `\interlinepenalty` are inserted between consecutive lines of each verse and chorus; 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; and penalties of value `\brkpenalty` are inserted wherever `\brk` is used on a line by itself. The higher the penalty, the less likely TEX is to place a page- or column-break at that penalty. If any are set to -10000 or lower, breaks are forced there. By default, `\interlinepenalty` is set to 1000 and the rest are set to 200 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 is 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
```

`\versejustify` `\chorusjustify` `\justifyleft` `\justifycenter`

Text Justification. To cause verse or chorus text to be justified flush-left or centered, set `\versejustify` or `\chorusjustify` 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). Authors interested in changing this behavior should consult §16.2 of the implementation section for more information about this macro.

`\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`. Authors interested in changing this behavior should consult §16.2 of the implementation section for more information about this macro.

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
`nolyrics`	the `\nolyrics` macro is in effect
`vnumbered`	the current verse is numbered (i.e., it was started with `\beginverse` instead of `\beginverse*`)

Table 1: Conditional macros

11.3 Scripture Appearance

`\scripturefont` By default, scripture quotations are 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, define:

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

`\printscrcite` By default, the citation at the end of a scripture quotation is 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 argument. For example, to cause citations to be printed in roman italics font, define:

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

11.4 Conditional Blocks

Conditional macros allow certain material to be included in some books but not others. For example, a musician's chord book might include extra verses with alternate chordings.

`\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`. Between the `\if`⟨*type*⟩ and the `\fi` may also appear an `\else`. 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.

11.5 Page Layout

`\songcolumns` The number of columns per page can be set with `\songcolumns`. For example, to create 3 columns per page, write

```
\songcolumns{3}
```

The number of columns should only be changed outside of `songs` environments.

Setting the number of columns to zero disables the page-building algorithm entirely. This can be useful if you want to use an external package, such as `multicol` or LaTeX's built-in `\twocolumn` macro, to build pages. For example, the following sets up an environment that is suitable for a lyric book that uses `\twocolumn`:

```
\songcolumns{0}
\flushbottom
\twocolumn[\LARGE\centering My Songs]
\begin{songs}{}
⋮
\end{songs}
```

When disabling the page-builder, please note the following potential issues:

- The `\repchoruses` feature does not work when the page-builder is disabled because the page-builder is responsible for inserting repeated choruses as new columns are formed.
- External page-building packages tend to allow column- and page-breaks within songs because they have no mechanism for moving an entire song to the next column or page to avoid such a break (see `\songpos` below).
- Indexes produced with `\showindex` are typeset to the width of the enclosing environment. Thus, you should be sure to reset LaTeX back to one column (via `\onecolumn`) before executing `\showindex`.

`\columnsep` The horizontal distance between consecutive columns is controlled by the `\columnsep` dimension. For example, to separate columns by 1 centimeter of space, write

```
\columnsep=1cm
```

\colbotglue When LaTeX ends each column it inserts glue equal to \colbotglue. In lyric books this macro is set to 0pt so that each column ends flush with the bottom of the page. In other books that have ragged bottoms, it is set to stretchable glue so that columns end at whatever vertical position is convenient. The recommended setting for typsetting columns with ragged bottoms is:

```
\renewcommand{\colbotglue}{0pt plus .5\textheight minus 0pt}
```

\lastcolglue The last column in a songs environment gets \lastcolglue appended to it instead. By default it is infinitely stretchable so that the last column ends at its natural height. By setting it to 0pt, you can force the last column to be flush with the bottom of the page:

```
\renewcommand{\lastcolglue}{0pt}
```

\songpos The songs package uses a song-positioning algorithm that moves songs to the next column or page in order to avoid column- or page-breaks within songs. The algorithm has four levels of aggressiveness, numbered from 0 to 3. You can change the aggressiveness level by typing

\songpos{⟨*level*⟩}

The default level is 3, which avoids column-breaks, page-breaks, and page-turns within songs whenever possible. (Page-turns are page-breaks after odd-numbered pages in two-sided documents, or after all pages in one-sided documents.) Level 2 avoids page-breaks and page-turns but allows column-breaks within songs. Level 1 avoids only page-turns within songs. Level 0 turns off the song-positioning algorithm entirely. This causes songs to be positioned wherever TeX thinks is best based on penalty settings (see \vvpenalty and \spenalty).

\spenalty The value of \spenalty controls the undesirability of column breaks at song boundaries. Usually it should be set to a value between 0 and \vvpenalty so that breaks between songs are preferable to breaks between verses within a song. By default it is set to 100. When it is -10000 or less, breaks between songs are required, so that each song always begins a fresh column.

11.6 Indexes

\indexsongsas **Index Appearance.** By default, the right-hand side of each index entry contains a list of one or more song numbers. To instead list page numbers, use the \indexsongsas macro:

\indexsongsas{⟨*id*⟩}{\thepage}

where ⟨*id*⟩ is the same identifier used in the \newindex, \newauthorindex, or \newscripindex macro that created the index. The second argument must always be something that expands into raw text without any formatting, since this text gets output to auxiliary files that are lexographically sorted by the index-generation program. To go back to indexing songs by song number, use \thesongnum in place of \thepage in the above.

\idxrefsfont To control the formatting of the list of references on the right-hand side of

index entries, redefine `\idxrefsfont`. For example, to typeset each list in boldface, write

```
\renewcommand{\idxrefsfont}{\bfseries}
```

`\idxtitlefont` `\idxlyricfont`

Title indexes contain entries for song titles and also entries for notable lines of lyrics. The fonts for these entries are controlled by `\idxtitlefont` and `\idxlyricfont`, respectively. For example, to show title entries in boldface sans-serif and lyric entries in regular roman font, one could define:

```
\renewcommand{\idxtitlefont}{\sffamily\bfseries}
\renewcommand{\idxlyricfont}{\rmfamily\mdseries}
```

`\idxheadfont`

To change the font used to typeset the capital letters that start each alphabetic section of a large title index, redefine `\idxheadfont`. For example, to typeset those letters in italics instead of boldface, type

```
\renewcommand{\idxheadfont}{\sffamily\it\LARGE}
```

`\idxbgcolor`

To change the background color of the shaded boxes that contain the capital letters that start each alphabetic sectino of a large title index, redefine `\idxbgcolor`. For example:

```
\renewcommand{\idxbgcolor}{red}
```

`\idxheadwidth`

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

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

`\idxauthfont`

The font used to typeset entries of an author index is controlled by `\idxauthfont`. For example, to typeset such entries in italics instead of boldface, type

```
\renewcommand{\idxauthfont}{\small\it}
```

`\idxscripfont`

The font used to typeset entries of a scripture index is controlled by `\idxscripfont`. For example, to typeset such entries in boldface instead of italics, type

```
\renewcommand{\idxscripfont}{\sffamily\small\bfseries}
```

`\idxbook`

To control the formatting of the lines that start each new book of the bible in a scripture index, redefine `\idxbook`, which accepts the book name as its single argument. For example, to typeset each book name in a box, one could define

```
\renewcommand{\idxbook}[1]{\framebox{\small\bfseries#1}}
```

`\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 receives the ⟨*bookname*⟩ as its single argument. For example, to typeset an index in German, one might define

```
\renewcommand{\idxcont}[1]{\small\textbf{#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" is typically indexed as "Song Title, The". To change this default behavior, you can use `\titleprefixword` in the document preamble to identify each word to 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 but may be used multiple times to declare multiple prefix words.

\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` and each of the following macros 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 and each of the following macros may only be used in the document preamble but may be used multiple times to declare multiple special words.

\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" is 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}
```

\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}
```

11.7 Page Headers and Footers

In LaTeX, page headers and footers are defined using a system of invisible *marks* that get inserted into the document at the beginning of each logical unit of the document (e.g., each section, song, verse, and chorus). The headers and footers are then defined so as to refer to the first and/or last invisible mark that ends up on each page once the document is divided into pages. This section describes the marks made available by the **songs** package. For more detailed information about the marks already provided by LaTeX and how to use them, consult any LaTeX user manual.

`\songmark` `\versemark` `\chorusmark`

To add song information to page headings and footers, 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. To access the current song's number or the current verse's number, use `\thesongnum` or `\theversenum` (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{\thesongnum}{\thesongnum}}
```

11.8 Defining New Beginsong Keyvals

`\newsongkey`

The `\beginsong` macro supports several optional keyval parameters for declaring song information, including `by=`, `sr=`, and `cr=`. 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 is 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 is executed whenever ⟨*key*⟩ is parsed as part of the `\beginsong` keyval arguments. In ⟨*setcode*⟩, `#1` expands to the value given by the user for 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{\arranger}{}
\newsongkey{arr}{\def\arranger{}}
                {\def\arranger{Arranged by #1\par}}
```

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

```
\renewcommand{\extendprelude}{
  \showrefs\showauthors
  {\bfseries\arranger}
}
```

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

```
\beginsong{The Title}[arr={R. Ranger}]
⋮
\endsong
```

This produces

1 *The Title*
Arranged by R. Ranger

For more detailed information about keyvals and how they work, consult the documentation for David Carlisle's `keyval` package, which comes standard with most LaTeX 2_ε installations.

11.9 Font Kerning Corrections

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` The best solution to this problem 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. (See §16.2 for documentation of the \chordlocals hook.) 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 (Gsus4/D) Overstrike (fixed)

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

\shiftdblquotes **Scripture Font Quotation Marks.** The songs package compensates for a kerning problem in the Zaph Chancery font (used to typeset scripture quotations) by redefining the `` and '' token sequences to be active characters that yield double-quotes shifted 1.1 points and 2 points left, respectively, of their normal positions. If you use a different font size for scripture quotations, then you can use the \shiftdblquotes macro when redefining \scripturefont to change this kerning correction. For example,

```
\renewcommand{\scripturefont}{
  \usefont{OT1}{pzc}{mb}{it}
  \shiftdblquotes{-1pt}{-2pt}{-3pt}{-4pt}
}
```

removes 1 point of space to the left and 2 points of space to the right of left-double-quote characters, and 3 points to the left and 4 points to the right of right-double-quotes, within scripture quotations.

12 Informational Macros

The macros described in this section can be used to retrieve information about the current song. This can be used when redefining \extendprelude, \extendpostlude, \makeprelude, \makepostlude, \songmark, \versemark, or \chorusmark, or any other macros that might typeset this information.

\songauthors To get 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 get the current song's list of scripture references (if any) use \songrefs. This yields the value of the sr= key used in the \beginsong line, but modified with hyphens changed to en-dashes and spaces falling within a list of verse numbers changed to thin spaces for better typesetting. In addition, various penalties have been added to inhibit line breaks in strange places and encourage line breaks in others.

\songcopyright To get 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 get 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 declared with \setlicense.

\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 get 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 get the song's next title, 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` The `\foreachtitle` macro accepts LaTeX code as its single argument and executes it once for each (remaining) song title. Within the provided code, use `\songtitle` to get the current title. For example, the following code generates a comma-separated list of all of the current song's titles:

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

`\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 alters the list of songs to be extracted. Redefining it after the preamble may have unpredictable results.

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 §11.6.

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` is written out for each ⟨*filename*⟩ defined using `\newindex`, `\newauthorindex`, or `\newscripindex`. These `.sxd` files are plain text 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 to be indexed (a title, author, or scripture reference). The second line has the song's number in the book (yielded by `\thesongnum`). 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. Since the standard `makeindex` program provided with LaTeX is not powerful enough to sort scripture references, distributions of `songs` package come with a specialized `songidx` program to do this.

3. The `.sbx` files produced by the `songidx` 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 consist of a series of `idxblock` environments, one for each such section. An `idxblock` environment begins and ends with

```
\begin{idxblock}{⟨letter⟩}
⋮
\end{idxblock}
```

where ⟨*letter*⟩ is the letter of the alphabet for that block.

`\idxentry` `\idxaltentry` The index entries themselves are created by 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` macro 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 produces 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 Resources

There are a number of other LaTeX packages available for typesetting songs, tablature diagrams, or song books. Probably the best of these is the Songbook package by Christopher Rath (`http://rath.ca/Misc/Songbook/`). Most of the differences between other packages 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. Much of the songs package programming is devoted to easing the burden of typing chords. With most LaTeX song book packages the user types chords using a standard LaTeX macro syntax like `\chord{`⟨*chord*⟩`}{`⟨*lyric*⟩`}`. The songs package uses a less conventional `\[`⟨*chord*⟩`]`⟨*lyric*⟩ syntax for several reasons detailed below.

First, macros in the standard LaTeX syntax require more key-presses than macros in the songs package's syntax. This can become become very 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 requires the user to estimate how much of the ⟨*lyric*⟩ will lie below the chord (because the ⟨*lyric*⟩ part must be enclosed in braces) whereas the songs package's syntax does not. Estimating this accurately can be quite difficult, since in many cases the ⟨*lyric*⟩ part must include punctuation or multiple words to get proper results. The songs package automates this for the user, significantly easing the task of chord-entry.

Third, unlike the standard LaTeX chord syntax, the songs package's syntax handles all hyphenation of chorded lyrics fully automatically. Extra hyphenation must be introduced in chord books wherever a chord is wider than the syllable it sits above. With the standard LaTeX chord syntax such hyphenation must be introduced manually by the user (usually via a special hyphenation macro), but the songs package does this automatically.

Fourth and finally, some other packages allow the user to use "`b`" in a ⟨*chord*⟩ to produce a flat symbol, whereas the songs package requires an "`&`" instead. Using "`b`" is more intuitive but prevents the use of "`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 songs package provides a relatively small number of macros for typesetting high-level song structure, including verses, choruses, textual comments, and conditional macros that indicate that certain sections should go in chord books but not lyric books. These can be combined to typeset more sophisticated structures such as intros, bridges, brackets, endings, and the like. This is done in lieu of providing a specific macro for each of these structures since it results in greater flexibility and fewer macros for users to learn.

Multiple columns. The songs package was designed from the ground up to produce song books with many songs per page, arranged in multiple columns. As a result, it includes elaborate support for many features not found in most other packages, such as 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. Another major feature of the songs package is its support for a variety of different index types, most notably indexes arranged by scripture reference. Scripture indexes can be invaluable for planning services around particular sermons or topics. The songs package allows book authors to specify the names and preferred ordering of books of the bible, and automatically handles complex issues like overlapping verse ranges to produce an easy-to-read, compact, and well-ordered index. Other supported indexes include those 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).

The songs package was developed entirely independently of all other LaTeX song book packages. 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 Songbook package and others, and wrote this summary of the most prominent differences.

For information on more song-typesetting resources for LaTeX, I recommend consulting the documentation provided with the Songbook 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,

(c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program

(INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

16 Implementation

The following provides the verbatim implementation of the songs LaTeX package, along with some brief commentary on how it works. In general, macro names that contain a @ symbol are not intended to be directly accessible by the outside world; they are for purely internal use. All other macros are intended to be used or redefined by document authors.

Most of the macros likely to be of real interest to song book authors can be found in §16.2. To find the implementation of any particular macro, the index at the end of this document should prove helpful.

16.1 Initialization

The code in this section detects any TeX versioning or configuration settings that are relevant to the rest of the song book code.

\ifSB@etex Numerous enhancements are possible when using an ε-TeX compatible version of LaTeX. We start by checking to see if ε-TeX primitives are available.

```
\newif\ifSB@etex
\ifx\eTeXversion\undefined\else
  \ifx\eTeXversion\relax\else
    \SB@etextrue
    \IfFileExists{etex.sty}{\RequirePackage{etex}}{}
  \fi
\fi
```

\ifSB@pdf Figure out if we're generating a pdf file or not.

```
\newif\ifSB@pdf\SB@pdffalse
\ifx\pdfoutput\undefined\else
  \ifx\pdfoutput\relax\else
    \ifnum\pdfoutput<\@ne\else
      \SB@pdftrue
    \fi
  \fi
\fi
```

\ifSB@preamble Document processing starts in the preamble. Some macros have different effects depending on when they're used in the preamble or in the document body, so set this to true for now and change it to false at the end of the preamble.

```
\newif\ifSB@preamble
\SB@preambletrue
```

\ifSB@test Reserve some control sequence names for scratch use.
\SB@temp
\SB@tempii
\SB@tempiii
\SB@tempiv
\SB@tempv

```
\newif\ifSB@test
\newcommand\SB@temp{}
\newcommand\SB@tempii{}
\newcommand\SB@tempiii{}
\newcommand\SB@tempiv{}
\newcommand\SB@tempv{}
```

\SB@dimen
\SB@dimenii
\SB@dimeniii
\SB@dimeniv
\SB@box
\SB@boxii
\SB@boxiii
\SB@toks
\SB@cnt
\SB@cntii
\SB@skip

Reserve some temp registers for various purposes.

```
\newdimen\SB@dimen
\newdimen\SB@dimenii
\newdimen\SB@dimeniii
\newdimen\SB@dimeniv
\newbox\SB@box
\newbox\SB@boxii
\newbox\SB@boxiii
\newtoks\SB@toks
\newcount\SB@cnt
\newcount\SB@cntii
\newskip\SB@skip
```

Load David Carlisle's `keyval` package for processing ⟨*key*⟩=⟨*value*⟩ style macro arguments.

```
\RequirePackage{keyval}
```

16.2 Default Parameters

This section defines macros and lengths that will typically be executed or redefined by the user in the document preamble to initialize the document. (Not all of these are restricted to preamble usage, however. Many can be used throughout the document to switch styles for different sections or different songs.)

\lyricfont

Define the font style to use for formatting song lyrics.

```
\newcommand\lyricfont{\normalfont\normalsize}
```

\stitlefont

Define the font style to use for formatting song titles.

```
\newcommand\stitlefont{%
  \ifslides\sffamily\Huge\else\sffamily\slshape\Large\fi%
}
```

\versefont
\chorusfont
\notefont

By default, verses, choruses, and textual notes just allow the `\lyricfont` style to continue.

```
\newcommand\versefont{}
\newcommand\chorusfont{}
\newcommand\notefont{}
```

\scripturefont

Define the font style to use for formatting scripture quotations (defaults to Zapf Chancery).

```
\newcommand\scripturefont{%
  \usefont{OT1}{pzc}{mb}{it}%
  \shiftdblquotes{-1.1\p@}\z@{-2\p@}\z@%
}
```

\printscrcite

Define the printing style for the citation at the end of a scripture quotation.

```
\newcommand\printscrcite[1]{\sffamily\small#1}
```

\snumbgcolor
\notebgcolor
\idxbgcolor

Define the background color used for shaded boxes containing song numbers, textual notes, and index section headers, respectively. To turn off all shading for a box type, use \def⟨*macroname*⟩{}.

```
\newcommand\snumbgcolor{SongbookShade}
\newcommand\notebgcolor{SongbookShade}
\newcommand\idxbgcolor{SongbookShade}
```

\versejustify
\chorusjustify

Verses and choruses are both left-justified with hanging indentation equal to \parindent,

```
\newcommand\versejustify{\justifyleft}
\newcommand\chorusjustify{\justifyleft}
```

\notejustify

Textual notes will be fully justified when they are too long to fit in a single line.

```
\newcommand\notejustify{%
  \advance\baselineskip\p@\relax%
  \leftskip\z@skip\rightskip\z@skip%
  \parfillskip\@flushglue\parindent\z@%
}
```

\placenote

Textual notes are placed flush-left. The single argument to this macro is horizontal material that comprises the note. Usually it will consist of various hboxes and specials that were produced by \colorbox.

```
\newcommand\placenote[1]{%
  \leftskip\z@skip\rightskip\@flushglue\SB@cbarshift%
  \noindent#1\par%
}
```

These counters define the current song number and verse number. They can be redefined by the user at any time.

```
\newcounter{songnum}
\newcounter{versenum}
```

\thesongnum
\songnumstyle

By default, the song numbering style will simply be an arabic number. Redefine \thesongnum to change it. (The \songnumstyle macro is obsolete and exists only for backward compatibility.)

```
\renewcommand\thesongnum{\songnumstyle{songnum}}
\newcommand\songnumstyle{}
\let\songnumstyle\arabic
```

\theversenum
\versenumstyle

By default, the verse numbering style will simply be an arabic number. Redefine \theversenum to change it. (The \versenumstyle macro is obsolete and exists only for backward compatibility.)

```
\renewcommand\theversenum{\versenumstyle{versenum}}
\newcommand\versenumstyle{}
\let\versenumstyle\arabic
```

\printsongnum

Define the printing style for the large, boxed song numbers starting each song.

```
\newcommand\printsongnum[1]{\sffamily\bfseries\LARGE#1}
```

`\printversenum` Define the printing style for the verse numbers that lie to the left of each verse.

```
\newcommand\printversenum[1]{\lyricfont#1.\ }
```

`\placeversenum` Verse numbers are placed flush-left. This is achieved by inserting horizontal glue that reverses both the `\leftskip` and the `\parindent`. The single argument to this macro is an hbox containing the verse number.

```
\newcommand\placeversenum[1]{%
  \hskip-\leftskip\hskip-\parindent\relax%
  \box#1%
}
```

`\everyverse` `\everychorus` The following hooks allow users to insert material at the head of each verse or chorus.

```
\newcommand\everyverse{}
\newcommand\everychorus{}
```

`\printchord` Define the printing style for chords.

```
\newcommand\printchord[1]{\sffamily\slshape\large#1}
```

`\chordlocals` This hook is expanded at the start of the scoping group that surrounds every chord name. Thus, it can be used to set any catcodes or definitions that should be local to chord names.

```
\newcommand\chordlocals{}
```

`\versesep` Specify the vertical distance between song verses. This gets set to a sentinel value by default; if the user doesn't redefine it by the end of the document preamble, it gets redefined to something sensible based on other settings.

```
\newskip\versesep
\versesep123456789sp\relax
```

`\baselineadj` Define an adjustment factor for the vertical distance between consecutive lyric baselines. Setting this to zero accepts the default baseline distance computed by the songs package.

```
\newskip\baselineadj
\baselineadj\z@skip
```

`\parindent` The `\parindent` length controls how far broken lyric lines are indented from the left margin.

```
\parindent.25in
```

`\idxheadwidth` Specify the width of the head-boxes in a large index.

```
\newlength\idxheadwidth
\setlength\idxheadwidth{1.5cm}
```

`\songnumwidth` The width of the song number boxes will be the width of typesetting the text "999."

```
\newlength\songnumwidth
\settowidth\songnumwidth{\printsongnum{999.}}
```

\versenumwidth Reserve some space for each verse number based on the definition of the \printversenum macro.

```
\newlength\versenumwidth
\settowidth\versenumwidth{\printversenum{9\kern1em}}
```

\cbarwidth This dictates the width of the vertical line placed to the left of choruses. Setting it to 0pt eliminates the line entirely.

```
\newlength\cbarwidth
\setlength\cbarwidth\p@
```

\sbarheight This dictates the height of the horizontal line placed between each pair of songs. Setting it to 0pt eliminates the line entirely.

```
\newlength\sbarheight
\setlength\sbarheight\p@
```

Column- and page-breaks should typically not occur within a verse or chorus unless they are unavoidable. Thus, we set the \interlinepenalty to a high number.

```
\interlinepenalty\@m
```

\vvpenalty \ccpenalty \vcpenalty \cvpenalty \brkpenalty The following count registers define the penalties inserted between verses, between choruses, after a verse followed by a chorus, after a chorus followed by a verse, and at \brk macros, respectively.

The default value of 200 was chosen based on the following logic: Chord books should not yield underfull vbox warnings no matter how short their columns are. However, we still want to put as much material in each column as possible while avoiding intra-song column-breaks when they can be avoided. Chorded mode therefore sets \colbotglue with glue whose stretchability is half of the \textheight. Such glue will stretch at most twice its stretchability, yielding a badness of 800 in the worst case. The default \vbadness setting starts issuing warnings at badness 1000, so we set the penalties below to $1000 - 800 = 200$.

```
\newcount\vvpenalty\vvpenalty200
\newcount\ccpenalty\ccpenalty200
\newcount\vcpenalty\vcpenalty200
\newcount\cvpenalty\cvpenalty200
\newcount\brkpenalty\brkpenalty200
```

\spenalty The following penalty gets inserted between songs. Setting it to a proper value is a somewhat delicate balancing act. It should typically be something between 0 and the default penalties above, so for now it defaults to 100. To start each song on a fresh column/page, set it to -10000 or below.

```
\newcount\spenalty\spenalty100
```

\songmark \versemark \chorusmark The user can redefine the following macros to add page marks for each song, each verse, or each chorus.

```
\newcommand\songmark{}
\newcommand\versemark{}
\newcommand\chorusmark{}
```

\extendprelude
\extendpostlude

To just add some fields to the existing \makeprelude or \makepostlude without having to redefine them entirely, users can redefine \extendprelude or \extendpostlude. By default, the prelude has the scripture references followed by the authors, and the postlude has the copyright info followed by the licensing info.

```
\newcommand\extendprelude{\showrefs\showauthors}
\newcommand\extendpostlude{\songcopyright\ \songlicense\unskip}
```

\idxheadfont

Users can redefine \idxheadfont to affect the font in which each capital letter that heads a section of a title index is rendered.

```
\newcommand\idxheadfont{\sffamily\bfseries\LARGE}
```

\idxtitlefont

Users can redefine \idxtitlefont to affect the font in which song title index entries are rendered.

```
\newcommand\idxtitlefont{\sffamily\slshape}
```

\idxlyricfont

Users can redefine \idxlyricfont to affect the font in which notable lines of lyrics are rendered in a title index.

```
\newcommand\idxlyricfont{\rmfamily}
```

\idxscripfont

Users can redefine \idxscripfont to affect the font in which scripture references are rendered in a scripture index.

```
\newcommand\idxscripfont{\sffamily\small\slshape}
```

\idxauthfont

Users can redefine \idxauthfont to affect the font in which contributor names are rendered in an author index.

```
\newcommand\idxauthfont{\small\bfseries}
```

\idxrefsfont

Users can redefine \idxrefsfont to affect the font in which the list of song references on the right-hand-side of an index entry is typeset.

```
\newcommand\idxrefsfont{\normalfont\normalsize}
```

\idxbook

Users can redefine \idxbook to dictate the book name header in a scripture index that begins each book of the bible.

```
\newcommand\idxbook[1]{\small\bfseries#1}
```

\idxcont

Users can redefine \idxcont to dictate the column header in a scripture index after a column break falls within a book of the bible.

```
\newcommand\idxcont[1]{\small\textbf{#1} (continued)}
```

\colbotglue

Glue of size \colbotglue is inserted at the bottom of each column. We use a macro instead of a glue register so that this can be redefined in terms of variable quantities such as \textheight.

```
\newcommand\colbotglue{}
\let\colbotglue\z@skip
```

`\lastcolglue` Glue of size `\lastcolglue` is inserted at the bottom of the last column.

```
\newcommand\lastcolglue{}
\let\lastcolglue\@flushglue
```

`\minfrets` Define the minimum number of fret rows that should appear in tablature diagrams.

```
\newcount\minfrets\minfrets4
```

`\SB@colwidth` Define a length to store the computed width of each column in a multi-column song page. The user shouldn't set this one directly, but some users might want to refer to it in calculations.

```
\newdimen\SB@colwidth
```

16.3 Package Options

This section defines code associated with the various option settings that can be specified on the `\usepackage` line. Many of these options can also be turned on or off subsequent to the `\usepackage` line, so macros for doing that are also located here. The options are not actually processed until §16.17 because some of the macros defined here refer to macros that have not yet been defined.

`slides` `\slides` *(Default: off)* Turning this option on generates a book of overhead slides—one for each song. It really just amounts to changing various parameter settings. Elsewhere in the code we also consult `\ifslides` to determine a few default parameter settings and to use a different song preamble structure. All the parameter changes below are local to the current scope; so to undo slides mode, just put `\slides` within a group and end the group wherever you want the slides settings to end.

```
\DeclareOption{slides}{\slides}
\newcommand\slides{%
  \slidestrue%
  \def\lyricfont{\normalfont\huge}%
  \def\chorusfont{\slshape}%
  \def\versejustify{\justifycenter}%
  \let\chorusjustify\versejustify
  \def\placenote##1{\justifycenter\noindent##1\par}%
  \scriptureoff%
  \onesongcolumn%
  \ifSB@preamble\ifSB@chordedspec\else\SB@chordsoff\fi\fi%
  \spenalty-\@M%
  \let\colbotglue\@flushglue%
  \setlength\cbarwidth\z@%
  \setlength\sbarheight\z@%
}
```

`\justifyleft` The `\justifyleft` macro sets up an environment in which lyrics are left-justified with hanging indentation equal to `\parindent`. It reserves spaces for verse numbers if used in a verse, and reserves space for the vertical bar left of choruses if used in a chorus.

```
\newcommand\justifyleft{%
  \leftskip\parindent%
  \ifSB@inverse\advance\leftskip\versenumwidth\fi%
  \SB@cbarshift%
  \parindent-\parindent%
}
```

`\justifycenter` The `\justifycenter` macro sets up an environment in which lyrics are centered on each line. Verse numbers continue to be placed flush-left, but `\placeversenum` is temporarily redefined to keep the rest of the line containing a verse number centered.

```
\newcommand\justifycenter{%
  \centering\SB@cbarshift\rightskip\leftskip%
  \def\placeversenum##1{%
    \hskip-\leftskip\hskip-\parindent\relax%
    \hangindent-\wd##1\hangafter\m@ne%
    \box##1\hfil%
  }%
}
```

`unouter` `\SB@outer` *(Default: off)* Several macros provided by the songs package are, by default, declared `\outer` to aid in debugging. However, unusual documents may need to use these macros within larger constructs. To do so, use the `unouter` option to prevent any of the macros supplied by this package from being declared `\outer`.

```
\newcommand\SB@outer{\outer}
\DeclareOption{unouter}{\let\SB@outer\relax}
```

`rawtext` *(Default: off)* Instead of generating a document, this dumps a text version of the song book to a file. This option can only be set in the `\usepackage` line because it dictates many top-level macro definitions. Turning rawtext on turns off the indexes by default, but this can be overridden by explicitly setting index options. (Note: Using rawtext with indexes turned on doesn't actually work yet, but might be added in a future revision.)

```
\DeclareOption{rawtext}{\rawtexttrue\indexesoff}
```

`nopdfindex` *(Default: off)* Inhibit the creation of the bookmark index in pdf files. This option can only be set in the `\usepackage` line because initializing the pdfbookmark library at all causes a (possibly empty) bookmark index to be created.

```
\DeclareOption{nopdfindex}{\pdfindexfalse}
```

`noshading` *(Default: off)* Inhibit all shaded boxes (e.g., if the color package is unavailable). This option can only be set in the `\usepackage` line because the color package must be loaded in the preamble if at all. (Note: In a future release this might be extended to be modifiable throughout the preamble.)

```
\DeclareOption{noshading}{\SB@colorboxesfalse}
```

noindexes
\indexeson
\indexesoff

(Default: off) Suppress generation of index files and displaying of in-document indexes. This option can only be set in the `\usepackage` line or in the preamble, since index files are opened for writing at the end of the preamble. The `\indexeson` and `\indexesoff` macros can be used elsewhere to toggle display of indexes, though generation will occur if indexes are turned on by the end of the preamble.

```
\DeclareOption{noindexes}{\indexesoff}
\newcommand\indexeson{%
  \ifSB@preamble%
    \SB@genindexestrue%
  \else%
    \ifSB@genindexes\else\SB@warnigen\fi%
  \fi%
  \songindexestrue%
}
\newcommand\indexesoff{%
  \ifSB@preamble\SB@genindexesfalse\fi%
  \songindexesfalse%
}
```

\ifSB@measurespec
\ifSB@chordedspec

The `showmeasures` and `chorded` options interact in the sense that by default, switching one of them on or off switches the other on or off as well. However, if the user explicitly says that one should be on or off, then switching the other shouldn't affect it. To produce this behavior, we need two extra conditionals to remember if each of these options has been explicitly specified by the user or if it is still in a default state.

```
\newif\ifSB@measurespec
\newif\ifSB@chordedspec
```

chorded
lyric
\chordson
\chordsoff
\SB@chordson
\SB@chordsoff

(Default: chorded) Determines whether chords should be shown. This option can be set in the `\usepackage` line or toggled elsewhere with the `\chordson` and `\chordsoff` macros. Chords cannot be turned on in conjunction with the `rawtext` option. If chords are turned on by the end of the preamble, no attempt will be made to balance columns on each page.

```
\DeclareOption{chorded}{\chordson}
\DeclareOption{lyric}{\chordsoff}
\newcommand\chordson{\SB@chordedspectrue\SB@chordson}
\newcommand\chordsoff{\SB@chordedspectrue\SB@chordsoff}
\newcommand\SB@chordson{%
  \ifrawtext%
    \SB@errrtopt%
  \else%
    \chordedtrue\lyricfalse%
    \let\SB@bracket\SB@chord%
    \let\SB@rechord\SB@@rechord%
    \let\SB@ch\SB@ch@on%
    \ifSB@measurespec%
      \ifmeasures\SB@measureson\else\SB@measuresoff\fi%
```

```
    \else%
      \SB@measureson%
    \fi%
    \ifSB@preamble\def\colbotglue{\z@\@plus.5\textheight}\fi%
    \SB@setbaselineskip%
  \fi%
}
\newcommand\SB@chordsoff{%
  \chordedfalse\lyrictrue%
  \def\SB@bracket##1]{\ignorespaces}%
  \let\SB@rechord\relax%
  \let\SB@ch\SB@ch@off%
  \ifSB@measurespec%
    \ifmeasures\SB@measureson\else\SB@measuresoff\fi%
  \else%
    \SB@measuresoff%
  \fi%
  \ifSB@preamble\let\colbotglue\z@skip\fi%
  \SB@setbaselineskip%
}
```

showmeasures
nomeasures
\measureson
\measuresoff
\SB@measureson
\SB@measuresoff

(Default: showmeasures if chorded, nomeasures otherwise) Determines whether measure bars and meter notes should be shown. Option can be set in the `\usepackage` line or toggled elsewhere with the `\measureson` and `\measuresoff` macros.

```
\DeclareOption{showmeasures}{\measureson}
\DeclareOption{nomeasures}{\measuresoff}
\newcommand\measureson{\SB@measurespectrue\SB@measureson}
\newcommand\measuresoff{\SB@measurespectrue\SB@measuresoff}
\newcommand\SB@measureson{%
  \measurestrue%
  \let\SB@mbar\SB@makembar%
  \ifchorded%
    \let\SB@mch\SB@mch@on%
  \else%
    \let\SB@mch\SB@mch@m%
  \fi%
  \ifSB@inverse\SB@loadactives\fi%
  \ifSB@inchorus\SB@loadactives\fi%
}
\newcommand\SB@measuresoff{%
  \measuresfalse%
  \let\SB@mbar\@gobbletwo%
  \ifchorded%
    \let\SB@mch\SB@ch@on%
  \else%
    \let\SB@mch\SB@ch@off%
  \fi%
  \ifSB@inverse\SB@loadactives\fi%
  \ifSB@inchorus\SB@loadactives\fi%
```

```
}
```

`transposecapos` *(Default: off)* If set, the `\capo` macro transposes the song instead of printing a note to use a capo. Use this option to generate a chord book for pianists who have trouble transposing or guitarists who don't have capos.

```
\DeclareOption{transposecapos}{\transcapostrue}
```

`noscripture` `\scriptureon` `\scriptureoff` *(Default: off)* Inhibits the display of scripture quotes. This option can also be toggled on and off anywhere with the `\sciptureon` and `\scriptureoff` macros.

```
\DeclareOption{noscripture}{\SB@omitscriptrue}
\newcommand\scriptureon{\SB@omitscripfalse}
\newcommand\scriptureoff{\SB@omitscriptrue}
```

`onesongcolumn` `twosongcolumns` `\onesongcolumn` `\twosongcolumns` `\songcolumns` *(Default: onesongcolumn is the default if generating slides or rawtext, twosongcolumns otherwise)* The number of columns per page is specified using the following package options and macros. In `rawtext` mode it must remain set to one column per page. The entire page-making system can be turned off by setting the number of columns to zero. This will cause each song to be contributed to the current vertical list without any attempt to form columns; the enclosing environment must handle the page layout. Probably this means that `\repchoruses` will not work, since an external package won't know to insert repeated choruses when building pages.

```
\DeclareOption{twosongcolumns}{\SB@numcols\tw@}
\DeclareOption{onesongcolumn}{\SB@numcols\@ne}
\newcommand\songcolumns[1]{%
  \SB@cnt#1\relax%
  \ifnum\SB@cnt=\SB@numcols\else%
    \ifSB@preamble\else{\SB@clearpage}\fi%
  \fi%
  \SB@numcols\SB@cnt%
  \ifnum\SB@numcols>\z@%
    \SB@colwidth-\columnsep%
    \multiply\SB@colwidth\SB@numcols%
    \advance\SB@colwidth\columnsep%
    \advance\SB@colwidth\textwidth%
    \divide\SB@colwidth\SB@numcols%
  \else%
    \ifrepchorus\SB@warnrc\fi%
  \fi%
}
\newcommand\onesongcolumn{\songcolumns\@ne}
\newcommand\twosongcolumns{\songcolumns\tw@}
```

`\includeonlysongs` `\songlist` Display only a select list of songs and ignore the rest.

```
\newcommand\songlist{}
\newcommand\includeonlysongs[1]{%
  \ifSB@songsenv\SB@errpl\else%
    \partiallisttrue%
```

```
    \renewcommand\songlist{#1}%
  \fi%
}
```

`\nosongnumbers` The user can turn off song numbering with the following macro.

```
\newcommand\nosongnumbers{\setlength\songnumwidth\z@}
```

`\noversenumbers` The user can turn off verse numbering with the following macro.

```
\newcommand\noversenumbers{%
  \renewcommand\printversenum[1]{}%
  \setlength\versenumwidth\z@%
}
```

`\repchoruses` `\norepchoruses` Using `\repchoruses` causes choruses to be automatically repeated on subsequent pages of the song. The feature requires ε-TEX because the supporting code needs an extended mark register class.

```
\ifSB@etex
  \newcommand\repchoruses{%
    \ifnum\SB@numcols<\@ne\SB@warnrc\fi%
    \repchorustrue%
  }
\else
  \newcommand\repchoruses{\SB@erretex}
\fi
\newcommand\norepchoruses{\repchorusfalse}
```

`\sepverses` The following penalty settings cause verses and choruses to be separated onto different slides when in slides mode, except that consecutive choruses remain together when they fit.

```
\newcommand\sepverses{%
  \vvpenalty-\@M%
  \ccpenalty100 %
  \vcpenalty\vvpenalty%
  \cvpenalty\vvpenalty%
  \let\colbotglue\@flushglue%
}
```

Some option settings, margins, and other lengths are finalized at the end of the preamble. That code is below.

```
\AtBeginDocument{
```

If the user hasn't set the `\versesep`, set it to the default.

```
  \SB@setversesep
```

Initialize page layout algorithm.

```
  \songcolumns\SB@numcols
```

Macros used after this point occur outside the preamble.

```
  \SB@preamblefalse
}
```

16.4 Page-builder

The following macros handle the building of pages that contain songs. They compute where best to place each song (e.g., whether to place it in the current column or move to the next column or page). The output routines for generating a partial list of songs in a specified order also can be found here.

`\SB@songbox` The most recently processed song (or scripture quotation) is stored in this box.

```
\newbox\SB@songbox
```

`\SB@numcols` `\SB@colnum` Reserve two count registers to hold the total number of columns and the current column number, respectively.

```
\newcount\SB@numcols\SB@numcols\tw@
\newcount\SB@colnum
```

`\SB@colbox` Reserve a box register to hold the current column in progress.

```
\newbox\SB@colbox
```

`\SB@colbox` Reserve a box register to hold the current page in progress.

```
\newbox\SB@pgbox
```

`\SB@mrkbox` Reserve a box register to hold marks that migrate out of songs as they get split into columns and pages.

```
\newbox\SB@mrkbox
```

`\SB@maxmin` The following helper macro takes the max or min of two dimensions. If ⟨*arg2*⟩="`<`", it sets ⟨*arg1*⟩ to the maximum of ⟨*arg1*⟩ and ⟨*arg3*⟩. If ⟨*arg2*⟩="`>`", it sets ⟨*arg1*⟩ to the minimum of ⟨*arg1*⟩ and ⟨*arg3*⟩.

```
\newcommand\SB@maxmin[3]{\ifdim#1#2#3#1#3\fi}
```

`\SB@mkpage` The following macro is the heart of the page-building engine. It splits the contents of a box into a page of columns. If `\repchoruses` is active, the contents of `\SB@chorusbox` are additionally inserted into fresh columns created during the spitting process. The macro arguments are:

1. the box b to split (must not be `\SB@box`, which is used as a temp register),
2. a count register i equal to the first column index (zero or greater), and
3. the desired column height.

Box b is split and i is incremented until i reaches `\SB@numcols` or b is emptied, whichever occurs first. If b is emptied first, the final column is *not* contributed; instead it is left in b and i is left equal to the index of the column that would have been added if b had been emptied. This allows the next call to reconsider whether to end the current column here or add some or all of the next contribution to it. Box b and count register i are globally modified. If `\SB@updatepage` is not redefined, boxes `\SB@pgbox` and `\SB@mrkbox` are also globally modified based on the results of the split.

```
\newcommand\SB@mkpage[3]{%
  \begingroup%
    \splitmaxdepth\maxdepth\splittopskip\z@skip%
    \global\setbox#1\vbox{%
      \unvbox#1%
      \nointerlineskip%
      \null%
      \vfil%
    }%
    \loop\ifnum#2<\SB@numcols%
      \setbox\SB@box\vsplit#1to#3\relax%
      \ifvoid#1%
        #2\SB@numcols%
      \else%
        \SB@updatepage%
        \global\advance#2\@ne%
        \ifrepchorus\ifvoid\SB@chorusbox\else%
          \SB@insertchorus#1%
        \fi\fi%
      \fi%
    \repeat%
    \global\setbox#1\vbox{%
      \unvbox\SB@box%
      \unvbox#1%
      \unskip%
      \setbox\SB@box\lastbox%
    }%
  \endgroup%
}
```

`\SB@updatepage` Update boxes `\SB@pgbox` and `\SB@mrkbox` immediately after splitting the contents of `\SB@colbox`.

```
\newcommand\SB@updatepage{%
  \global\setbox\SB@mrkbox\vbox{%
    \unvbox\SB@mrkbox%
    \edef\SB@temp{\splitfirstmark}%
    \ifx\SB@temp\@empty\else\mark{\splitfirstmark}\fi%
    \edef\SB@temp{\splitbotmark}%
    \ifx\SB@temp\@empty\else\mark{\splitbotmark}\fi%
  }%
  \global\setbox\SB@pgbox\hbox{%
```

```
    \SB@dimen\SB@colwidth%
    \advance\SB@dimen\columnsep%
    \multiply\SB@dimen\SB@colnum%
    \advance\SB@dimen-\wd\SB@pgbox%
    \unhbox\SB@pgbox%
    \ifdim\SB@dimen=\z@\else\hskip\SB@dimen\relax\fi%
    \box\SB@box%
  }%
}
```

`\SB@droppage` This alternate definition of `\SB@updatepage` drops the just-created page instead of contributing it. This allows `\SB@mkpage` to be called by the song-positioning algorithm as a trial run without outputting anything.

```
\newcommand\SB@droppage{\setbox\SB@box\box\voidb@x}
```

`\SB@output` Split the contents of the current column box into separate columns, emitting pages as they are completed. Leave the final in-progress column unfinished, pending future contributions.

```
\newcommand\SB@output{%
  \ifnum\SB@numcols>\z@\begingroup%
    \loop%
      \SB@dimen\textheight%
      \ifinner\else\advance\SB@dimen-\pagetotal\fi%
      \SB@mkpage\SB@colbox\SB@colnum\SB@dimen%
      \ifnum\SB@colnum<\SB@numcols\else%
        \unvbox\SB@mrkbox%
        \ifinner\else\kern\z@\fi%
        \box\SB@pgbox%
        \ifinner\else\vfil\break\vskip\vsize\relax\fi%
        \global\SB@colnum\z@%
    \repeat%
  \endgroup\else%
    \unvbox\SB@colbox\unskip%
  \fi%
}
```

`\SB@putboxes` Create a vertical list consisting of the already committed contents of the current column plus the most recently submitted song box. The LaTeX primitive that should be used to contribute each box is specified in the first argument.

```
\newcommand\SB@putboxes[1]{%
  \SB@dimen\ifnum\SB@numcols>\z@\ht\SB@colbox\else\p@\fi%
  #1\SB@colbox%
  \ifdim\SB@dimen>\z@%
    \SB@breakpoint\spenalty%
    \ifdim\sbarheight>\z@%
      \vskip-\sbarheight\relax%
    \fi%
  \fi%
  #1\SB@songbox%
}
```

`\SB@nextcol` Force n column breaks, where n is given by the first argument. The first created column is finished with the glue specified in the second argument. When the second argument is `\@flushglue`, this forces a break that leaves whitespace at the bottom of the column. When it's `\colbotglue`, it acts like a natural column break chosen by the page-breaker. However, if the current column is empty, `\@flushglue` is always used so that an empty column will result.

```
\newcommand\SB@nextcol[2]{%
  \ifnum#1>\z@%
    \ifnum\SB@numcols>\z@%
      \global\setbox\SB@colbox\vbox{%
        \SB@cnt#1\relax%
        \SB@dimen\ht\SB@colbox%
        \unvbox\SB@colbox%
        \unskip%
        \ifdim\SB@dimen>\z@%
          \vskip#2\relax%
          \break%
          \advance\SB@cnt\m@ne%
        \fi%
        \loop\ifnum\SB@cnt>\z@%
          \nointerlineskip%
          \null%
          \vfil%
          \break%
          \advance\SB@cnt\m@ne%
        \repeat%
      }%
      \SB@output%
    \else%
      \ifnum\lastpenalty=-\@M\null\fi%
      \break%
    \fi%
  \fi%
}
```

`\SB@selectcol` This is the entrypoint to the song-positioning algorithm. It gets defined by `\songpos` to either `\SB@@selectcol` (below) or `\relax` (when song-positioning is turned off).

```
\newcommand\SB@selectcol{}
```

`\SB@@selectcol` Songs should be squeezed in wherever they fit, but breaking a column or page within a song should be avoided. The following macro outputs zero or more column breaks to select a good place for `\SB@songbox` to be contributed to the current (or the next) page.

```
\newcommand\SB@@selectcol{%
  \begingroup%
    \SB@cnt\z@%
    \vbadness\@M\vfuzz\maxdimen%
```

```
    \let\SB@updatepage\SB@droppage%
    \SB@dimen\textheight%
    \ifinner\else\advance\SB@dimen-\pagetotal\fi%
    \setbox\SB@boxii\vbox{\SB@putboxes\unvcopy}%
    \SB@cntii\SB@colnum%
    \SB@mkpage\SB@boxii\SB@cntii\SB@dimen%
    \SB@spos%
    \global\SB@cnt\SB@cnt%
  \endgroup%
  \SB@nextcol\SB@cnt\colbotglue%
}
```

`\SB@spbegnew` Begin a trial typesetting of the current song on a fresh page to see if it fits within a page.

```
\newcommand\SB@spbegnew{%
  \setbox\SB@boxiii\copy\SB@songbox%
  \SB@cntii\z@%
  \SB@mkpage\SB@boxiii\SB@cntii\textheight%
}
```

`\SB@spextold` Tentatively extend the song previously typeset on the current even page to the next odd page to see if it fits on a double-page. If the current page is odd-numbered, do nothing since extending the song to the next page would introduce a page-turn.

```
\newcommand\SB@spextold{%
  \ifodd\c@page\else%
    \SB@cntii\z@%
    \SB@mkpage\SB@boxii\SB@cntii\textheight%
  \fi%
}
```

`\SB@spextnew` Extend the trial typesetting started with `\SB@spbegnew` to a second page to see if the song fits on a fresh double-page.

```
\newcommand\SB@spextnew{%
  \SB@cntii\z@%
  \SB@mkpage\SB@boxiii\SB@cntii\textheight%
}
```

`\SB@spdblpg` Compute the number of column breaks required to shift the current song to the next double-page if the result of the last test run fits within its page (as indicated by counter `\SB@cntii`). Otherwise leave the requested number of column breaks set to zero.

```
\newcommand\SB@spdblpg{%
  \ifnum\SB@cntii<\SB@numcols%
    \SB@cnt\SB@numcols%
    \advance\SB@cnt-\SB@colnum%
    \if@twoside\ifodd\c@page\else%
      \advance\SB@cnt\SB@numcols%
    \fi\fi%
  \fi%
}
```

\SB@sposi This is the level-1 song positioning algorithm. It moves songs to the next double-page only if doing so would avoid a page-turn that would otherwise appear within the song.

```
\newcommand\SB@sposi{%
  \ifnum\SB@cntii<\SB@numcols\else\if@twoside%
    \SB@spextold%
  \fi\fi%
  \ifnum\SB@cntii<\SB@numcols\else%
    \SB@spbegnew%
    \ifnum\SB@cntii<\SB@numcols\else\if@twoside%
      \SB@spextnew%
    \fi\fi%
    \SB@spdblpg%
  \fi%
}
```

\SB@sposii This is the level-2 song-positioning algorithm. It moves songs to the next page or double-page if doing so avoids a page-break or page-turn that would otherwise appear within the song.

```
\newcommand\SB@sposii{%
  \ifnum\SB@cntii<\SB@numcols\else%
    \SB@spbegnew%
    \ifnum\SB@cntii<\SB@numcols%
      \SB@cnt\SB@numcols%
      \advance\SB@cnt-\SB@colnum%
    \else%
      \if@twoside%
        \SB@spextold%
        \ifnum\SB@cntii<\SB@numcols\else%
          \SB@spextnew%
          \SB@spdblpg%
        \fi%
      \fi%
    \fi%
  \fi%
}
```

\SB@sposiii This is the level-3 song-positioning algorithm. It moves songs to the next column, the next page, or the next double-page if doing so avoids a column-break, page-break, or page-turn that would otherwise appear within the song.

```
\newcommand\SB@sposiii{%
  \ifnum\SB@cntii>\SB@colnum%
    \SB@cnt\SB@colnum%
    \advance\SB@cnt\@ne%
    \ifnum\SB@cnt<\SB@numcols%
      \setbox\SB@boxiii\copy\SB@songbox%
      \SB@mkpage\SB@boxiii\SB@cnt\SB@dimen%
      \advance\SB@cnt\m@ne%
    \fi%
```

```
    \ifnum\SB@cnt>\SB@colnum%
      \SB@cnt\z@%
      \SB@sposii%
    \else%
      \SB@cnt\@ne%
    \fi%
  \fi%
}
```

`\songpos` This is the macro by which the user adjusts the aggressiveness level of the song-positioning algorithm. See the macros above for what each level does.

```
\newcommand\songpos[1]{%
  \ifcase#1%
    \let\SB@selectcol\relax%
    \let\SB@spos\relax%
  \or%
    \let\SB@selectcol\SB@@selectcol%
    \let\SB@spos\SB@sposi%
  \or%
    \let\SB@selectcol\SB@@selectcol%
    \let\SB@spos\SB@sposii%
  \or%
    \let\SB@selectcol\SB@@selectcol%
    \let\SB@spos\SB@sposiii%
  \else%
    \SB@errspos%
  \fi%
}
```

`\SB@spos` The `\SB@spos` macro gets redefined by `\songpos` above depending on the current song-positioning aggressiveness level. By default it is set to level 3.

```
\newcommand\SB@spos{}
\songpos\thr@@
```

`\SB@clearpage` Output all contributed material as a new page unless there is no contributed material. In that case do nothing (i.e., don't produce a blank page).

```
\newcommand\SB@clearpage{%
  \SB@testtrue%
  \ifvoid\SB@pgbox\ifvoid\SB@colbox\SB@testfalse\fi\fi%
  \ifSB@test%
    \SB@cnt\SB@numcols%
    \advance\SB@cnt-\SB@colnum%
    \SB@nextcol\SB@cnt\lastcolglue%
  \fi%
}
```

`\SB@cleardpage` Like `\SB@clearpage` but shift to a fresh *even-numbered* page in two-sided documents. Note that this differs from LaTeX's `\cleardoublepage`, which shifts to

odd-numbered pages. Song books prefer starting things on even-numbered pages because this maximizes the distance until the next page-turn.

```
\newcommand\SB@cleardpage{%
  \SB@clearpage%
  \if@twoside\ifodd\c@page%
    \SB@nextcol\SB@numcols\@flushglue%
  \fi\fi%
}
```

`\SB@stype` There are two song submission types: column- and page-submissions. This macro gets set to the desired type for the current submission. Mostly it stays set to the default column-submission type.

```
\newcommand\SB@stype{\SB@stypcol}
```

`\SB@stypcol` Column-submissions contribute the contents of `\SB@songbox` to either the current column or the next column or page, depending on where it best fits.

```
\newcommand\SB@stypcol{%
  \ifnum\SB@numcols>\z@%
    \SB@selectcol%
    \global\setbox\SB@colbox\vbox{\SB@putboxes\unvbox}%
    \SB@output%
  \else%
    \unvbox\voidb@x%
    \SB@breakpoint\spenalty%
    \ifdim\sbarheight>\z@%
      \vskip-\sbarheight\relax%
    \fi%
    \unvbox\SB@songbox%
  \fi%
}
```

`\SB@styppage` Page-submissions go directly to the top of the nearest fresh page.

```
\newcommand\SB@styppage{%
  \ifnum\SB@numcols>\z@%
    \SB@clearpage%
    \unvbox\SB@songbox%
    \null\nointerlineskip%
  \else%
    \unvbox\SB@songbox%
  \fi%
}
```

`\SB@sgroup` This macro controls whether songs submitted to the page-builder are actually contributed to the final document when using `\includeonlysongs` to generate a partial list. If `\SB@sgroup` is empty, then the song is silently dropped. Otherwise it is contributed only if `\SB@sgroup` is a member of `\songlist`.

```
\newcommand\SB@sgroup{}
\let\SB@sgroup\@empty
```

`\SB@groupcnt` This counter assigns a unique integer to each item of a group. Environments that come before the group's song are numbered decreasingly from -1. The song itself has number 0. Environments that come after the song are numbered increasingly from 1.

```
\newcount\SB@groupcnt
```

`\SB@submitpart` When a song completes and we're generating a partial list, save the song in a box so that it can be submitted at the end of the section in the order specified by `\includeonlysongs`.

```
\newcommand\SB@submitpart{%
  \ifx\SB@sgroup\@empty\else%
    \@for\SB@temp:=\songlist\do{%
      \ifx\SB@temp\SB@sgroup%
        \edef\SB@tempii{\SB@sgroup @\the\SB@groupcnt}%
        \expandafter\newbox\csname songbox@\SB@tempii\endcsname%
        \global\expandafter\setbox
          \csname songbox@\SB@tempii\endcsname\box\SB@songbox%
        \global\expandafter\let%
          \csname stype@\SB@tempii\endcsname\SB@stype%
        \ifrepchorus\ifvoid\SB@chorusbox\else%
          \expandafter\newbox\csname chbox@\SB@tempii\endcsname%
          \global\expandafter\setbox%
            \csname chbox@\SB@tempii\endcsname\box\SB@chorusbox%
        \fi\fi%
      \fi%
    }%
    \global\advance\SB@groupcnt%
      \ifnum\SB@groupcnt<\z@\m@ne\else\@ne\fi%
  \fi%
  \setbox\SB@songbox\box\voidb@x%
  \setbox\SB@chorusbox\box\voidb@x%
}
```

`\SB@submitsong` Submit the most recently finished song (or block of other vertical material) for output. If we're generating a partial list of songs, save it in a box instead of submitting it here. (The saved boxes will be submitted in the requested order at the end of the songs section.)

```
\newcommand\SB@submitsong{%
  \ifpartiallist\SB@submitpart\else\SB@stype\fi%
}
```

`\SB@songlistbrk` `\SB@songlistnc` `\SB@songlistcp` `\SB@songlistcdp` These macros define the words that, when placed in a `\songlist`, force a column break at that point. Using `brk` produces a soft break (like `\brk`) that won't leave whitespace at the bottom of the broken column in lyric books. Using `nextcol` produces a hard break (like `\nextcol`) that may insert whitespace to finish the column. Using `sclearpage` moves to the next page if the current page is nonempty. Using `scleardpage` moves to the next double-page if the current double-page is nonempty.

```
\newcommand\SB@songlistbrk{}
\def\SB@songlistbrk{brk}
\newcommand\SB@songlistnc{}
\def\SB@songlistnc{nextcol}
\newcommand\SB@songlistcp{}
\def\SB@songlistcp{sclearpage}
\newcommand\SB@songlistcdp{}
\def\SB@songlistcdp{scleardpage}
```

`\SB@finloop` Contribute a sequence of boxes saved for a partial list.

```
\newcommand\SB@finloop{%
  \loop\edef\SB@tempii{\SB@temp @\the\SB@groupcnt}%
       \expandafter\ifx%
         \csname songbox@\SB@tempii\endcsname\relax\else%
    \setbox\SB@songbox\expandafter\box%
        \csname songbox@\SB@tempii\endcsname%
    \expandafter\ifx\csname chbox@\SB@tempii\endcsname\relax%
      \repchorusfalse%
    \else%
      \repchorustrue%
      \setbox\SB@chorusbox\expandafter\box%
        \csname chbox@\SB@tempii\endcsname%
    \fi%
    \csname stype@\SB@tempii\endcsname%
    \advance\SB@groupcnt\ifnum\SB@groupcnt<\z@\m@ne\else\@ne\fi%
  \repeat%
}
```

`\commitsongs` If we're generating only a partial list, then wait until the end of the section and then output all the songs we saved in boxes in the order specified.

```
\newcommand\commitsongs{%
  \ifpartiallist%
    \ifnum\SB@numcols>\z@%
      \@for\SB@temp:=\songlist\do{%
        \ifx\SB@temp\SB@songlistnc\SB@nextcol\@ne\@flushglue\else%
        \ifx\SB@temp\SB@songlistbrk\SB@nextcol\@ne\colbotglue\else%
        \ifx\SB@temp\SB@songlistcp\SB@clearpage\else%
        \ifx\SB@temp\SB@songlistcdp\SB@cleardpage\else%
          \SB@groupcnt\m@ne\SB@finloop%
          \SB@groupcnt\z@\SB@finloop%
        \fi\fi\fi\fi%
      }%
    \else%
      \@for\SB@temp:=\songlist\do{%
        \ifx\SB@temp\SB@songlistnc\vfil\break\else%
        \ifx\SB@temp\SB@songlistbrk\break\else%
        \ifx\SB@temp\SB@songlistcp\clearpage\else%
        \ifx\SB@temp\SB@songlistcdp%
          \clearpage%
          \ifodd\c@page\null\newpage\fi%
```

```
        \else%
          \SB@groupcnt\m@ne\SB@finloop%
          \SB@groupcnt\z@\SB@finloop%
        \fi\fi\fi\fi%
      }%
    \fi%
  \fi%
  \SB@clearpage%
}
```

`\SB@insertchorus` If necessary, insert a chorus into the first marked spot in box given in the first argument. This is usually achieved by splitting the box at the first valid breakpoint after the first `\SB@cmark` in the box. The box is globally modified.

```
\newcommand\SB@insertchorus[1]{{%
  \vbadness\@M\vfuzz\maxdimen%
  \setbox\SB@box\copy#1%
  \setbox\SB@box\vsplit\SB@box to\maxdimen%
  \edef\SB@temp{\splitfirstmarks\SB@nocmarkclass}%
  \ifx\SB@temp\SB@nocmark\else%
    \edef\SB@temp{\splitfirstmarks\SB@cmarkclass}%
    \ifx\SB@temp\SB@cmark%
      \SB@dimen4096\p@%
      \SB@dimenii\maxdimen%
      \SB@dimeniii\SB@dimen%
      \loop%
        \SB@dimeniii.5\SB@dimeniii%
        \setbox\SB@box\copy#1%
        \setbox\SB@box\vsplit\SB@box to\SB@dimen%
        \edef\SB@temp{\splitfirstmarks\SB@cmarkclass}%
        \ifx\SB@temp\SB@cmark%
          \SB@dimenii\SB@dimen%
          \advance\SB@dimen-\SB@dimeniii%
        \else%
          \advance\SB@dimen\SB@dimeniii%
        \fi%
      \ifdim\SB@dimeniii>2\p@\repeat%
      \setbox\SB@box\vsplit#1to\SB@dimenii%
      \global\setbox#1\vbox{%
        \unvbox\SB@box\unskip%
        \SB@inversefalse\SB@prevversetrue\SB@stanzabreak%
        \SB@putbox\unvcopy\SB@chorusbox%
        \SB@inversetrue\SB@prevversefalse\SB@stanzabreak%
        \unvbox#1%
      }%
```

However, if the first mark is a `\SB@lastcmark`, it means that this chorus should go after the last verse in the song. There is no valid breakpoint there, so to get a chorus into that spot, we have to do a rather ugly hack: We pull the bottom material off the box with `\unskip`, `\unpenalty`, and `\lastbox`, then insert the chorus, then put the bottom material back on. This works because the high-

level structure of the bottom material should be static. Even if the user redefines `\makepostlude`, the new definition gets put in a single box that can be manipulated with `\lastbox`. However, if we ever change the high-level structure, we need to remember to change this code accordingly.

```
    \else\ifx\SB@temp\SB@lastcmark%
      \global\setbox#1\vbox{%
        \unvbox#1%
        \unskip%
        \ifdim\sbarheight>\z@%
          \setbox\SB@box\lastbox%
          \unskip\unpenalty%
        \fi%
        \setbox\SB@box\lastbox%
        \unskip\unskip%
        \SB@inversefalse\SB@prevversetrue\SB@stanzabreak%
        \marks\SB@nocmarkclass{\SB@nocmark}%
        \unvcopy\SB@chorusbox%
        \vskip\versesep\vskip2\p@\@plus4\p@%
        \nointerlineskip\box\SB@box%
        \ifdim\sbarheight>\z@%
          \nobreak\vskip2\p@\@plus\p@%
          \hrule\@height\sbarheight\@width\SB@colwidth%
        \fi%
      }%
    \fi\fi%
  \fi%
}}
```

`\nextcol` End the current column (inserting vertical space as needed) unless we're generating a partial list. (Partial lists should be broken via the `nextcol` argument to the `\includeonlysongs` macro.) This differs from column breaks produced with `\brk`, which does not introduce any empty vertical space.

```
\newcommand\nextcol{%
  \@ifstar{\SB@nextcol\@ne\@flushglue}%
          {\ifpartiallist\else\SB@nextcol\@ne\@flushglue\fi}%
}
```

`\sclearpage` Move to the next page if the current page is nonempty.

```
\newcommand\sclearpage{%
  \@ifstar\SB@clearpage{\ifpartiallist\else\SB@clearpage\fi}%
}
```

`\scleardpage` Move to the next even-numbered page if the current page is odd or nonempty.

```
\newcommand\scleardpage{%
  \@ifstar\SB@cleardpage{\ifpartiallist\else\SB@cleardpage\fi}%
}
```

16.5 Songs

The following macros handle the parsing and formatting of the material that begins and ends each song.

\SB@lop
\SB@@lop

The following macros were adapted from Donald Knuth's *The TEXbook*, for manipulating lists of the form *item1**item2*\\...*itemN*\\.

\SB@emptylist
\SB@ifempty

```
\newcommand\SB@lop[1]{\expandafter\SB@@lop\the#1\SB@@lop#1}
\newcommand\SB@@lop{}
\def\SB@@lop\\#1\\#2\SB@@lop#3#4{\global#3{\\#2}\global#4{#1}}
\newcommand\SB@emptylist{}
\def\SB@emptylist{\\}
\newcommand\SB@ifempty[3]{%
  \edef\SB@temp{\the#1}%
  \ifx\SB@temp\SB@emptylist#2\else#3\fi%
}
```

\SB@titlelist
\SB@titletail

These registers hold the full list of titles for the current song and the tail list of titles that has not yet been iterated over.

```
\newtoks\SB@titlelist
\newtoks\SB@titletail
```

\songtitle

The \songtitle macro will initially hold the primary title of the current song. The user can iterate over titles using \nexttitle or \foreachtitle.

```
\newcommand\songtitle{}
```

\resettitles

Initialize the title list iterator.

```
\newcommand\resettitles{%
  \global\SB@titletail\SB@titlelist%
  \nexttitle%
}
```

\nexttitle

Advance the title list iterator to the next title.

```
\newcommand\nexttitle{%
  \SB@ifempty\SB@titletail{%
    \global\let\songtitle\relax%
  }{%
    \SB@lop\SB@titletail\SB@toks%
    \edef\songtitle{\the\SB@toks}%
  }%
}
```

\foreachtitle

Execute a block of code for each remaining title in the title list.

```
\newcommand\foreachtitle[1]{%
  \ifx\songtitle\relax\else%
    \loop#1\nexttitle\ifx\songtitle\relax\else\repeat%
  \fi%
}
```

\ifSB@insong
\ifSB@intersong
\ifSB@inverse
\ifSB@inchorus

To help the user locate errors, keep track of which environments we're inside and immediately signal an error if someone tries to use a song command inside a scripture quotation, etc.

```
\newif\ifSB@songsenv\SB@songsenvfalse
\newif\ifSB@insong\SB@insongfalse
\newif\ifSB@intersong\SB@intersongfalse
\newif\ifSB@inverse\SB@inversefalse
\newif\ifSB@inchorus\SB@inchorusfalse
```

\SB@closeall

If an error is detected using one of the above, the following macro will contain a macro sequence sufficient to end the unclosed environment, hopefully allowing processing to continue.

```
\newcommand\SB@closeall{}
```

\SB@rawrefs
\songauthors
\songcopyright
\songlicense

The current song's scripture references, authors, copyright info, and copyright license information are stored in these macros.

```
\newcommand\SB@rawrefs{}
\newcommand\songauthors{}
\newcommand\songcopyright{}
\newcommand\songlicense{}
```

\songrefs

When the user asks for the song's scripture references, rather than give them the raw token list that the author entered, we return a prettier version in which spaces, dashes, and penalties have been adjusted. The prettier version is stored in the following control sequence.

```
\newcommand\songrefs{}
```

\setlicense

The user sets the licensing info for the current song with this command.

```
\newcommand\setlicense{\gdef\songlicense}
```

\newsongkey
\SB@clearbskeys

Defining a new key for `\beginsong` is just like the `keyval` package's `\define@key` macro except that we must also define some initializer code for each key. This provides an opportunity to clear registers before each song. (Otherwise when a key wasn't specified, we'd inherit the old values from the previous song.)

```
\newcommand\SB@clearbskeys{}
\newcommand\newsongkey[2]{%
  \expandafter\gdef\expandafter\SB@clearbskeys\expandafter%
    {\SB@clearbskeys#2}%
  \define@key{beginsong}{#1}%
}
```

Define keys `sr`, `by`, `cr`, `li`, `index`, and `ititle` for scripture references, authors, copyright info, licensing info, lyric index entries, and alternate title index entries, respectively.

```
\newsongkey{sr}{\def\SB@rawrefs{}\gdef\songrefs{}}
                {\def\SB@rawrefs{#1}\SB@parsesrefs{#1}}
\newsongkey{by}{\def\songauthors{}}{\def\songauthors{#1}}
```

```
\newsongkey{cr}{\def\songcopyright{}}{\def\songcopyright{#1}}
\newsongkey{li}{\setlicense{}}{\setlicense{#1}}
\newsongkey{index}{}{\indexentry{#1}}
\newsongkey{ititle}{}{\indextitleentry{#1}}
```

song
\beginsong
\SB@@beginsong
\SB@bsoldfmt
\SB@@bskvfmt

Parse the arguments of a \beginsong macro. The \beginsong macro supports two syntaxes. The preferred syntax takes the song title(s) as its first argument and an optional keyval list in brackets as its second argument. A legacy syntax supports four arguments, all enclosed in braces, which are: the title(s), scripture references, authors, and copyright info.

```
\newenvironment{song}{\beginsong}{\SB@endsong}
\newcommand\beginsong[1]{%
  \ifSB@insong\SB@errboo\SB@closeall\fi%
  \ifSB@intersong\SB@errbor\SB@closeall\fi%
  \SB@insongtrue%
  \def\SB@closeall{\endsong}%
  \SB@parsetitles{#1}%
  \global\setbox\SB@songwrites\box\voidb@x%
  \SB@clearbskeys%
  \@ifnextchar[\SB@bskvfmt\SB@@beginsong%
}
\newcommand\SB@@beginsong{%
  \@ifnextchar\bgroup\SB@bsoldfmt\SB@@@beginsong%
}
\newcommand\SB@bsoldfmt[3]{%
  \SB@bskvfmt[sr={#1},by={#2},cr={#3}]%
}
\newcommand\SB@bskvfmt{}
\def\SB@bskvfmt[#1]{%
  \setkeys{beginsong}{#1}%
  \SB@@@beginsong%
}
```

\SB@@@beginsong

Begin typesetting a song. Beginning a song involves typesetting the title and other info, adding entries to the indexes, and setting up the environment in which verses and choruses lie.

```
\newcommand\SB@@@beginsong{%
  \global\SB@stanzafalse%
  \setbox\SB@chorusbox\box\voidb@x%
  \SB@gotchorusfalse%
  \setbox\SB@songbox\vbox\bgroup\begingroup%
    \ifnum\SB@numcols>\z@\hsize\SB@colwidth\fi%
    \leftskip\z@skip\rightskip\z@skip%
    \parfillskip\@flushglue\parskip\z@skip%
    \SB@raggedright%
    \global\SB@transposefactor\z@%
    \global\SB@cr@{\\}%
    \protected@edef\@currentlabel{\p@songnum\thesongnum}%
    \setcounter{versenum}{1}%
```

```
    \SB@prevversetrue%
    \meter44%
    \resettitles%
    \SB@addtoindexes\songtitle\SB@rawrefs\songauthors%
    \nexttitle%
    \foreachtitle{\expandafter\SB@addtotitles\expandafter{\songtitle}}%
    \resettitles%
    \lyricfont%
    \SB@setbaselineskip%
}
```

`\SB@endsong` Ending a song involves creating the song header (with `\makeprelude`), creating the song footer (with `\makepostlude`), and then assembling everything together into the `\SB@songbox`. The box is then submitted to the page-builder via `\SB@submitsong`. We do things this way instead of just contributing material directly to the main vertical list because submitting material song by song allows for a more sophisticated page-breaking algorithm than is possible with TEX's built-in algorithm.

```
\newcommand\SB@endsong{%
  \ifSB@insong%
      \ifSB@inverse\SB@erreov\endverse\fi%
      \ifSB@inchorus\SB@erreoc\endchorus\fi%
      \global\SB@skip\versesep%
      \unskip%
      \ifrepchorus\ifvoid\SB@chorusbox\else%
        \ifSB@prevverse\ifvnumbered%
          \marks\SB@cmarkclass{\SB@lastcmark}%
        \fi\fi%
      \fi\fi%
    \endgroup\egroup%
    \setbox\SB@songbox\vbox{%
      \songmark%
      \unvbox\SB@songwrites%
      \ifnum\SB@numcols>\z@\hsize\SB@colwidth\fi%
      \leftskip\z@skip\rightskip\z@skip%
      \parfillskip\@flushglue\parskip\z@skip\parindent\z@%
      \ifdim\sbarheight>\z@%
        \hrule\@height\sbarheight\@width\hsize%
        \nobreak\vskip5\p@%
      \fi%
      \ifpdfindex\begingroup%
        \ifx\pdfbookmark\undefined\else%
        \ifx\pdfbookmark\relax\else%
          \resettitles%
          \pdfbookmark[\ifnum\c@section=\z@1\else2\fi]%
            {\thesongnum. \songtitle}%
            {song\theSB@songsnum-\thesongnum}%
        \fi\fi%
      \endgroup\fi%
```

```
      \vbox{\makeprelude}%
      \nobreak\vskip\SB@skip%
      \vskip2\p@\@plus4\p@%
      \unvbox\SB@songbox%
      \nobreak\vskip\SB@skip%
      \vskip2\p@\@plus4\p@%
      \nointerlineskip%
      \vbox{\makepostlude}%
      \ifdim\sbarheight>\z@%
        \nobreak\vskip2\p@\@plus\p@%
        \nointerlineskip%
        \hbox{\vrule\@height\sbarheight\@width\hsize}%
      \fi%
    }%
    \SB@insongfalse%
    \edef\SB@sgroup{\thesongnum}%
    \global\SB@groupcnt\z@%
    \SB@submitsong%
    \ifnum\SB@grouplvl=\z@\let\SB@sgroup\@empty\fi%
    \stepcounter{songnum}%
  \else%
    \ifSB@intersong\SB@erreor\SB@closeall%
    \else\SB@erreot\fi%
  \fi%
}
```

`\SB@setbaselineskip` Set the `\baselineskip` to an appropriate line height.

```
\newcommand\SB@setbaselineskip{%
  \SB@dimen\f@size\p@%
  \baselineskip\SB@dimen\relax%
  \ifchorded%
    \setbox\SB@box\hbox{{\printchord{ABCDEFG\shrp\flt/j7}}}%
    \advance\baselineskip\ht\SB@box%
  \fi%
  \ifslides%
    \advance\baselineskip.2\SB@dimen\@plus.5\SB@dimen%
    \@minus.2\SB@dimen%
  \else%
    \advance\baselineskip\z@\@plus.1\SB@dimen\relax%
  \fi%
  \advance\baselineskip\baselineadj%
}
```

`\SB@setversesep` Set the `\versesep` to an appropriate amount.

```
\newcommand\SB@setversesep{%
  \SB@dimen123456789sp%
  \edef\SB@temp{\the\SB@dimen}%
  \edef\SB@tempii{\the\versesep}%
  \ifx\SB@temp\SB@tempii%
    \begingroup%
```

```
      \lyricfont%
      \SB@dimen\f@size\p@%
      \ifchorded%
        \setbox\SB@box\hbox{{\printchord{ABCDEFG\shrp\flt/j7}}}%
        \advance\SB@dimen\ht\SB@box%
      \fi%
      \ifslides%
        \global\versesep1.2\SB@dimen\@plus.3\SB@dimen%
        \@minus.3\SB@dimen%
      \else%
        \global\versesep.75\SB@dimen\@plus.25\SB@dimen%
        \@minus.13\SB@dimen%
      \fi%
    \endgroup%
  \fi%
}
```

`\makeprelude` Generate the material that begins each song. This macro is invoked at `\endsong` so that its code can access song info defined throughout the song.

```
\newcommand\makeprelude{%
  \resettitles%
  \ifslides%
    \hbox to\hsize{{%
      \hfil\stitlefont\songtitle\hfil%
    }}%
    \vskip5\p@%
    \hbox to\hsize{%
      \hfil%
      \vbox{%
        \divide\hsize\tw@\parskip\p@\relax%
        \centering\small\extendprelude%
      }%
      \hfil%
    }%
  \else%
    \ifdim\songnumwidth>\z@%
      \setbox\SB@boxii\hbox{{\SB@colorbox\snumbgcolor{%
        \hbox to\songnumwidth{%
          \printsongnum{\thesongnum}\hfil%
        }%
      }}}%
    \fi%
    \setbox\SB@box\vbox{%
      \ifnum\SB@numcols>\z@\hsize\SB@colwidth\fi%
      \ifdim\songnumwidth>\z@%
        \advance\hsize-\wd\SB@boxii%
        \advance\hsize-3\p@%
      \fi%
      \SB@raggedright\offinterlineskip\lineskip\p@%
      {\stitlefont%
```

```
      \songtitle\par%
      \nexttitle%
      \foreachtitle{(\songtitle)\par}}%
    \ifdim\prevdepth=\z@\kern\p@\fi%
    \parskip\p@\relax\tiny%
    \extendprelude%
    \kern\z@%
  }%
  \ifdim\songnumwidth>\z@%
    \hbox{%
      \ifdim\ht\SB@boxii>\ht\SB@box%
        \box\SB@boxii%
        \kern3\p@%
        \vtop{\box\SB@box}%
      \else%
        \SB@colorbox\snumbgcolor{\vbox to\ht\SB@box{{%
          \hbox to\songnumwidth{%
            \printsongnum{\thesongnum}\hfil%
          }\vfil%
        }}}%
        \kern3\p@%
        \box\SB@box%
      \fi%
    }%
  \else%
    \unvbox\SB@box%
  \fi%
\fi%
}
```

`\makepostlude` Generate the material that ends each song.

```
\newcommand\makepostlude{%
  \SB@raggedright\baselineskip\z@skip\parskip\z@skip\parindent\z@%
  \tiny\extendpostlude%
}
```

`\showauthors` Display the author line in the prelude.

```
\newcommand\showauthors{%
  \setbox\SB@box\hbox{\bfseries\sfcode`.\@m\songauthors}%
  \ifdim\wd\SB@box>\z@\unhbox\SB@box\par\fi%
}
```

`\showrefs` Display the scripture references in the prelude.

```
\newcommand\showrefs{%
  \setbox\SB@box\hbox{\slshape\songrefs\vphantom,}%
  \ifdim\wd\SB@box>\z@\unhbox\SB@box\par\fi%
}
```

\SB@next
\SB@donext
\SB@dothis

Several macros use \futurelet to look ahead in the input stream, and then take various actions depending on what is seen. In these macros, \SB@next is assigned the token seen, \SB@dothis is assigned the action to be taken on this loop iteration, and \SB@donext is assigned the action to be taken to continue (or terminate) the loop.

```
\newcommand\SB@next{}
\newcommand\SB@donext{}
\newcommand\SB@dothis{}
```

\SB@nextname

Sometimes when scanning ahead we \stringify the name of the next token. When that happens, the name is stored in this macro for safekeeping.

```
\newcommand\SB@nextname{}
```

\SB@appendsp

Append an explicit space token (catcode 10) to a token register. This is a useful macro to have around because inlining this code directly into a larger macro is harder than it seems: If you write the following code but with an explicit control sequence instead of #1, then the space immediately following the name will get stripped by the TEX parser. But invoking the following macro with a control sequence as an argument works fine, because in that case the explicit space has already been tokenized when this macro was first defined and won't be stripped as it is expanded.

```
\newcommand\SB@appendsp[1]{#1\expandafter{\the#1␣}}
```

\SB@parsetitles

Parse a list of song titles. This just involves removing leading and trailing spaces from around each title in the \\-separated list.

```
\newcommand\SB@parsetitles[1]{%
  \begingroup%
    \global\SB@titlelist{\\}%
    \SB@toks{}%
    \let\\\SB@titlesep%
    \let\SB@dothis\SB@pthead%
    \SB@ptstart#1\SB@endparse%
  \endgroup%
}
```

\SB@ptstart

The iterator of the title parser loop just scans the next token.

```
\newcommand\SB@ptstart{\futurelet\SB@next\SB@dothis}
```

\SB@pthead

While processing tokens at the head of a title, we skip over all spaces until we reach a non-space token.

```
\newcommand\SB@pthead{%
  \ifcat\noexpand\SB@next\noexpand\@sptoken%
    \expandafter\SB@ptsp%
  \else%
    \SB@toks{}%
    \let\SB@dothis\SB@ptmain%
    \expandafter\SB@ptmain%
  \fi%
}
```

`\SB@ptmain` Once we've reached a non-space token in the title, we consume the remainder of the title as-is, except that space tokens should be trimmed from the end of each title.

```
\newcommand\SB@ptmain{%
  \ifcat\noexpand\SB@next\noexpand\@sptoken%
    \let\SB@donext\SB@ptsp%
  \else\ifcat\noexpand\SB@next\noexpand\bgroup%
    \let\SB@donext\SB@ptbg%
  \else\ifx\SB@next\SB@endparse%
    \global\SB@titlelist\expandafter{\the\SB@titlelist\\}%
    \let\SB@donext\@gobble%
  \else%
    \ifx\SB@next\\%
      \SB@toks{}%
      \let\SB@dothis\SB@pthead%
    \fi%
    \let\SB@donext\SB@ptstep%
  \fi\fi\fi%
  \SB@donext%
}
```

`\SB@ptstep` Consume a non-space, non-left-brace token and add it to the current song title. If any spaces preceded it, add those too.

```
\newcommand\SB@ptstep[1]{%
  \global\SB@titlelist\expandafter\expandafter\expandafter{%
    \expandafter\the\expandafter\SB@titlelist\the\SB@toks#1}%
  \SB@toks{}%
  \SB@ptstart%
}
```

`\SB@ptbg` The next title token is a left-brace. It should be balanced, so consume the entire group and add it (along with its surrounding braces) as-is to the current title.

```
\newcommand\SB@ptbg[1]{\SB@ptstep{{#1}}}
```

`\SB@ptsp` The next title token is a space. We won't know whether to include it in the title until we see what follows it. Strings of spaces followed by the `\\` title-delimiter token, or that conclude a title argument, should be stripped. So rather than add the space token to the title, we remember it in a token register for possible later inclusion.

```
\newcommand\SB@ptsp{
  \SB@appendsp\SB@toks%
  \afterassignment\SB@ptstart%
  \let\SB@next= }
```

`\SB@titlesep` While parsing song titles, we temporarily assign `\\` a non-trivial top-level expansion (`\SB@titlesep`) in order to distinguish it from other macros.

```
\newcommand\SB@titlesep{SB@titlesep}
```

`\SB@endparse` The `\SB@endparse` token marks the end of a token sequence being parsed. If parsing works as intended, the macro should never be expanded, so produce an error if it is.

```
\newcommand\SB@endparse{%
  \SB@Error{Title parsing failed}{This error should not occur.}%
}
```

`\SB@testdigit` `\SB@@testdigit` The following decides whether a token or `\let`-defined control sequence is a digit and sets conditional `\ifSB@test` accordingly.

```
\newcommand\SB@testdigit[1]{%
  \SB@testfalse%
  \ifcat1\noexpand#1\SB@@testdigit#1\fi%
}
\newcommand\SB@@testdigit[1]{%
  \ifx0#1\SB@testtrue\else%
  \ifx1#1\SB@testtrue\else%
  \ifx2#1\SB@testtrue\else%
  \ifx3#1\SB@testtrue\else%
  \ifx4#1\SB@testtrue\else%
  \ifx5#1\SB@testtrue\else%
  \ifx6#1\SB@testtrue\else%
  \ifx7#1\SB@testtrue\else%
  \ifx8#1\SB@testtrue\else%
  \ifx9#1\SB@testtrue%
  \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi%
}
```

`\SB@parsesrefs` Store into `\songrefs` a processed version of a scripture reference in which the following adjustments have been made: (1) Spaces not preceded by a comma or semicolon are made non-breaking. For example, `2 John 1:1` and `Song of Solomon 1:1` become `2~John~1:1` and `Song~of~Solomon~1:1`. (2) Spaces between a semicolon and a book name are lengthened to en-spaces. (3) Single hyphens are lengthened to en-dashes (`--`). (4) Non-breaking, thin spaces are appended to commas not followed by a space. For example `John 3:16,17` becomes `John~3:16,\nobreak\thinspace17`. (5) Everything within an explicit group is left unchanged, allowing the user to suppress all of the above as desired.

To achieve this, we must change all commas, hyphens, and spaces in the scripture reference into active characters. Unfortunately, the catcodes of everything in the text were set back when the full keyval list was digested as an argument to `\beginsong`, so we must unset and reset the catcodes. One obvious solution is to use `\scantokens` from ε-TEX to do this, but that doesn't allow us to suppress the re-catcoding process within groups, and we'd like to avoid intoducing features that require ε-TEX anyway for compatibility reasons. Therefore, we build the following small scanner instead.

The scanner walks through the text token by token, replacing each important token by its active equivalent. No character codes are modified during this process and no tokens inserted because some of these tokens might end up being arguments to multi-byte unicode character rather than being expanded directly. The

inputenc package only cares about the character codes, not the category codes, so modifying only the category codes should be safe.

```
\newcommand\SB@parsesrefs[1]{%
  \begingroup%
    \SB@toks{\begingroup\SB@sractives}%
    \SB@prloop#1\SB@endparse%
    \xdef\songrefs{\the\SB@toks\endgroup}%
  \endgroup%
}
```

\SB@prloop
\SB@prstep
\SB@@prstep

The main loop of the scripture reference scanner identifies each space, hyphen, and comma for special treatment.

```
\newcommand\SB@prloop{\futurelet\SB@next\SB@prstep}
\newcommand\SB@prstep{%
  \ifcat\noexpand\SB@next A%
    \expandafter\SB@prcpy%
  \else%
    \expandafter\SB@@prstep%
  \fi%
}
\newcommand\SB@@prstep{%
  \ifcat\noexpand\SB@next\noexpand\@sptoken%
    \let\SB@donext\SB@prspace%
  \else\ifx\SB@next-%
    \let\SB@donext\SB@prhyphen%
  \else\ifx\SB@next,%
    \let\SB@donext\SB@prcomma%
  \else\ifx\SB@next\SB@endparse
    \let\SB@donext\@gobble%
  \else\ifcat\noexpand\SB@next\bgroup%
    \let\SB@donext\SB@prgr%
  \else%
    \let\SB@donext\SB@prcpy%
  \fi\fi\fi\fi\fi%
  \SB@donext%
}
```

\SB@prcpy
\SB@prgr

Anything that isn't one of the special tokens above, and anything in a group, is copied without modification.

```
\newcommand\SB@prcpy[1]{\SB@toks\expandafter{\the\SB@toks#1}\SB@prloop}
\newcommand\SB@prgr[1]{\SB@toks\expandafter{\the\SB@toks{#1}}\SB@prloop}
```

\SB@prcomma
\SB@prhyphen

Commas and hyphens are replaced with active equivalents.

```
\newcommand\SB@prcomma[1]{}
{\catcode`,\active
 \gdef\SB@prcomma#1{\SB@toks\expandafter{\the\SB@toks,}\SB@prloop}}
\newcommand\SB@prhyphen[1]{}
{\catcode`-\active
 \gdef\SB@prhyphen#1{\SB@toks\expandafter{\the\SB@toks-}\SB@prloop}}
```

\SB@prspace
\SB@@prspace

Spaces are made active as well, but doing so requires some specialized code since they cannot be consumed as implicit macro arguments.

```
\newcommand\SB@prspace[1]{}
{\obeyspaces
\gdef\SB@prspace{\SB@toks\expandafter{\the\SB@toks␣}\SB@@prspace}}
\newcommand\SB@@prspace{\afterassignment\SB@prloop\let\SB@temp= }
```

\SB@sractives

Assign macro definitions to active commas, hyphens, spaces, and returns when the token list generated by `\SB@parsesrefs` is used to typeset a scripture reference list.

```
\newcommand\SB@sractives{}
{\catcode‘,\active\catcode‘-\active\obeyspaces\catcode‘\^^M\active%
\gdef\SB@sractives{%
\let,\SB@srcomma\let-\SB@srhyphen%
\let␣\SB@srspace\let^^M\SB@srspace%
\SB@srspacing}%
}
```

\SB@srspacing

The space factors of semicolons and commas are what the active spaces within a scripture reference text use to decide what came before. The following sets them to their default values in case they have been changed, but sets all other space factors to 1000.

```
\newcommand\SB@srspacing{%
  \nonfrenchspacing\sfcode‘\;=1500\sfcode‘\,=1250\relax%
}
```

\SB@srcomma
\SB@@srcomma

Commas not already followed by whitespace are appended with a thin, non-breaking space.

```
\newcommand\SB@srcomma{,\futurelet\SB@next\SB@@srcomma}
\newcommand\SB@@srcomma{%
  \ifx\SB@next\SB@srspace\else%
    \nobreak\thinspace%
  \fi%
}
```

\SB@srhyphen
\SB@@srhyphen
\SB@srdash
\SB@@srdash

Hyphens that are not already part of a ligature (an en- or em-dash) become en-dashes.

```
\newcommand\SB@srhyphen{\futurelet\SB@next\SB@@srhyphen}
\newcommand\SB@@srhyphen{%
  \ifx\SB@next\SB@srhyphen\expandafter\SB@srdash\else--\fi%
}
\newcommand\SB@srdash[1]{\futurelet\SB@next\SB@@srdash}
\newcommand\SB@@srdash{%
  \ifx\SB@next\SB@srhyphen---\expandafter\@gobble\else--\fi%
}
```

\SB@srspace
\SB@@srspace

To compress consecutive whitespace, we ignore spaces immediately followed by more whitespace. Spaces not preceded by a semicolon or comma become non-breaking. Most spaces following a semicolon become en-spaces with favorable breakpoints, but a special case arises for spaces between a semicolon and a digit (see `\SB@srcso` below).

```
\newcommand\SB@srspace{\futurelet\SB@next\SB@@srspace}
\newcommand\SB@@srspace{%
  \let\SB@donext\relax%
  \ifx\SB@next\SB@srspace\else%
    \ifnum\spacefactor>\@m%
      \ifnum\spacefactor>1499 %
        \ifcat\noexpand\SB@next0%
          \let\SB@donext\SB@srcso%
        \else%
          \penalty-5\enskip%
        \fi%
      \else%
        \space%
      \fi%
    \else%
      \nobreak\space%
    \fi%
  \fi%
  \SB@donext%
}
```

\SB@srcso
\SB@@srcso

A space between a semicolon and a digit could be within a list of verse references for a common book (e.g., `Job 1:1; 2:2`); or it could separate the previous book from a new book whose name starts with a number (e.g., `Job 1:1; 1 John 1:1`). In the former case, we should just use a regular space; but in the latter case we should be using an en-space with a favorable breakpoint. To distinguish between the two, we peek ahead at the next two tokens. If the second one is a space, assume the latter; otherwise assume the former.

```
\newcommand\SB@srcso[1]{\futurelet\SB@temp\SB@@srcso}
\newcommand\SB@@srcso{%
  \ifx\SB@temp\SB@srspace%
    \penalty-5\enskip%
  \else%
    \space%
  \fi%
  \SB@next%
}
```

16.6 Verses and Choruses

The following programming typesets song contents, including verses, choruses, and textual notes.

`\ifSB@stanza` The following conditional remembers if we've seen any stanzas yet in the current song.

```
\newif\ifSB@stanza
```

`\SB@stanzabreak` End this song stanza and start a new one.

```
\newcommand\SB@stanzabreak{%
  \ifhmode\par\fi%
  \ifSB@stanza%
    \SB@breakpoint{%
      \ifSB@inverse%
        \ifSB@prevverse\vvpenalty\else\cvpenalty\fi%
      \else%
        \ifSB@prevverse\vcpenalty\else\ccpenalty\fi%
      \fi%
    }%
    \vskip\versesep%
  \fi%
}
```

`\SB@breakpoint` Insert a valid breakpoint into the vertical list comprising a song.

```
\newcommand\SB@breakpoint[1]{%
  \begingroup%
    \ifnum#1<\@M%
      \SB@skip\colbotglue\relax%
      \SB@skip-\SB@skip%
    \else%
      \SB@skip\z@skip%
    \fi%
    \advance\SB@skip\lastskip%
    \unskip%
    \nobreak%
    \ifnum#1<\@M%
      \vskip\colbotglue\relax%
      \penalty#1%
    \fi%
    \vskip\SB@skip%
  \endgroup%
}
```

`\SB@putbox` Unbox a vbox and follow it by vertical glue if its depth is unusually shallow. This ensures that verses and choruses will look equally spaced even if one of them has a final line with no letters that dangle below the baseline.

```
\newcommand\SB@putbox[2]{%
  \begingroup%
    \SB@dimen\dp#2%
    #1#2%
    \setbox\SB@box\hbox{{\lyricfont p}}%
    \ifdim\SB@dimen<\dp\SB@box%
      \advance\SB@dimen-\dp\SB@box%
```

```
      \vskip-\SB@dimen%
    \fi%
    \setbox\SB@box\box\voidb@x%
  \endgroup%
}
```

`\SB@obeylines` Within verses and choruses we would like to use `\obeylines` so that each EOL in the source file ends a paragraph without having to say `\par` explicitly. The LaTeX base code establishes the convention that short-term changes to `\par` will restore `\par` by setting it equal to `\@par`. Long-term (i.e., environment-long) changes to `\par` should therefore redefine `\@par` to restore the desired long-term definition. The following code starts a long-term redefinition of `\par` adhering to these conventions, and extends that definition to end-of-line as well.

```
\newcommand\SB@obeylines{%
  \let\par\SB@par%
  \obeylines%
  \let\@par\SB@@par%
}
```

`\SB@par` The following replacement definition of `\par` constructs paragraphs in which page-breaks are disallowed, since no wrapped line in a song should span a page- or column-break. It then inserts an interlinepenalty after the paragraph so that such penalties will appear between consecutive lines in each verse. (Note: The `\endgraf` macro must not be uttered within a local group since this prevents parameters like `\hangindent` from being reset at the conclusion of each paragraph.)

```
\newcommand\SB@par{%
  \ifhmode%
    \SB@cnt\interlinepenalty%
    \interlinepenalty\@M%
    \endgraf%
    \interlinepenalty\SB@cnt%
    \ifSB@inchorus%
      \ifdim\cbarwidth>\z@\nobreak\else\SB@ilpenalty\fi%
    \else%
      \SB@ilpenalty%
    \fi%
  \fi%
}
```

`\SB@ilpenalty` By default, breaking a vertical list between paragraphs incurs a penalty of zero. Thus, we only insert an explicit penalty between lines if `\interlinepenalty` is non-zero. This avoids cluttering the vertical list with superfluous zero penalties.

```
\newcommand\SB@ilpenalty{%
  \ifnum\interlinepenalty=\z@\else%
    \penalty\interlinepenalty%
  \fi%
}
```

`\SB@@par` This replacement definition of `\@par` restores the `\SB@par` definition of `\par` and then ends the paragraph.

```
\newcommand\SB@@par{\let\par\SB@par\par}
```

`\SB@parindent` Reserve a length to remember the current `\parindent`.

```
\newdimen\SB@parindent
```

`\SB@everypar` Reserve a control sequence to hold short-term changes to `\everypar`.

```
\newcommand\SB@everypar{}
```

`\SB@raggedright` Perform `\raggedright` except don't nuke the `\parindent`.

```
\newcommand\SB@raggedright{%
  \SB@parindent\parindent%
  \raggedright%
  \parindent\SB@parindent%
}
```

`\vnumbered` The following conditional remembers whether this verse is being numbered or not (i.e., it distinguishes between `\beginverse` and `\beginverse*`).

```
\newif\ifvnumbered
```

`\ifSB@prevverse` Reserve a conditional to remember if the previous block in this song was a verse.

```
\newif\ifSB@prevverse
```

Before replacing the little-used `verse` environment with a new one, issue a warning if the current definition of `\verse` is not the LaTeX-default one. This may indicate a package clash.

```
\CheckCommand\verse{%
  \let\\\@centercr%
  \list{}{%
    \itemsep\z@%
    \itemindent-1.5em%
    \listparindent\itemindent%
    \rightmargin\leftmargin%
    \advance\leftmargin1.5em%
  }%
  \item\relax%
}
```

`verse` `verse*` `\beginverse` Begin a new verse. This can be done by beginning a `verse` environment or by using the `\beginverse` macro. The latter must check for a trailing star to determine if this verse should be numbered. We use `\@ifstar` to scan ahead for the star, but this needs to be done carefully because while scanning we might encounter tokens that should be assigned different catcodes once the verse really begins. Thus, we temporarily invoke `\SB@loadactives` for the duration of `\@ifstar` so that everything gets the right catcode.

```
\renewenvironment{verse}
  {\vnumberedfalse\SB@beginverse}
```

```
  {\SB@endverse}
\newenvironment{verse*}
  {\vnumberedtrue\SB@beginverse}
  {\SB@endverse}
\newcommand\beginverse{%
  \begingroup%
    \SB@loadactives%
    \@ifstar{\endgroup\vnumberedfalse\SB@beginverse}%
            {\endgroup\vnumberedtrue\SB@beginverse}%
}
```

`\SB@beginverse` Start the body of a verse. We begin by inserting a mark if `\repchoruses` is active and this verse was preceded by a numbered verse (making this an eligible place to insert a chorus later).

Verse numbering is implemented using `\everypar` so that if there is any vertical material between the `\beginverse` and the first line of the verse, that material will come before the verse number. Intervening horizontal material (e.g., `\textnote`) can temporarily clear `\everypar` to defer the verse number until later.

```
\newcommand\SB@beginverse{%
  \ifSB@insong%
    \ifSB@inverse\SB@errbvv\endverse\fi%
    \ifSB@inchorus\SB@errbvc\endchorus\fi%
  \else%
    \SB@errbvt\beginsong{Unknown Song}%
  \fi%
  \ifrepchorus\ifvoid\SB@chorusbox\else%
    \SB@gotchorustrue%
    \ifSB@prevverse\ifvnumbered%
      \marks\SB@cmarkclass{\SB@cmark}%
    \fi\fi%
  \fi\fi%
  \SB@inversetrue%
  \def\SB@closeall{\endverse\endsong}%
  \SB@stanzabreak%
  \versemark\nobreak%
  \global\SB@stanzatrue%
  \SB@ifempty\SB@cr@\memorize{\replay[]}%
  \setbox\SB@box\vbox\bgroup\begingroup%
    \ifvnumbered%
      \protected@edef\@currentlabel{\p@versenum\theversenum}%
      \def\SB@everypar{%
        \setbox\SB@box\hbox{{%
          \printversenum{\theversenum}%
        }}%
        \ifdim\wd\SB@box<\versenumwidth%
          \setbox\SB@box%
          \hbox to\versenumwidth{\unhbox\SB@box\hfil}%
        \fi%
        \ifchorded\vrule\@height\baselineskip\@width\z@\@depth\z@\fi%
```

```
        {\placeversenum\SB@box}%
        \gdef\SB@everypar{}%
      }%
    \else%
      \def\SB@everypar{%
        \ifchorded\vrule\@height\baselineskip\@width\z@\@depth\z@\fi%
        \gdef\SB@everypar{}%
      }%
    \fi%
    \everypar{\SB@everypar\everypar{}}%
    \versefont\versejustify%
    \SB@loadactives%
    \SB@obeylines%
    \penalty12345 %
    \everyverse\relax%
}
```

`\SB@endverse` End a verse. This involves unboxing the verse material with `\SB@putbox`, which corrects for last lines that are unusually shallow.

```
\newcommand\SB@endverse{%
  \ifSB@insong%
    \ifSB@inverse%
        \unpenalty%
      \endgroup\egroup%
      \SB@putbox\unvbox\SB@box%
      \SB@inversefalse%
      \def\SB@closeall{\endsong}%
      \ifvnumbered\stepcounter{versenum}\fi%
      \SB@prevversetrue%
    \else\ifSB@inchorus\SB@errevc\endchorus%
    \else\SB@errevo\fi\fi%
  \else%
    \SB@errevt%
  \fi%
}
```

`\ifSB@chorustop` When a chorus is broken in to several pieces by column-breaks (via `\brk`), the following conditional remembers whether the current piece is the topmost one for this chorus.

```
\newif\ifSB@chorustop
```

`\SB@chorusbox` When `\repchoruses` is used, the first sequence of consecutive choruses is remembered in the following box register.

```
\newbox\SB@chorusbox
```

`\ifSB@gotchorus` The following conditional remembers whether we've completed storing the first block of consecutive choruses.

```
\newif\ifSB@gotchorus
```

\SB@cmarkclass
\SB@nocmarkclass

The \repeatchoruses feature requires the use of two extended mark classes provided by ε-TEX. We use the \newmarks macro to allocate these classes, if it's available. If \newmarks doesn't exist, then that means the user has an ε-TEX compatible version of LATEX, but no etex style file to go with it; we just have to pick two mark classes and hope that nobody else is using them.

```
\ifSB@etex
  \@ifundefined{newmarks}{
    \@ifundefined{newmark}{
      \mathchardef\SB@cmarkclass83
      \mathchardef\SB@nocmarkclass84
    }{
      \newmark\SB@cmarkclass
      \newmark\SB@nocmarkclass
    }
  }{
    \newmarks\SB@cmarkclass
    \newmarks\SB@nocmarkclass
  }
\fi
```

\SB@cmark
\SB@lastcmark
\SB@nocmark

To determine where choruses should be inserted when \repchoruses is active, three kinds of marks are inserted into song boxes: \SB@cmark is used to mark places where a chorus might be inserted between verses, and \SB@lastcmark marks a place where a chorus might be inserted after the last verse of the song. Both marks are ε-TEX marks of class \SB@cmarkclass, to avoid disrupting the use of standard TEX marks. Each time a chorus is automatically inserted, \SB@nocmark is inserted with mark class \SB@nocmarkclass just above it (and at the top of each additional page it spans). This inhibits future chorus inserts until the already-inserted chorus has been fully committed to the output file. Otherwise some choruses could get auto-inserted multiple times at the same spot, possibly even leading to an infinite loop!

```
\newcommand\SB@cmark{}
\def\SB@cmark{SB@cmark}
\newcommand\SB@lastcmark{}
\def\SB@lastcmark{SB@lastcmark}
\newcommand\SB@nocmark{}
\def\SB@nocmark{SB@nocmark}
```

chorus
\beginchorus

Start a new chorus. If \repchoruses is active and this is part of the first set of consecutive choruses in the song, then include it and its preceding vertical material in the \SB@chorusbox for possible later duplication elsewhere.

```
\newenvironment{chorus}{\beginchorus}{\SB@endchorus}
\newcommand\beginchorus{%
  \ifSB@insong
    \ifSB@inverse\SB@errbcv\endverse\fi%
    \ifSB@inchorus\SB@errbcc\endchorus\fi%
  \else%
    \SB@errbct\beginsong{Unknown Song}%
```

```
  \fi%
  \SB@inchorustrue%
  \def\SB@closeall{\endchorus\endsong}%
  \SB@chorustoptrue%
  \vnumberedfalse%
  \SB@stanzabreak%
  \chorusmark%
  \ifrepchorus%
    \ifSB@gotchorus\else\ifSB@prevverse\else%
      \global\setbox\SB@chorusbox\vbox{%
        \unvbox\SB@chorusbox%
        \SB@stanzabreak%
        \chorusmark%
      }%
    \fi\fi%
  \fi%
  \global\SB@stanzatrue%
  \replay[]%
  \SB@@beginchorus%
  \everychorus\relax%
}
```

`\SB@@beginchorus` Begin the body of a chorus, or continue the body of a chorus after `\brk` has paused it to insert a valid breakpoint. We insert an empty class-`\SB@cmarkclass` mark here so that this chorus will not be duplicated elsewhere on the same page(s) where it initially appears.

```
\newcommand\SB@@beginchorus{%
  \ifrepchorus\marks\SB@cmarkclass{}\fi%
  \setbox\SB@box\vbox\bgroup\begingroup%
    \ifchorded%
      \def\SB@everypar{%
        \vrule\@height\baselineskip\@width\z@\@depth\z@%
        \gdef\SB@everypar{}%
      }%
      \everypar{\SB@everypar\everypar{}}%
    \fi%
    \chorusfont\chorusjustify%
    \SB@loadactives%
    \SB@obeylines%
    \penalty12345 %
}
```

`\SB@endchorus` End a chorus. This involves creating the vertical line to the left of the chorus and then unboxing the chorus material that was previously accumulated.

```
\newcommand\SB@endchorus{%
  \ifSB@insong%
    \ifSB@inchorus%
        \unpenalty%
      \endgroup\egroup%
```

```
      \SB@inchorusfalse%
      \def\SB@closeall{\endsong}%
      \setbox\SB@box\vbox{%
        \SB@chorusbar\SB@box%
        \SB@putbox\unvbox\SB@box%
      }
      \ifrepchorus\ifSB@gotchorus\else%
        \global\setbox\SB@chorusbox\vbox{%
          \unvbox\SB@chorusbox%
          \unvcopy\SB@box%
        }%
      \fi\fi%
      \unvbox\SB@box%
      \SB@prevversefalse%
    \else\ifSB@inverse\SB@errecv\endverse%
    \else\SB@erreco\fi\fi%
  \else%
    \SB@errect%
  \fi%
}
```

`\SB@cbarshift` Increase `\leftskip` to accommodate the chorus bar, if any.

```
\newcommand\SB@cbarshift{%
  \ifSB@inchorus\ifdim\cbarwidth>\z@%
    \advance\leftskip\cbarwidth%
    \advance\leftskip5\p@\relax%
  \fi\fi%
}
```

`\SB@chorusbar` Create the vertical bar that goes to the left of a chorus. Rather than boxing up the chorus in order to put the bar to the left, the bar is introduced as leaders directly into the vertical list of the main song box. This allows it to stretch and shrink when a column is typeset by the page-builder.

```
\newcommand\SB@chorusbar[1]{%
  \ifdim\cbarwidth>\z@%
    \SB@dimen\ht#1%
    \SB@dimenii\dp#1%
    \advance\SB@dimen%
      \ifSB@chorustop\ifchorded\else2\fi\fi\SB@dimenii%
    \SB@skip\SB@dimen\relax%
    \SB@computess\SB@skip1\@plus#1%
    \SB@computess\SB@skip{-1}\@minus#1%
    \nointerlineskip\null\nobreak%
    \leaders\vrule\@width\cbarwidth\vskip\SB@skip%
    \ifSB@chorustop\ifchorded\else%
      \advance\SB@skip-\SB@dimenii%
    \fi\fi%
    \nobreak\vskip-\SB@skip%
  \fi%
}
```

`\SB@computess` This computes the stretchability or shrinkability of a vbox and stores the result in the skip register given by ⟨*arg1*⟩. If ⟨*arg2*⟩=1 and ⟨*arg3*⟩ is "plus", then the stretchability of box ⟨*arg4*⟩ is added to the plus component of ⟨*arg1*⟩. If ⟨*arg2*⟩=-1 and ⟨*arg3*⟩ is "minus", then the shrinkability of the box is added to the minus component of ⟨*arg1*⟩. If the stretchability or shrinkability is infinite, then we guess 1fil for that component.

```
\newcommand\SB@computess[4]{%
  \begingroup%
    \vbadness\@M\vfuzz\maxdimen%
    \SB@dimen4096\p@%
    \setbox\SB@box\vbox spread#2\SB@dimen{\unvcopy#4}%
    \ifnum\badness=\z@%
      \global\advance#1\z@#31fil\relax%
    \else%
      \SB@dimenii\SB@dimen%
      \loop%
        \SB@dimenii.5\SB@dimenii%
        \ifnum\badness<100 %
          \advance\SB@dimen\SB@dimenii%
        \else
          \advance\SB@dimen-\SB@dimenii%
        \fi%
        \setbox\SB@box\vbox spread#2\SB@dimen{\unvcopy#4}%
        \ifnum\badness=100 \SB@dimenii\z@\fi%
      \ifdim\SB@dimenii>.1\p@\repeat%
      \ifdim\SB@dimen<.1\p@\SB@dimen\z@\fi%
      \global\advance#1\z@#3\SB@dimen\relax%
    \fi%
  \endgroup%
}
```

`\brk` Placing `\brk` within a line in a verse or chorus tells TEX to break the line at that point (if it needs to be broken at all).

Placing `\brk` on a line by itself within a chorus stops the chorus (and its vertical bar), inserts a valid breakpoint, and then restarts the chorus with no intervening space so that if the breakpoint isn't used, there will be no visible effect. Placing it on a line by itself within a verse just inserts a breakpoint.

Placing `\brk` between songs forces a column- or page-break, but only if generating a non-partial list of songs. When generating a partial list, `\brk` between songs is ignored.

```
\newcommand\brk{%
  \ifSB@insong%
    \ifhmode\penalty-5 \else%
      \unpenalty%
      \ifSB@inchorus%
        \ifdim\cbarwidth=\z@%
          \ifrepchorus\marks\SB@cmarkclass{}\fi%
          \SB@breakpoint\brkpenalty%
```

```
        \else%
          \endgroup\egroup%
          \ifrepchorus\ifSB@gotchorus\else%
            \global\setbox\SB@chorusbox\vbox{%
              \unvbox\SB@chorusbox%
              \SB@chorusbar\SB@box%
              \unvcopy\SB@box%
              \SB@breakpoint\brkpenalty%
            }%
          \fi\fi%
          \SB@chorusbar\SB@box%
          \unvbox\SB@box%
          \SB@breakpoint\brkpenalty%
          \SB@chorustopfalse%
          \SB@@beginchorus%
        \fi%
      \else%
        \SB@breakpoint\brkpenalty%
      \fi%
    \fi%
  \else%
    \ifpartiallist\else\SB@nextcol\@ne\colbotglue\fi%
  \fi%
}
```

`\SB@boxup` Typeset a shaded box containing a textual note to singers or musicians. We first try typesetting the note on a single line. If it's too big, then we try again in paragraph mode with full justification.

```
\newcommand\SB@boxup[1]{%
  \setbox\SB@box\hbox{{\notefont#1}}%
  \SB@dimen\wd\SB@box%
  \advance\SB@dimen6\p@%
  \advance\SB@dimen\leftskip%
  \advance\SB@dimen\rightskip%
  \ifdim\SB@dimen>\hsize%
    \vbox{{%
      \advance\hsize-6\p@%
      \advance\hsize-\leftskip%
      \advance\hsize-\rightskip%
      \notejustify%
      \unhbox\SB@box\par%
      \kern\z@%
    }}%
  \else%
    \vbox{\box\SB@box\kern\z@}%
  \fi%
}
```

`\textnote` Create a textual note for singers and musicians. If the note begins a verse or chorus, it should not be preceded by any spacing. Verses and choruses begin with

the sentinel penalty 12345, so we check `\lastpenalty` to identify this case. When typesetting the note, we must be sure to temporarily clear `\everypar` to inhibit any verse numbering that might be pending. We also readjust the `\baselineskip` as if we weren't doing chords, since no chords go above a textual note.

```
\newcommand\textnote[1]{%
  \ifhmode\par\fi%
  \ifnum\lastpenalty=12345\else%
    \ifSB@inverse%
      \vskip2\p@\relax%
    \else\ifSB@inchorus%
      \vskip2\p@\relax%
    \else\ifSB@stanza%
      \nobreak\vskip\versesep%
    \fi\fi\fi%
  \fi%
  \begingroup%
    \everypar{}%
    \ifchorded\chordedfalse\SB@setbaselineskip\chordedtrue\fi%
    \placenote{\SB@colorbox\notebgcolor{\SB@boxup{#1}}}%
  \endgroup%
  \nobreak%
  \ifSB@inverse%
    \vskip2\p@\relax%
  \else\ifSB@inchorus%
    \vskip2\p@\relax%
  \else\ifSB@stanza\else%
    \nobreak\vskip\versesep%
  \fi\fi\fi%
}
```

`\musicnote` Create a textual note for musicians.

```
\newcommand\musicnote[1]{\ifchorded\textnote{#1}\fi}
```

`\echo` `\SB@echo` `\SB@@echo` Typeset an echo part in the lyrics. Echo parts will be oblique and parenthesized. We toggle between oblique and upright shapes like `\emph`, but we use `\slshape` instead of `\itshape` because it tends to look nicer with the larger fonts used in slides mode.

The `\echo` macro must be able to accept chords in its argument. This complicates the implementation because chord macros should change catcodes, but if we grab `\echo`'s argument in the usual way then all the catcodes will be set before the chord macros have a chance to change them. This would disallow chord name abbreviations like `#` and `&` within `\echo` parts.

If we're using ε-TEX then the solution is easy: we use `\scantokens` to rescan the argument and thereby re-assign the catcodes. (One subtlety: Whenever LaTeX consumes an argument to a macro, it changes `#` to `##` so that when the argument text is substituted into the body of the macro, the replacement text will not contain unsubstituted parameters (such as `#1`). If `\scantokens` is used on the replacement text and the scanned tokens assign a new catcode to `#`, that

causes #'s to be doubled in the *output*, which was not the intent. To avoid this problem, we use `\@sanitize` before consuming the argument to `\echo`, which sets the catcodes of most special tokens (including #) to 12, so that LaTeX will not recognize any of them as parameters and will therefore not double any of them.)

```
\ifSB@etex
  \newcommand\echo{\begingroup\@sanitize\SB@echo}
  \newcommand\SB@echo[1]{%
    \endgroup%
    \begingroup%
      \ifdim\fontdimen\@ne\font>\z@\upshape\else\slshape\fi%
      \endlinechar\m@ne%
      \scantokens{(#1)}%
    \endgroup%
  }
\else
```

If we're not using ε-TeX, we must do something more complicated. We set up the appropriate font within a local group and finish with `\hbox` so that the argument to `\echo` is treated as the body of the box. Control is reacquired after the box using `\aftergroup`, whereupon we unbox the box and insert the closing parenthesis. This almost works except that if the last thing in an echo part is a long chord name atop a short lyric, the closing parenthesis will float out away from the lyric instead of being sucked under the chord. I can find no solution to this problem, so to avoid it users must find a version of LaTeX that is ε-TeX compatible.

```
  \newcommand\echo{%
    \begingroup%
      \ifdim\fontdimen\@ne\font>\z@\upshape\else\slshape\fi%
      \afterassignment\SB@echo%
      \setbox\SB@box\hbox%
  }
  \newcommand\SB@echo{\aftergroup\SB@@echo(}
  \newcommand\SB@@echo{\unhbox\SB@box)\endgroup}
\fi
```

`\rep` Place `\rep{`$\langle n \rangle$`}` at the end of a line to indicate that it should be sung $\langle n \rangle$ times.

```
\newcommand\rep[1]{%
  (\raise.25ex\hbox{%
    \fontencoding{OMS}\fontfamily{cmsy}\selectfont\char\tw@%
   }#1)%
}
```

16.7 Scripture Quotations

The macros in this section typeset scripture quotations and other between-songs environments.

`songgroup` A `songgroup` environment associates all enclosed environments with the enclosed song. When generating a partial list, all the enclosed environments are contributed if and only if the enclosed song is contributed.

```
\newenvironment{songgroup}{%
  \ifnum\SB@grouplvl=\z@%
    \edef\SB@sgroup{\thesongnum}%
    \global\SB@groupcnt\m@ne%
  \fi%
  \advance\SB@grouplvl\@ne%
}{%
  \advance\SB@grouplvl\m@ne%
  \ifnum\SB@grouplvl=\z@\let\SB@sgroup\@empty\fi%
}
```

`\SB@grouplvl` Count the `songgroup` environment nesting depth.

```
\newcount\SB@grouplvl
```

`intersong` An intersong block contributes vertical material to a column between the songs of a songs section. It is subject to the same column-breaking algorithm as real songs, but receives none of the other formatting applied to songs.

```
\newenvironment{intersong}{%
  \ifSB@insong\SB@errbro\SB@closeall\fi%
  \ifSB@intersong\SB@errbrr\SB@closeall\fi%
  \setbox\SB@chorusbox\box\voidb@x%
  \SB@intersongtrue%
  \def\SB@closeall{\end{intersong}}%
  \setbox\SB@songbox\vbox\bgroup\begingroup%
    \ifnum\SB@numcols>\z@\hsize\SB@colwidth\fi%
    \ifdim\sbarheight>\z@%
      \hrule\@height\sbarheight\@width\hsize%
      \nobreak%
    \fi%
}{%
  \ifSB@intersong
      \ifdim\sbarheight>\z@%
        \ifhmode\par\fi%
        \SB@skip\lastskip%
        \unskip\nobreak\vskip\SB@skip%
        \hbox{\vrule\@height\sbarheight\@width\hsize}%
      \fi%
    \endgroup\egroup%
    \ifSB@omitscrip%
      \setbox\SB@songbox\box\voidb@x%
    \else%
      \SB@submitsong%
    \fi%
    \SB@intersongfalse%
  \else%
    \ifSB@insong\SB@errero\SB@closeall\else\SB@errert\fi%
```

```
  \fi%
}
```

The starred form contributes page-spanning vertical material directly to the top of the nearest fresh page.

```
\newenvironment{intersong*}{%
  \ifSB@insong\SB@errbro\SB@closeall\fi%
  \ifSB@intersong\SB@errbrr\SB@closeall\fi%
  \setbox\SB@chorusbox\box\voidb@x%
  \SB@intersongtrue%
  \def\SB@closeall{\end{intersong*}}%
  \setbox\SB@songbox\vbox\bgroup\begingroup%
}{%
  \ifSB@intersong%
    \endgroup\egroup%
    \ifSB@omitscrip%
      \setbox\SB@songbox\box\voidb@x%
    \else%
      \def\SB@stype{\SB@styppage}%
      \SB@submitsong%
      \def\SB@stype{\SB@stypcol}%
    \fi%
    \SB@intersongfalse%
  \else%
    \ifSB@insong\SB@errero\SB@closeall\else\SB@errert\fi%
  \fi%
}
```

`\SB@srbox` The following box register holds the citation information that is to be typeset at the end of a scripture quotation.

```
\newbox\SB@srbox
```

`scripture` `\beginscripture` Begin a scripture quotation. We first store the reference in a box for later use, and then set up a suitable environment for the quotation. Quotations cannot typically be reworded if line-breaking fails, so we set `\emergencystretch` to a relatively high value at the outset.

```
\newenvironment{scripture}{\beginscripture}{\SB@endscripture}
\newcommand\beginscripture[1]{%
  \begin{intersong}%
    \SB@parsesrefs{#1}%
    \setbox\SB@srbox\hbox{{\printscrcite\songrefs}}%
    \def\SB@closeall{\endscripture}%
    \nobreak\vskip5\p@%
    \SB@parindent\parindent\parindent\z@%
    \parskip\z@skip\parfillskip\@flushglue%
    \leftskip\SB@parindent\rightskip\SB@parindent\relax%
    \scripturefont%
    \baselineskip\f@size\p@\@plus\p@\relax%
    \advance\baselineskip\p@\relax%
    \emergencystretch.3em%
```

```
}
```

`\SB@endscripture` End a scripture quotation.

```
\newcommand\SB@endscripture{%
  \ifSB@intersong
      \scitehere%
      \ifhmode\par\fi%
      \vskip-3\p@%
    \end{intersong}%
  \fi%
}
```

`\scitehere` Usually the scripture citation should just come at the `\endscripture` line, but at times the user might want to invoke this macro explicitly at a more suitable point. A good example is when something near the end of the scripture quotation drops TEX into vertical mode. In such cases, it is often better to issue the citation before leaving horizontal mode.

In any case, this macro should work decently whether in horizontal or vertical mode. In horizontal mode life is easy: we just append the reference to the current horizontal list using the classic code from p. 106 of The TEXbook. However, if we're now in vertical mode, the problem is a little harder. We do the best we can by using `\lastbox` to remove the last line, then adding the reference and re-typesetting it. This isn't as good as the horizontal mode solution because TEX only gets to reevaluate the last line instead of the whole paragraph, but usually the results are passable.

```
\newcommand\scitehere{%
  \ifSB@intersong%
    \ifvoid\SB@srbox\else%
      \ifvmode%
        \setbox\SB@box\lastbox%
        \nointerlineskip\noindent\hskip-\leftskip%
        \unhbox\SB@box\unskip%
      \fi%
      \unskip\nobreak\hfil\penalty50\hskip.8em\null\nobreak\hfil%
      \box\SB@srbox\kern-\SB@parindent%
      {\parfillskip\z@\finalhyphendemerits2000\par}%
    \fi%
  \else%
    \SB@errscrip\scitehere%
  \fi%
}
```

`\Acolon` Typeset a line of poetry in a scripture quotation.

`\Bcolon`

```
\newcommand\Acolon{\SB@colon2\Acolon}
\newcommand\Bcolon{\SB@colon1\Bcolon}
```

\SB@colon Begin a group of temporary definitions that will end at the next EOL. The EOL will end the paragraph and close the local scope.

```
\newcommand\SB@colon[2]{%
  \ifSB@intersong\else%
    \SB@errscrip#2%
    \beginscripture{Unknown}%
  \fi%
  \ifhmode\par\fi%
  \begingroup%
    \rightskip\SB@parindent\@plus4em%
    \advance\leftskip2\SB@parindent%
    \advance\parindent-#1\SB@parindent%
    \def\par{\endgraf\endgroup}%
    \obeylines%
}
```

\strophe Insert blank space indicative of a strophe division in a scripture quotation.

```
\newcommand\strophe{%
  \ifSB@intersong\else%
    \SB@errscrip\strophe\beginscripture{Unknown}%
  \fi%
  \vskip.9ex\@plus.45ex\@minus.68ex\relax%
}
```

\scripindent \scripoutdent \SB@scripdent Create an indented sub-block within a scripture quotation.

```
\newcommand\SB@scripdent[2]{%
  \ifSB@intersong\else%
    \SB@errscrip#2\beginscripture{Unknown}%
  \fi%
  \ifhmode\par\fi%
  \advance\leftskip#1\SB@parindent\relax%
}
\newcommand\scripindent{\SB@scripdent1\scripindent}
\newcommand\scripoutdent{\SB@scripdent-\scripoutdent}
```

\shiftdblquotes \SB@ldqleft \SB@ldqright \SB@rdqleft \SB@rdqright \SB@scanlq \SB@scanrq \SB@dolq \SB@dorq The Zaph Chancery font used by default to typeset scripture quotations seems to have some kerning problems with double-quote ligatures. The \shiftdblquotes macro allows one to modify the spacing around all double-quotes until the current group ends.

```
\newcommand\SB@quotesactive{%
  \catcode‘’\active%
  \catcode‘‘\active%
}
\newcommand\shiftdblquotes[4]{}
\newcommand\SB@ldqleft{}
\newcommand\SB@ldqright{}
\newcommand\SB@rdqleft{}
\newcommand\SB@rdqright{}
\newcommand\SB@scanlq{}
```

```
\newcommand\SB@scanrq{}
\newcommand\SB@dolq{}
\newcommand\SB@dorq{}
{
  \SB@quotesactive
  \gdef\shiftdblquotes#1#2#3#4{%
    \def\SB@ldqleft{\kern#1}%
    \def\SB@ldqright{\kern#2}%
    \def\SB@rdqleft{\kern#3}%
    \def\SB@rdqright{\kern#4}%
    \SB@quotesactive%
    \def‘{\futurelet\SB@next\SB@scanlq}%
    \def’{\futurelet\SB@next\SB@scanrq}%
  }
  \gdef\SB@scanlq{%
    \ifx\SB@next‘%
      \expandafter\SB@dolq%
    \else%
      \expandafter\lq%
    \fi%
  }
  \gdef\SB@scanrq{%
    \ifx\SB@next’%
      \expandafter\SB@dorq%
    \else%
      \expandafter\rq%
    \fi%
  }
  \gdef\SB@dolq‘{%
    \ifvmode\leavevmode\else\/\fi%
    \vadjust{}%
    \SB@ldqleft\lq\lq\SB@ldqright%
    \vadjust{}%
  }
  \gdef\SB@dorq’{%
    \ifvmode\leavevmode\else\/\fi%
    \vadjust{}%
    \SB@rdqleft\rq\rq\SB@rdqright%
    \vadjust{}%
  }
}
```

16.8 Transposition

The macros that transpose chords are contained in this section.

`\SB@transposefactor` This counter identifies the requested number of halfsteps by which chords are to be transposed (from -11 to +11).

```
\newcount\SB@transposefactor
```

\ifSB@convertnotes Even when transposition is not requested, the transposition logic can be used to automatically convert note names to another form. The following conditional turns that feature on or off.

```
\newif\ifSB@convertnotes
```

\notenameA \notenameB \notenameC \notenameD \notenameE \notenameF \notenameG Reserve a control sequence for each note of the diatonic scale. These will be used to identify which token sequences the input file uses to denote the seven scale degrees. Their eventual definitions *must* consist entirely of uppercase letters, and they must be assigned using `\def`, but that comes later.

```
\newcommand\notenameA{}
\newcommand\notenameB{}
\newcommand\notenameC{}
\newcommand\notenameD{}
\newcommand\notenameE{}
\newcommand\notenameF{}
\newcommand\notenameG{}
```

\printnoteA \printnoteB \printnoteC \printnoteD \printnoteE \printnoteF \printnoteG These control sequences are what the transposition logic actually outputs to denote each scale degree. They can include any LaTeX code that is legal in horizontal mode.

```
\newcommand\printnoteA{}
\newcommand\printnoteB{}
\newcommand\printnoteC{}
\newcommand\printnoteD{}
\newcommand\printnoteE{}
\newcommand\printnoteF{}
\newcommand\printnoteG{}
```

\notenamesin Set the note names used by the input file.

```
\newcommand\notenamesin[7]{%
  \def\notenameA{#1}%
  \def\notenameB{#2}%
  \def\notenameC{#3}%
  \def\notenameD{#4}%
  \def\notenameE{#5}%
  \def\notenameF{#6}%
  \def\notenameG{#7}%
  \SB@convertnotestrue%
}
```

\notenamesout Set the note names that are output by the transposition logic.

```
\newcommand\notenamesout[7]{%
  \def\printnoteA{#1}%
  \def\printnoteB{#2}%
  \def\printnoteC{#3}%
  \def\printnoteD{#4}%
  \def\printnoteE{#5}%
  \def\printnoteF{#6}%
```

```
  \def\printnoteG{#7}%
  \SB@convertnotestrue%
}
```

`\notenames` Set an identical input name and output name for each scale degree.

```
\newcommand\notenames[7]{%
  \notenamesin{#1}{#2}{#3}{#4}{#5}{#6}{#7}%
  \notenamesout{#1}{#2}{#3}{#4}{#5}{#6}{#7}%
  \SB@convertnotesfalse%
}
```

`\alphascale` `\solfedge` Predefine scales for alphabetic names and solfedge names, and set alphabetic scales to be the default.

```
\newcommand\alphascale{\notenames ABCDEFG}
\newcommand\solfedge{\notenames{LA}{SI}{DO}{RE}{MI}{FA}{SOL}}
\alphascale
```

`\ifSB@prefshrps` When a transposed chord falls on a black key, the code must choose which enharmonically equivalent name to give the new chord. (For example, should C transposed by +1 be named C# or D♭?) A heuristic is used to guess which name is most appropriate. The following conditional records whether the current key signature is sharped or flatted according to this heuristic guess.

```
\newif\ifSB@prefshrps
```

`\ifSB@needkey` The first chord seen is usually the best indicator of the key of the song. (Even when the first chord isn't the tonic, it will often be the dominant or subdominant, which usually has the same kind of accidental in its key signatures as the actual key.) This conditional remembers if the current chord is the first one seen in the song, and should therefore be used to guess the key of the song.

```
\newif\ifSB@needkey
```

`\transpose` The `\transpose` macro sets the transposition adjustment factor and informs the transposition logic that the next chord seen will be the first one in the new key.

```
\newcommand\transpose[1]{%
  \advance\SB@transposefactor by#1\relax%
  \SB@cnt\SB@transposefactor%
  \divide\SB@cnt12 %
  \multiply\SB@cnt12 %
  \advance\SB@transposefactor-\SB@cnt%
  \SB@needkeytrue%
}
```

`\capo` Specifying a `\capo` normally just causes a textual note to musicians to be typeset, but if the `transposecapos` option is active, it activates transposition of the chords.

```
\newcommand\capo[1]{%
  \iftranscapos\transpose{#1}\else\musicnote{capo #1}\fi%
}
```

`\prefersharps` `\preferflats` One of these macros is called after the first chord has been seen to register that we're transposing to a key with a sharped or flatted key signature.

```
\newcommand\prefersharps{\SB@prefshrpstrue\SB@needkeyfalse}
\newcommand\preferflats{\SB@prefshrpsfalse\SB@needkeyfalse}
```

`\transposehere` If automatic transposition has been requested, yield the given chord transposed by the requested amount. Otherwise return the given chord verbatim.

```
\newcommand\transposehere[1]{%
  \ifnum\SB@transposefactor=\z@%
    \ifSB@convertnotes%
      \SB@dotranspose{#1}%
      \the\SB@toks%
    \else%
      #1%
    \fi%
  \else%
    \ifSB@convertnotes%
      {\SB@transposefactor\z@%
       \SB@dotranspose{#1}%
       \xdef\SB@tempv{\the\SB@toks}}%
    \else%
      \def\SB@tempv{#1}%
    \fi%
    \SB@dotranspose{#1}%
    \expandafter\trchordformat\expandafter{\SB@tempv}{\the\SB@toks}%
  \fi%
}
```

`\SB@dotranspose` Parse the argument to a chord macro, yielding the transposed equivalent in the `\SB@toks` token register.

```
\newcommand\SB@dotranspose[1]{%
  \SB@toks{}%
  \let\SB@dothis\SB@trmain%
  \SB@trscan#1\SB@trend%
}
```

`\trchordformat` By default, transposing means replacing old chords with new chords in the new key. However, sometimes the user may want to typeset something more sophisticated, like old chords followed by new chords in parentheses so that musicians who use capos and those who don't can play from the same piece of music. Such typesetting is possible by redefining the following macro to something like `#1 (#2)` instead of `#2`.

```
\newcommand\trchordformat[2]{#2}
```

`\SB@trscan` This is the entrypoint to the code that scans over the list of tokens comprising a chord and transposes note names as it goes. Start by peeking ahead at the next symbol without consuming it.

```
\newcommand\SB@trscan{\futurelet\SB@next\SB@dothis}
```

`\SB@trmain` Test to see if the token was a begin-brace, end-brace, or space. These tokens require special treatment because they cannot be accepted as implicit arguments to macros.

```
\newcommand\SB@trmain{%
  \ifx\SB@next\bgroup%
    \let\SB@donext\SB@trgroup%
  \else\ifx\SB@next\egroup%
    \SB@toks\expandafter{\the\SB@toks\egroup}%
    \let\SB@donext\SB@trskip%
  \else\ifcat\noexpand\SB@next\noexpand\@sptoken%
    \SB@appendsp\SB@toks%
    \let\SB@donext\SB@trskip%
  \else%
    \let\SB@donext\SB@trstep%
  \fi\fi\fi%
  \SB@donext%
}
```

`\SB@trgroup` A begin-group brace lies next in the input stream. Consume the entire group as an argument to this macro, and append it, including the begin- and end-group tokens, to the list of tokens processed so far. No transposition takes place within a group; they are copied verbatim because they probably contain macro code.

```
\newcommand\SB@trgroup[1]{%
  \SB@toks\expandafter{\the\SB@toks{#1}}%
  \SB@trscan%
}
```

`\SB@trspace` A space or end-brace lies next in the input stream. It has already been added to the token list, so skip over it.

```
\newcommand\SB@trskip{%
  \afterassignment\SB@trscan%
  \let\SB@next= }
```

`\SB@trstep` A non-grouping token lies next in the input stream. Consume it as an argument to this macro, and then test it to see if it's a note letter or some other recognized item. If so, process it; otherwise just append it to the token list and continue scanning.

```
\newcommand\SB@trstep[1]{%
  \let\SB@donext\SB@trscan%
  \ifcat\noexpand\SB@next A%
    \ifnum\uccode`#1=`#1%
      \def\SB@temp{#1}%
      \let\SB@dothis\SB@trnote%
    \else%
      \SB@toks\expandafter{\the\SB@toks#1}%
    \fi%
  \else\ifx\SB@next\gtab%
    \let\SB@donext\SB@trtab%
```

```
  \else\ifx\SB@next\SB@trend
    \let\SB@donext\relax%
  \else%
    \SB@toks\expandafter{\the\SB@toks#1}%
  \fi\fi\fi%
  \SB@donext%
}
```

`\SB@trnote` We're in the midst of processing a sequence of uppercase letters that might comprise a note name. Check to see if the next token is an accidental (sharp or flat), or yet another letter.

```
\newcommand\SB@trnote{%
  \ifcat\noexpand\SB@next A%
    \let\SB@donext\SB@trnotestep%
  \else\ifnum\SB@transposefactor=\z@%
    \SB@cnt\z@%
    \let\SB@donext\SB@trtrans%
  \else\ifx\SB@next\flt%
    \SB@cnt\m@ne%
    \let\SB@donext\SB@tracc%
  \else\ifx\SB@next\shrp%
    \SB@cnt\@ne%
    \let\SB@donext\SB@tracc%
  \else%
    \SB@cnt\z@%
    \let\SB@donext\SB@trtrans%
  \fi\fi\fi\fi%
  \SB@donext%
}
```

`\SB@trnotestep` The next token is a letter. Consume it and test to see if it is an uppercase letter. If so, add it to the note name being assembled; otherwise reinsert it into the input stream and jump directly to the transposition logic.

```
\newcommand\SB@trnotestep[1]{%
  \ifnum\uccode`#1=`#1%
    \expandafter\def\expandafter\SB@temp\expandafter{\SB@temp#1}%
    \expandafter\SB@trscan%
  \else%
    \SB@cnt\z@%
    \expandafter\SB@trtrans\expandafter#1%
  \fi%
}
```

`\SB@tracc` We've encountered an accidental (sharp or flat) immediately following a note name. Peek ahead at the next token without consuming it, and then jump to the transposition logic. This is done because the transposition logic might need to infer the key signature of the song, and if the next token is an m (for minor), then that information can help.

```
\newcommand\SB@tracc[1]{\futurelet\SB@next\SB@trtrans}
```

\SB@trtrans We've assembled a sequence of capital letters (in \SB@temp) that might comprise a note name to be transposed. If the letters were followed by a \shrp then \SB@cnt is 1; if they were followed by a \flt then it is -1; otherwise it is 0. If the assembled letters turn out to not match any valid note name, then do nothing and return to scanning. Otherwise compute a new transposed name.

```
\newcommand\SB@trtrans{%
  \advance\SB@cnt%
    \ifx\SB@temp\notenameA\z@%
    \else\ifx\SB@temp\notenameB\tw@%
    \else\ifx\SB@temp\notenameC\thr@@%
    \else\ifx\SB@temp\notenameD5 %
    \else\ifx\SB@temp\notenameE7 %
    \else\ifx\SB@temp\notenameF8 %
    \else\ifx\SB@temp\notenameG10 %
    \else-99 \fi\fi\fi\fi\fi\fi\fi%
  \ifnum\SB@cnt<\m@ne%
    \SB@toks\expandafter\expandafter\expandafter{%
      \expandafter\the\expandafter\SB@toks\SB@temp}%
  \else%
    \advance\SB@cnt\SB@transposefactor%
    \ifnum\SB@cnt<\z@\advance\SB@cnt12 \fi%
    \ifnum\SB@cnt>11 \advance\SB@cnt-12 \fi%
    \ifSB@needkey\ifnum\SB@transposefactor=\z@\else\SB@setkeysig\fi\fi%
    \edef\SB@temp{%
      \the\SB@toks%
      \ifSB@prefshrps%
        \ifcase\SB@cnt\printnoteA\or\printnoteA\noexpand\shrp\or%
          \printnoteB\or\printnoteC\or\printnoteC\noexpand\shrp\or%
          \printnoteD\or\printnoteD\noexpand\shrp\or\printnoteE\or%
          \printnoteF\or\printnoteF\noexpand\shrp\or\printnoteG\or%
          \printnoteG\noexpand\shrp\fi%
      \else%
        \ifcase\SB@cnt\printnoteA\or\printnoteB\noexpand\flt\or%
          \printnoteB\or\printnoteC\or\printnoteD\noexpand\flt\or%
          \printnoteD\or\printnoteE\noexpand\flt\or\printnoteE\or%
          \printnoteF\or\printnoteG\noexpand\flt\or\printnoteG\or%
          \printnoteA\noexpand\flt\fi%
      \fi}%
    \SB@toks\expandafter{\SB@temp}%
  \fi%
  \let\SB@dothis\SB@trmain%
  \SB@trscan%
}
```

\SB@setkeysig If this is the first chord of the song, assume that this is the tonic of the key, and select whether to use a sharped or flatted key signature for the rest of the song based on that. Even if this isn't the tonic, it's probably the dominant or sub-dominant, which almost always has a number of sharps or flats similar to the tonic. If the bottom note of the chord turns out to be a black key, we choose the

enharmonic equivalent that is closest to C on the circle of fifths (i.e., the one that has fewest sharps or flats).

```
\newcommand\SB@setkeysig{%
  \global\SB@needkeyfalse%
  \ifcase\SB@cnt%
    \global\SB@prefshrpstrue\or% A
    \global\SB@prefshrpsfalse\or% Bb
    \global\SB@prefshrpstrue\or% B
    \ifx\SB@next m% C
      \global\SB@prefshrpsfalse%
    \else%
      \global\SB@prefshrpstrue%
    \fi\or%
    \global\SB@prefshrpstrue\or% C#
    \ifx\SB@next m% D
      \global\SB@prefshrpsfalse%
    \else%
      \global\SB@prefshrpstrue%
    \fi\or%
    \global\SB@prefshrpsfalse\or% Eb
    \global\SB@prefshrpstrue\or% E
    \global\SB@prefshrpsfalse\or% F
    \global\SB@prefshrpstrue\or% F#
    \ifx\SB@next m% G
      \global\SB@prefshrpsfalse%
    \else%
      \global\SB@prefshrpstrue%
    \fi\or%
    \global\SB@prefshrpsfalse\else% Ab
    \global\SB@needkeytrue% non-chord
  \fi%
}
```

`\SB@trtab` We've been asked to transpose a tablature diagram. We can't do that automatically, but we can at least extract the chord name and just transpose that.

```
\newcommand\SB@trtab{\expandafter\SB@trscan\@firstoftwo}
```

`\SB@trend` The following macro marks the end of chord text to be processed. It should always be consumed and discarded by the chord-scanning logic above, so generate an error if it is ever expanded.

```
\newcommand\SB@trend{%
  \SB@Error{Internal Error: Transposition failed}%
           {This error should not occur.}%
}
```

16.9 Measure Bars

The following code handles the typesetting of measure bars.

\SB@metertop
\SB@meterbot

These macros remember the current numerator and denominator of the meter.

```
\newcommand\SB@metertop{}
\newcommand\SB@meterbot{}
```

\meter

Set the current meter without producing an actual measure bar yet.

```
\newcommand\meter[2]{\gdef\SB@metertop{#1}\gdef\SB@meterbot{#2}}
```

\SB@measuremark

Normally measure bar boxes should be as thin as possible so that they can be slipped into lyrics without making them hard to read. But when two measure bars appear consecutively, they need to be spaced apart more so that they look like two separate lines instead of one thick line. To achieve this, there needs to be a way to pull a vbox off the current list and determine whether or not it is a box that contains a measure bar. The solution is to insert a mark (`\SB@measuremark`) at the top of each measure bar vbox. We can then see if this measure bar immediately follows another measure bar by using `\vsplit` on `\lastbox`.

```
\newcommand\SB@measuremark{SB@IsMeasure}
```

\SB@makembar

Typeset a measure bar. If provided, ⟨*arg1*⟩ is the numerator and ⟨*arg2*⟩ is the denominator of the meter to be rendered above the bar. If those arguments are left blank, render a measure bar without a meter marking.

```
\newcommand\SB@makembar[2]{%
  \ifSB@inverse\else%
    \ifSB@inchorus\else\SB@errmbar\fi%
  \fi%
  \ifhmode%
    \SB@skip\lastskip\unskip%
    \setbox\SB@box\lastbox%
    \copy\SB@box%
    \ifvbox\SB@box%
      \begingroup%
        \setbox\SB@boxii\copy\SB@box%
        \vbadness\@M\vfuzz\maxdimen%
        \setbox\SB@boxii%
          \vsplit\SB@boxii to\maxdimen%
      \endgroup%
      \long\edef\SB@temp{\splitfirstmark}%
      \ifx\SB@temp\SB@measuremark%
        \penalty100\hskip1em%
      \else%
        \penalty100\hskip\SB@skip%
      \fi%
    \else%
      \penalty100\hskip\SB@skip%
    \fi%
  \fi%
  \setbox\SB@box\hbox{\tiny\sffamily{#1}}%
  \setbox\SB@boxii\hbox{\tiny\sffamily{#2}}%
  \ifdim\wd\SB@box>\wd\SB@boxii%
```

```
    \SB@dimen\wd\SB@box\relax%
  \else%
    \SB@dimen\wd\SB@boxii\relax%
  \fi%
  \ifdim\SB@dimen<.5\p@%
    \SB@dimen.5\p@%
  \fi%
  \SB@dimenii\baselineskip%
  \advance\SB@dimenii-2\p@%
  \advance\SB@dimenii-\ht\SB@box%
  \advance\SB@dimenii-\dp\SB@box%
  \advance\SB@dimenii-\ht\SB@boxii%
  \advance\SB@dimenii-\dp\SB@boxii%
  \ifvmode\leavevmode\fi%
  \vbox{%
    \mark{\SB@measuremark}%
    \hbox to\SB@dimen{%
      \hfil%
      \box\SB@box%
      \hfil%
    }%
    \nointerlineskip%
    \hbox to\SB@dimen{%
      \hfil%
      \box\SB@boxii%
      \hfil%
    }%
    \nointerlineskip%
    \hbox to\SB@dimen{%
      \hfil%
      \vrule\@width.5\p@\@height\SB@dimenii%
      \hfil%
    }%
  }%
}
```

`\mbar` The `\mbar` macro invokes `\SB@mbar`, which gets redefined by macros and options that turn measure bars on and off.

```
\newcommand\mbar{\SB@mbar}
```

`\measurebar` Make a measure bar using the most recently defined meter. Then set the meter to nothing so that the next measure bar will not display any meter unless the meter changes.

```
\newcommand\measurebar{%
  \mbar\SB@metertop\SB@meterbot%
  \meter{}{}%
}
```

`\SB@repcolon` Create the colon that preceeds or follows a repeat sign.

```
\newcommand\SB@repcolon{{%
  \usefont{OT1}{cmss}{m}{n}\selectfont%
  \ifchorded%
    \baselineskip.5\SB@dimen%
    \vbox{\hbox{:}\hbox{:}\kern.5\p@}%
  \else%
    \raise.5\p@\hbox{:}%
  \fi%
}}
```

`\lrep` Create a begin-repeat sign.

```
\newcommand\lrep{%
  \SB@dimen\baselineskip%
  \advance\SB@dimen-2\p@%
  \vrule\@width1.5\p@\@height\SB@dimen\@depth\p@%
  \kern1.5\p@%
  \vrule\@width.5\p@\@height\SB@dimen\@depth\p@%
  \SB@repcolon%
}
```

`\rrep` Create an end-repeat sign.

```
\newcommand\rrep{%
  \SB@dimen\baselineskip%
  \advance\SB@dimen-2\p@%
  \SB@repcolon%
  \vrule\@width.5\p@\@height\SB@dimen\@depth\p@%
  \kern1.5\p@%
  \vrule\@width1.5\p@\@height\SB@dimen\@depth\p@%
}
```

16.10 Lyric Scanning

The obvious way to create a chord macro is as a normal macro with two arguments, one for the chord name and one for the lyrics to go under the chord—e.g. `\chord{`⟨*chordname*⟩`}{`⟨*lyric*⟩`}`. However, in practice such a macro is extremely cumbersome and difficult to use. The problem is that in order to use such a macro properly, the user must remember a bunch of complex style rules that govern what part of the lyric text needs to go in the ⟨*lyric*⟩ parameter and what part should be typed after the closing brace. To avoid separating a word from its trailing punctuation, the ⟨*lyric*⟩ parameter must often include punctuation but not certain special punctuation like hyphens, should include the rest of the word but not if there's another chord in the word, should omit measure bars but only if measure bars are being shown, etc. This is way too difficult for the average user.

To avoid this problem, we define chords using a one-argument macro (the argument is the chord name), but with no explicit argument for the lyric part. Instead, the macro scans ahead in the input stream, automatically determining

what portion of the lyric text that follows should be sucked in as an implicit second argument. The following code does this look-ahead scanning.

`\ifSB@wordends` `\ifSB@brokenword` Chord macros must look ahead in the input stream to see if this chord is immediately followed by whitespace or the remainder of a word. If the latter, hyphenation might need to be introduced. These macros keep track of the need for hyphenation, if any.

```
\newif\ifSB@wordends
\newif\ifSB@brokenword
```

`\SB@lyric` Lyrics appearing after a chord are scanned into the following token list register.

```
\newtoks\SB@lyric
```

`\SB@numhyps` Hyphens appearing in lyrics require special treatment. The following counter counts the number of explicit hyphens ending the lyric syllable that follows the current chord.

```
\newcount\SB@numhyps
```

`\SB@lyricnohyp` When a lyric syllable under a chord ends in exactly one hyphen, the following token register is set to be the syllable without the hyphen.

```
\newtoks\SB@lyricnohyp
```

`\SB@lyricbox` `\SB@chordbox` The following two boxes hold the part of the lyric text that is to be typeset under the chord, and the chord text that is to be typeset above.

```
\newbox\SB@lyricbox
\newbox\SB@chordbox
```

`\SB@chbstok` When `\MultiwordChords` is active, the following reserved control sequence remembers the first (space) token not yet included into the `\SB@lyricbox` box.

```
\newcommand\SB@chbstok{}
```

`\SB@setchord` Store the chord text into `\SB@chordbox`.

```
\newcommand\SB@setchord{}
{
  \catcode`^\active
  \catcode`!7
  \gdef\SB@setchord#1{%
    \SB@gettabindtrue\SB@nohattrue%
    \setbox\SB@chordbox\hbox{%
      \unhbox\SB@chordbox%
      \begingroup%
        \ifSB@trackch%
          \def\SB@activehat{\ifmmode!\else\global\SB@nohatfalse\fi}%
        \else%
          \def\SB@activehat{%
            \ifmmode!\else\SB@lop\SB@ctail\SB@toks\the\SB@toks\fi%
          }%
        \fi%
```

```
        \let^\SB@activehat%
        \printchord{%
          \ifSB@firstchord\else\kern.15em\fi%
          \vphantom/%
          \transposehere{#1}%
          \kern.2em%
        }%
      \endgroup%
    }%
    \SB@gettabindfalse%
    \ifSB@trackch\ifSB@nohat%
      \global\SB@creg\expandafter{\the\SB@creg#1\\}%
    \fi\fi%
    \let\SB@noreplay\@firstofone%
  }
}
```

`\SB@outertest` `\SB@@outertest` Macros declared `\outer` are not allowed in arguments, so determining if a token is `\outer` is a delicate process. The following does so by consulting `\meaning`.

```
\newcommand\SB@outertest{}
\edef\SB@outertest#1{%
  \noexpand\SB@@outertest#1%
  \string\outer%
  \noexpand\SB@@outertest%
}
\newcommand\SB@@outertest{}
\expandafter\def\expandafter\SB@@outertest%
\expandafter#\expandafter1\string\outer#2\SB@@outertest{%
  \def\SB@temp{#2}%
  \ifx\SB@temp\@empty\else\SB@testtrue\fi%
}
```

`\SB@UTFtest` `\SB@two` `\SB@three` `\SB@four` `\SB@UTFtester` To support UTF-8 encoded LaTeX source files, we need to be able to identify multibyte characters during the lyric scanning process. Alas, the `utf8.def` file provides no clean way of identifying the macros it defines for this purpose. The best solution seems to be to look for any token named `\UTFviii@...@octets` in the top-level expansion of the macro.

```
\newcommand\SB@UTFtest{}
\edef\SB@UTFtest#1{%
  \noexpand\SB@UTFtester#1%
  \string\UTFviii@zero@octets%
  \noexpand\SB@UTFtester%
}
\begingroup
  \escapechar\m@ne
  \xdef\SB@two{\string\two}
  \xdef\SB@three{\string\three}
  \xdef\SB@four{\string\four}
  \xdef\SB@temp{\string\@octets}
```

```
\endgroup
\edef\SB@temp{##1\string\UTFviii@##2\SB@temp##3}
\expandafter\def\expandafter\SB@UTFtester\SB@temp\SB@UTFtester{%
  \def\SB@temp{#2}%
  \ifx\SB@temp\SB@two%
    \SB@cnt\tw@%
  \else\ifx\SB@temp\SB@three%
    \SB@cnt\thr@@%
  \else\ifx\SB@temp\SB@four%
    \SB@cnt4 %
  \else%
    \SB@cnt\z@%
  \fi\fi\fi%
}
```

\DeclareLyricChar
\DeclareNonLyric
\DeclareNoHyphen
\SB@declare

When scanning the lyric text that follows a chord, it is necessary to distinguish accents and other intra-word macros (which should be included in the under-chord lyric text) from other macros (which should be pushed out away from the text). The following macros allow users to declare a token to be lyric-continuing or lyric-ending.

```
\newcommand\SB@declare[3]{%
  \afterassignment\iffalse\let\SB@next= #3\relax\fi%
  \expandafter\SB@UTFtest\expandafter{\meaning\SB@next}%
  \ifcase\SB@cnt%
    \ifcat\noexpand#3\relax%
      \SB@addNtest\SB@macrotests#1#2#3%
    \else\ifcat\noexpand#3.%
      \SB@addDtest\SB@othertests#1#2#3%
    \else\ifcat\noexpand#3A%
      \SB@addDtest\SB@lettertests#1#2#3%
    \else%
      \SB@addDtest\relax0#2#3%
    \fi\fi\fi%
  \or%
    \SB@addNtest\SB@macrotests#1#2#3%
  \else%
    \SB@addMtest\SB@multitests#1#2{#3}%
  \fi%
}
\newcommand\DeclareLyricChar{\SB@declare\SB@testtrue0}
\newcommand\DeclareNonLyric{%
  \SB@declare\SB@testfalse\SB@testfalse%
}
\newcommand\DeclareNoHyphen{%
  \SB@declare\SB@testfalse\SB@testtrue%
}
```

\SB@lettertests
\SB@macrotests
\SB@multitests
\SB@othertests
\SB@hyphtests

For speed, token tests introduced by `\DeclareLyricChar` and friends are broken out into separate macros based on category codes.

```
\newcommand\SB@lettertests{}
\newcommand\SB@macrotests{}
\newcommand\SB@multitests{}
\newcommand\SB@othertests{}
\newcommand\SB@hyphtests{}
```

The following macros add tests to the test macros defined above. In each, ⟨*arg1*⟩ is the test macro to which the test should be added, ⟨*arg2*⟩ and ⟨*arg3*⟩ is the code to be executed at scanning-time and at hyphenation-time if the test succeeds (or "0" if no action is to be performed), and ⟨*arg4*⟩ is the token to which the currently scanned token should be compared to determine if it matches.

`\SB@addtest`

```
\newcommand\SB@addtest[2]{%
  \expandafter\gdef\expandafter#1\expandafter{#1#2}%
}
```

`\SB@addDtest` A definition-test: The test succeeds if the *definition* at test-time of the next lyric token matches the *definition at test-time* of the control sequence that was given to the `\Declare` macro.

```
\newcommand\SB@addDtest[4]{%
  \ifx0#2\else\SB@addtest#1{\ifx\SB@next#4#2\fi}\fi%
  \ifx0#3\else\SB@addtest\SB@hyphtests{\ifx\SB@next#4#3\fi}\fi%
}
```

`\SB@addNtest` A name-test: The test succeeds if the next token is a non-`\outer` macro or active character and its `\string`ified name matches the `\string`ified name of the control sequence that was given to the `\Declare` macro.

```
\newcommand\SB@addNtest[4]{%
  \ifx0#2\else%
    \SB@addtest#1{%
      \edef\SB@temp{\string#4}\ifx\SB@temp\SB@nextname#2\fi%
    }%
  \fi%
  \ifx0#3\else%
    \SB@addtest\SB@hyphtests{%
      \edef\SB@temp{\string#4}\ifx\SB@temp\SB@nextname#3\fi%
    }%
  \fi%
}
```

`\SB@addMtest` A multibyte-test: The test succeeds if the next lyric token is the beginning of a UTF-8 encoded multibyte character sequence that matches the multibyte sequence given to the `\Declare` macro.

```
\newcommand\SB@addMtest[4]{%
  \ifx0#2\else%
    \SB@addtest#1{\def\SB@temp{#4}\ifx\SB@next\SB@temp#2\fi}%
  \fi%
```

```
  \ifx0#3\else\SB@addtest\SB@hyphtests{%
    \def\SB@temp{#4}\ifx\SB@next\SB@temp#3\fi}%
  \fi%
}
```

The following code declares the common intra-word macros provided by TEX (as listed on p. 52 of The TEXbook) to be lyric-continuing.

```
\DeclareLyricChar\‘
\DeclareLyricChar\’
\DeclareLyricChar\^
\DeclareLyricChar\"
\DeclareLyricChar\~
\DeclareLyricChar\=
\DeclareLyricChar\.
\DeclareLyricChar\u
\DeclareLyricChar\v
\DeclareLyricChar\H
\DeclareLyricChar\t
\DeclareLyricChar\c
\DeclareLyricChar\d
\DeclareLyricChar\b
\DeclareLyricChar\oe
\DeclareLyricChar\OE
\DeclareLyricChar\ae
\DeclareLyricChar\AE
\DeclareLyricChar\aa
\DeclareLyricChar\AA
\DeclareLyricChar\o
\DeclareLyricChar\O
\DeclareLyricChar\l
\DeclareLyricChar\L
\DeclareLyricChar\ss
\DeclareLyricChar\i
\DeclareLyricChar\j
\DeclareLyricChar\/
\DeclareLyricChar\-
\DeclareLyricChar\discretionary
```

We declare `\par` to be lyric-ending without introducing hyphenation. The `\par` macro doesn't actually appear in most verses because we use `\obeylines`, but we include a check for it in case the user says `\par` explicitly somewhere.

```
\DeclareNoHyphen\par
```

`\SB@bracket` This macro gets invoked by the `\[` macro whenever a chord begins. It gets redefined by code that turns chords on and off, so its initial definition doesn't matter.

```
\newcommand\SB@bracket{}
```

`\SB@chord` Begin parsing a chord macro. While parsing the chord name argument, we set some special catcodes so that chord names can use `#` and `&` for sharps and flats.

```
\newcommand\SB@chord{\SB@begincname\SB@@chord}
```

\SB@begincname
\SB@endcname

While parsing a chord name, certain characters such as # and & are temporarily set active so that they can be used as abbreviations for sharps and flats. To accomplish this, \SB@begincname must always be invoked before any macro whose argument is a chord name, and \SB@endcname must be invoked at the start of the body of any macro whose argument is a chord name. To aid in debugging, we also temporarily set end-of-line characters and chord macros outer. This will cause TEX to halt with a runaway argument error on the correct source line if the user forgets to type a closing end-brace (a common typo). Colon characters are also set non-active to avoid a conflict between the Babel French package and the \gtab macro.

```
\newcommand\SB@begincname{}
{\catcode`\^^M\active
 \gdef\SB@begincname{%
   \begingroup%
     \catcode`##\active\catcode`&\active%
     \catcode`:12\relax%
     \catcode`\^^M\active\SB@outer\def^^M{}%
     \SB@outer\def\[{}%
     \chordlocals%
  }
}
\newcommand\SB@endcname{}
\let\SB@endcname\endgroup
```

\SB@nbsp

Non-breaking spaces (~) should be treated as spaces by the lyric-scanner code that follows. Although ~ is usually an active character that creates a non-breaking space, some packages (e.g., the Babel package) redefine it to produce accents. To distinguish the real ~ from redefined ~, we need to create a macro whose definition is the non-breaking space definition normally assigned to ~.

```
\newcommand\SB@nbsp{}
\def\SB@nbsp{\nobreakspace{}}
```

\SB@firstchord

The following conditional is true when the current chord is the first chord in a sequence of one or more chord macros.

```
\newif\ifSB@firstchord\SB@firstchordtrue
```

\SB@@chord

Process the chord and then begin scanning the implicit lyric argument. (This is the main entrypoint to the lyric-scanner code.)

```
\newcommand*\SB@@chord{}
\def\SB@@chord#1]{%
  \SB@endcname%
  \ifSB@firstchord%
    \setbox\SB@lyricbox\hbox{\kern\SB@tabindent}%
    \global\SB@tabindent\z@%
    \SB@lyric{}%
    \SB@numhyps\z@%
    \SB@spcinit%
    \setbox\SB@chordbox\box\voidb@x%
```

```
  \fi%
  \SB@setchord{#1}%
  \SB@firstchordfalse%
  \let\SB@dothis\SB@chstart%
  \SB@chscan%
}
```

`\MultiwordChords` `\SB@spcinit` The `\SB@spcinit` macro is invoked at the beginning of the lyric scanning process. By default it does nothing, but if `\MultiwordChords` is invoked, it initializes the lyric-scanner state to process spaces as part of lyrics.

```
\newcommand\SB@spcinit{}
\newcommand\MultiwordChords{%
  \def\SB@spcinit{%
    \let\SB@chdone\SB@chlyrdone%
    \def\SB@chimpspace{\let\SB@donext\SB@chdone}%
    \def\SB@chexpspace{\let\SB@donext\SB@chdone}%
    \let\SB@chespace\SB@chendspace%
  }%
}
```

`\SB@chscan` `\SB@chmain` This is the main loop of the lyric-scanner. Peek ahead at the next token without consuming it, then execute a loop body based on the current state (`\SB@dothis`), and finally go to the next iteration (`\SB@donext`).

```
\newcommand\SB@chscan{%
  \let\SB@nextname\relax%
  \futurelet\SB@next\SB@chmain%
}
\newcommand\SB@chmain{\SB@dothis\SB@donext}
```

Warning: In the lyric-scanner macros that follow, `\SB@next` might be a macro declared `\outer`. This means that it must *never* be passed as an argument to a macro and it must never explicitly appear in any untaken branch of a conditional. If it does, the TEX parser will complain of a runaway argument when it tries to skip over an `\outer` macro while consuming tokens at high speed.

`\SB@chstart` We begin lyric-scanning with two special cases: (1) If the chord macro is immediately followed by another chord macro with no intervening whitespace, drop out of the lyric scanner and reenter it when the second macro is parsed. The chord texts will get concatenated together above the lyric that follows. (2) If the chord macro is immediately followed by one or more quote tokens, then consume them all and output them *before* the chord. This causes the chord to sit above the actual word instead of the left-quote or left-double-quote symbol, which looks better.

```
\newcommand\SB@chstart{%
  \ifx\SB@next\[%
    \let\SB@donext\relax%
  \else\ifx\SB@next\SB@activehat%
    \let\SB@donext\relax%
  \else\ifx\SB@next\ch%
```

```
    \let\SB@donext\relax%
  \else\ifx\SB@next\mch%
    \let\SB@donext\relax%
  \else\ifx\SB@next`%
    \let\SB@donext\SB@chstep%
  \else\ifx\SB@next'%
    \let\SB@donext\SB@chstep%
  \else\ifx\SB@next"%
    \let\SB@donext\SB@chstep%
  \else%
    \the\SB@lyric%
    \SB@lyric{}%
    \SB@firstchordtrue%
    \let\SB@dothis\SB@chnorm%
    \SB@chnorm%
  \fi\fi\fi\fi\fi\fi\fi%
}
```

`\SB@chnorm` First, check to see if the lyric token is a letter. Since that's the most common case, we do this check first for speed.

```
\newcommand\SB@chnorm{%
  \ifcat\noexpand\SB@next A%
    \SB@testtrue\SB@lettertests%
    \ifSB@test%
      \SB@chespace\let\SB@donext\SB@chstep%
    \else%
      \let\SB@donext\SB@chdone%
    \fi%
  \else%
    \SB@chtrymacro%
  \fi%
}
```

`\SB@chtrymacro` Next, check to see if it's a macro or active character. We do these checks next because these are the only cases when the token might be `\outer`. Once we eliminate that ugly possibility, we can write the rest of the code without having to worry about putting `\SB@next` in places where `\outer` tokens are illegal.

```
\newcommand\SB@chtrymacro{%
  \ifcat\noexpand\SB@next\relax%
    \SB@chmacro%
  \else%
    \SB@chother%
  \fi%
}
```

`\SB@chother` The token is not a letter, macro, or active character. The only other cases of interest are spaces, braces, and hyphens. If it's one of those, take the appropriate action; otherwise end the lyric here. Since we've eliminated the possibility of

macros and active characters, we can be sure that the token isn't `\outer` at this point.

```
\newcommand\SB@chother{%
  \ifcat\noexpand\SB@next\noexpand\@sptoken%
    \SB@chexpspace%
  \else\ifcat\noexpand\SB@next\noexpand\bgroup%
    \SB@chespace\let\SB@donext\SB@chbgroup%
  \else\ifcat\noexpand\SB@next\noexpand\egroup%
    \SB@chespace\let\SB@donext\SB@chegroup%
  \else\ifx\SB@next-%
    \SB@numhyps\@ne\relax%
    \SB@lyricnohyp\expandafter{\the\SB@lyric}%
    \let\SB@dothis\SB@chhyph%
    \SB@chespace\let\SB@donext\SB@chstep%
  \else\ifcat\noexpand\SB@next.%
    \SB@testtrue\SB@othertests%
    \ifSB@test%
      \SB@chespace\let\SB@donext\SB@chstep%
    \else%
      \let\SB@donext\SB@chdone%
    \fi%
  \else%
    \SB@chespace\let\SB@donext\SB@chstep%
  \fi\fi\fi\fi\fi%
}
```

`\SB@chmacro` The lyric-scanner has encountered a macro or active character. If it's `\outer`, it should never be used in an argument, so stop here.

```
\newcommand\SB@chmacro{%
  \SB@testfalse%
  \expandafter\SB@outertest\expandafter{\meaning\SB@next}%
  \ifSB@test%
    \let\SB@donext\SB@chdone%
  \else%
    \let\SB@donext\SB@chgetname%
  \fi%
}
```

`\SB@chgetname` We've encountered a non-`\outer` macro or active character. Use `\string` to get its name, but insert the token back into the input stream since we haven't decided whether to consume it yet.

```
\newcommand\SB@chgetname[1]{%
  \edef\SB@nextname{\string#1}%
  \SB@@chmacro\SB@donext#1%
}
```

`\SB@@chmacro` The lyric-scanner has encountered a non-`\outer` macro or active character. Its `\string`ified name has been stored in `\SB@nextname`. Test to see if it's a known macro or the beginning of a multibyte-encoded international character. If the

former, dispatch some macro-specific code to handle it. If the latter, grab the full multibyte sequence and include it in the lyric.

```
\newcommand\SB@@chmacro{%
  \ifx\SB@next\SB@activehat%
    \let\SB@donext\SB@chdone%
  \else\ifx\SB@next\SB@par%
    \let\SB@donext\SB@chdone%
  \else\ifx\SB@next\measurebar%
    \SB@chmbar%
  \else\ifx\SB@next\mbar%
    \SB@chmbar%
  \else\ifx\SB@next\ch%
    \SB@chespace\let\SB@donext\SB@chlig%
  \else\ifx\SB@next\mch%
    \SB@chespace\let\SB@donext\SB@mchlig%
  \else\ifx\SB@next\ %
    \SB@chimpspace%
  \else\ifx\SB@next\SB@nbsp%
    \SB@chimpspace%
  \else%
    \expandafter\SB@UTFtest\expandafter{\meaning\SB@next}%
    \ifcase\SB@cnt\SB@chothermac%
    \or\or\SB@chespace\let\SB@donext\SB@chsteptwo%
    \or\SB@chespace\let\SB@donext\SB@chstepthree%
    \or\SB@chespace\let\SB@donext\SB@chstepfour\fi%
  \fi\fi\fi\fi\fi\fi\fi\fi%
}
```

`\SB@chothermac` The lyric-scanner has encountered a macro or active character that is not `\outer`, not a known macro that requires special treatment, and not a multibyte international character. First, check the macro's name (stored in `\SB@nextname`) to see if it begins with a non-escape character. If so, it's probably an accenting or punctuation character made active by the `inputenc` or `babel` packages. Most such characters should be included in the lyric, so include it by default; otherwise exclude it by default. The user can override the defaults using `\DeclareLyricChar` and friends.

```
\newcommand\SB@chothermac{%
  \SB@testfalse%
  \afterassignment\iffalse%
  \SB@cnt\expandafter`\SB@nextname x\fi%
  \ifnum\the\catcode\SB@cnt=\z@\else\SB@testtrue\fi%
  \SB@macrotests%
  \ifSB@test%
    \SB@chespace\let\SB@donext\SB@chstep%
  \else%
    \let\SB@donext\SB@chdone%
  \fi%
}
```

\SB@chstep
\SB@chsteptwo
\SB@chstepthree
\SB@chstepfour
\SB@chmulti
\SB@chmstop

We've encountered one or more tokens that should be included in the lyric text. (More than one means we've encountered a multibyte encoding of an international character.) Consume them (as arguments to this macro) and add them to the list of tokens we've already consumed.

```
\newcommand\SB@chstep[1]{%
  \SB@lyric\expandafter{\the\SB@lyric#1}%
  \SB@chscan%
}
\newcommand\SB@chsteptwo[2]{\SB@chmulti{#1#2}}
\newcommand\SB@chstepthree[3]{\SB@chmulti{#1#2#3}}
\newcommand\SB@chstepfour[4]{\SB@chmulti{#1#2#3#4}}
\newcommand\SB@chmulti[1]{%
  \def\SB@next{#1}%
  \let\SB@nextname\relax%
  \SB@testtrue\SB@multitests%
  \ifSB@test%
    \SB@lyric\expandafter{\the\SB@lyric#1}%
    \expandafter\SB@chscan%
  \else%
    \expandafter\SB@chmstop%
  \fi%
}
\newcommand\SB@chmstop{\expandafter\SB@chdone\SB@next}
```

\SB@chhyph

We've encountered a hyphen. Continue to digest hyphens, but terminate as soon as we see anything else.

```
\newcommand\SB@chhyph{%
  \ifx\SB@next-%
    \advance\SB@numhyps\@ne\relax%
    \let\SB@donext\SB@chstep%
  \else%
    \let\SB@donext\SB@chdone%
  \fi%
}
```

\SB@chimpspace
\SB@chexpspace

We've encountered an implicit or explicit space. Normally this just ends the lyric, but if `\MultiwordChords` is active, these macros both get redefined to process the space.

```
\newcommand\SB@chimpspace{\let\SB@donext\SB@chdone}
\newcommand\SB@chexpspace{\let\SB@donext\SB@chdone}
```

\SB@chespace
\SB@chendspace

The `\SB@chespace` macro gets invoked by the lyric-scanner just before a non-space token is about to be accepted as part of an under-chord lyric. Normally it does nothing; however, if `\MultiwordChords` is active, it gets redefined to do one of three things: (1) Initially it is set equal to `\SB@chendspace` so that if the very first token following the chord macro is not a space, the lyric-scanner macros are redefined to process any future spaces encountered. Otherwise the very first token is a space, and the lyric ends immediately. (2) While scanning

non-space lyric tokens, it is set to nothing, since no special action needs to be taken until we encounter a sequence of one or more spaces. (3) When a space token is encountered (but not the very first token after the chord macro), it is set equal to `\SB@chendspace` again so that `\SB@chendspace` is invoked once the sequence of one or more space tokens is finished.

```
\newcommand\SB@chespace{}
\newcommand\SB@chendspace{%
  \let\SB@chdone\SB@chlyrdone%
  \def\SB@chexpspace{\SB@chbspace\SB@chexpspace}%
  \def\SB@chimpspace{\SB@chbspace\SB@chimpspace}%
  \def\SB@chespace{}%
}
```

`\SB@chbspace` `\SB@chgetspace`

The `\SB@chbspace` macro gets invoked when `\MultiwordChords` is active and the lyric-scanner has encountered a space token that was immediately preceded by a non-space token. Before processing the space, we add all lyrics seen so far to the `\SB@lyricbox` and check its width. If we've seen enough lyrics to match or exceed the width of the chord, a space stops the lyric-scanning process. (This is important because it minimizes the size of the chord box, providing as many line breakpoints as possible to the paragraph-formatter.)

Otherwise we begin scanning space tokens without adding them to the lyric until we see what the next non-space token is. If the next non-space token would have ended the lyric anyway, roll back and end the lyric here, reinserting the space tokens back into the token stream. If the next non-space token would have been included in the lyric, the lyric-scanner proceeds as normal.

```
\newcommand\SB@chbspace{%
  \setbox\SB@lyricbox\hbox{%
    \unhbox\SB@lyricbox%
    \the\SB@lyric%
  }%
  \SB@lyric{}%
  \ifdim\wd\SB@lyricbox<\wd\SB@chordbox%
    \let\SB@chbstok= \SB@next%
    \def\SB@chexpspace{\let\SB@donext\SB@chgetspace}%
    \def\SB@chimpspace{\let\SB@donext\SB@chstep}%
    \let\SB@chespace\SB@chendspace%
    \let\SB@chdone\SB@chspcdone%
  \else%
    \def\SB@chimpspace{\let\SB@donext\SB@chdone}%
    \def\SB@chexpspace{\let\SB@donext\SB@chdone}%
  \fi%
}
\newcommand\SB@chgetspace{%
  \SB@appendsp\SB@lyric%
  \let\SB@nextname\relax%
  \afterassignment\SB@chscan%
  \let\SB@next= }
```

\SB@chmbar We've encountered a measure bar. Either ignore it or end the lyric text, depending on whether measure bars are being displayed.

```
\newcommand\SB@chmbar{%
  \ifmeasures%
    \let\SB@donext\SB@chdone%
  \else%
    \SB@chespace\let\SB@donext\SB@chstep%
  \fi%
}
```

\SB@chbgroup We've encountered a begin-group brace. Consume the entire group that it begins, and add it to the list of tokens including the begin and end group tokens.

```
\newcommand\SB@chbgroup[1]{%
  \SB@lyric\expandafter{\the\SB@lyric{#1}}%
  \SB@chscan%
}
```

\SB@chegroup
\SB@chegrpscan
\SB@chegrpmacro
\SB@chegrpouter
\SB@chegrpname
\SB@chegrpdone
We've encountered an end-group brace whose matching begin-group brace must have come before the chord macro itself. This forcibly ends the lyric text. Before stopping, we must set \SB@next to the token following the brace and \SB@nextname to its \stringified name so that \SB@emitchord will know whether to add hyphenation. Therefore, we temporarily consume the end-group brace, then scan the next token without consuming it, and finally reinsert the end-group brace and stop.

```
\newcommand\SB@chegroup{%
  \let\SB@nextname\relax%
  \afterassignment\SB@chegrpscan%
  \let\SB@next= }
\newcommand\SB@chegrpscan{%
  \futurelet\SB@next\SB@chegrpmacro%
}
\newcommand\SB@chegrpmacro{%
  \ifcat\noexpand\SB@next\relax%
    \expandafter\SB@chegrpouter%
  \else%
    \expandafter\SB@chegrpdone%
  \fi%
}
\newcommand\SB@chegrpouter{%
  \SB@testfalse%
  \expandafter\SB@outertest\expandafter{\meaning\SB@next}%
  \ifSB@test%
    \expandafter\SB@chegrpdone%
  \else%
    \expandafter\SB@chegrpname%
  \fi%
}
\newcommand\SB@chegrpname[1]{%
  \edef\SB@nextname{\string#1}%
```

```
  \SB@chegrpdone#1%
}
\newcommand\SB@chegrpdone{\SB@chdone\egroup}
```

`\SB@chlig` `\SB@mchlig` We've encountered a `\ch` chord-over-ligature macro, or an `\mch` measurebar-and-chord-over-ligature macro. Consume it and all of its arguments, and load them into some registers for future processing. (Part of the ligature might fall into this lyric text or might not, depending on if we decide to add hyphenation.) Then end the lyric text here.

```
\newcommand\SB@chlig[5]{%
  \gdef\SB@ligpre{{#3}}%
  \gdef\SB@ligpost{\[#2]{#4}}%
  \gdef\SB@ligfull{\[\SB@noreplay{\hphantom{{\lyricfont#3}}}#2]{#5}}%
  \SB@chdone%
}
\newcommand\SB@mchlig[5]{%
  \SB@lyric\expandafter{\the\SB@lyric#3}%
  \let\SB@next\measurebar%
  \edef\SB@nextname{\string\measurebar}%
  \gdef\SB@ligpost{\measurebar\[#2]{#4}}%
  \gdef\SB@ligfull{\measurebar\[#2]{#4}}%
  \SB@chdone%
}
```

`\SB@chdone` `\SB@chlyrdone` `\SB@chspcdone` The `\SB@chdone` macro is invoked when we've decided to end the lyric text (usually because we've encountered a non-lyric token). Normally this expands to `\SB@chlyrdone`, which adds any uncontributed lyric material to the `\SB@lyricbox` and jumps to the main chord formatting macro. However, if `\MultiwordChords` is active and if the lyric ended with a sequence of one or more space tokens, then we instead reinsert the space tokens into the token stream without contributing them to the `\SB@lyricbox`.

```
\newcommand\SB@chlyrdone{%
  \setbox\SB@lyricbox\hbox{%
    \unhbox\SB@lyricbox%
    \ifnum\SB@numhyps=\@ne%
      \the\SB@lyricnohyp%
    \else%
      \the\SB@lyric%
    \fi%
  }%
  \SB@emitchord%
}
\newcommand\SB@chspcdone{%
  \let\SB@nextname\relax%
  \let\SB@next= \SB@chbstok%
  \expandafter\SB@emitchord\the\SB@lyric%
}
\newcommand\SB@chdone{}
\let\SB@chdone\SB@chlyrdone
```

\SB@ligpre
\SB@ligpost
\SB@ligfull

The following three macros record arguments passed to a \ch macro that concludes the lyric text of the \[] macro currently being processed.

```
\newcommand\SB@ligpre{}
\newcommand\SB@ligpost{}
\newcommand\SB@ligfull{}
```

\SB@clearlig

Clear all ligature-chord registers.

```
\newcommand\SB@clearlig{%
  \gdef\SB@ligpre{}%
  \gdef\SB@ligpost{}%
  \gdef\SB@ligfull{}%
}
```

16.11 Chords

\SB@emitchord

The \SB@emitchord macro does the actual work of typesetting chord text over lyric text, introducing appropriate hyphenation when necessary. We begin by consulting \SB@next, which should have been set by the lyric-scanning code in §16.10 to the token that immediately follows the lyric under this chord, to determine whether the lyric text ends on a word boundary.

```
\newcommand\SB@emitchord{%
  \ifSB@inverse\else\ifSB@inchorus\else\SB@errchord\fi\fi%
  \SB@testfalse%
  \ifcat\noexpand\SB@next\noexpand\@sptoken\SB@testtrue\fi%
  \ifcat\noexpand\SB@next.\SB@testtrue\fi%
  \ifx\SB@next\SB@par\SB@testtrue\fi%
  \ifx\SB@next\egroup\SB@testtrue\fi%
  \ifx\SB@next\endgroup\SB@testtrue\fi%
  \SB@hyphtests%
  \ifSB@test\SB@wordendstrue\else\SB@wordendsfalse\fi%
```

Next, compare the width of the lyric to the width of the chord to determine if hyphenation might be necessary. The original lyric text might have ended in a string of one or more explicit hyphens, enumerated by \SB@numhyps. If it ended in exactly one, the lyric-scanning code suppresses that hyphen so that we can here add a new hyphen that floats out away from the word when the chord above it is long. If it ended in more than one (e.g., the encoding of an en- or em-dash) then the lyric-scanner leaves it alone; we must not add any hyphenation or float the dash away from the word.

There is also code here to insert a penalty that discourages linebreaking immediately before lyricless chords. Beginning a wrapped line with a lyricless chord is undesirable because it makes it look as though the wrapped line is extra-indented (due to the empty lyric space below the chord). It should therefore happen only as a last resort.

```
  \SB@dimen\wd\SB@chordbox%
  \ifvmode\leavevmode\fi%
  \SB@brokenwordfalse%
  \ifdim\wd\SB@lyricbox>\z@%
```

```
    \ifdim\SB@dimen>\wd\SB@lyricbox%
      \ifSB@wordends\else\SB@brokenwordtrue\fi%
    \fi%
  \else%
    \SB@skip\lastskip%
    \unskip\penalty200\hskip\SB@skip%
  \fi%
  \ifnum\SB@numhyps>\z@%
    \ifnum\SB@numhyps>\@ne%
      \SB@brokenwordfalse%
    \else%
      \SB@brokenwordtrue%
    \fi%
  \fi%
```

If lyrics are suppressed on this line (e.g., by using `\nolyrics`), then just typeset the chord text on the natural baseline.

```
  \SB@testfalse%
  \ifnolyrics\ifdim\wd\SB@lyricbox=\z@\SB@testtrue\fi\fi%
  \ifSB@test%
    \unhbox\SB@chordbox%
    \gdef\SB@temp{\expandafter\SB@clearlig\SB@ligfull}%
  \else%
```

Otherwise, typeset the chord above the lyric on a double-height line.

```
    \vbox{\baselineskip\f@size\p@\kern2\p@%
      \ifSB@brokenword%
        \global\setbox\SB@lyricbox\hbox{%
          \unhbox\SB@lyricbox%
          \SB@ligpre%
        }%
        \SB@maxmin\SB@dimen<{\wd\SB@lyricbox}%
        \advance\SB@dimen.5em%
        \hbox to\SB@dimen{\unhbox\SB@chordbox\hfil}%
        \kern-2\p@%
        \hbox to\SB@dimen{%
          \unhcopy\SB@lyricbox\hfil\char\hyphenchar\font\hfil%
        }%
        \global\SB@cnt\@m%
        \gdef\SB@temp{\expandafter\SB@clearlig\SB@ligpost}%
      \else%
        \hbox{\unhbox\SB@chordbox\hfil}%
        \kern-2\p@%
        \hbox{%
          \unhcopy\SB@lyricbox%
          \global\SB@cnt\spacefactor%
          \hfil%
        }%
        \gdef\SB@temp{\expandafter\SB@clearlig\SB@ligfull}%
      \fi%
    }%
```

If the chord is lyricless, inhibit a linebreak immediately following it. This prevents sequences of lyricless chords (which often end lines) from being wrapped in the middle, which looks very unsightly and makes them difficult to read. If the chord has a lyric but it doesn't end on a word boundary, insert an appropriate penalty to prevent linebreaking without hyphenation. Also preserve the spacefactor in this case, which allows LaTeX to fine-tune the spacing between consecutive characters in the word that contains the chord.

```
    \ifSB@wordends%
      \ifdim\wd\SB@lyricbox>\z@\else\nobreak\fi%
    \else%
      \penalty%
        \ifnum\SB@numhyps>\z@\exhyphenpenalty%
        \else\ifSB@brokenword\hyphenpenalty%
        \else\@M\fi\fi%
      \spacefactor\SB@cnt%
    \fi%
  \fi%
```

Finally, end the macro with some code that handles the special case that this chord is immediately followed by a chord-over-ligature macro. The code above sets `\SB@temp` to the portion of the ligature that should come after this chord but before the chord that tops the ligature. This text must be inserted here.

```
  \SB@temp%
}
```

`\SB@accidental` Typeset an accidental symbol as a superscript within a chord. Since chord names are often in italics but math symbols like sharp and flat are not, we need to do some kerning adjustments before and after the accidental to position it as if it were italicized. The pre-adjustment is just a simple italic correction using `\/`. The post-adjustment is based on the current font's slant-per-point metric.

```
\newcommand\SB@accidental[1]{{%
  \/%
  \m@th#1%
  \SB@dimen-\fontdimen\@ne\font%
  \advance\SB@dimen.088142\p@%
  \ifdim\SB@dimen<\z@%
    \kern\f@size\SB@dimen%
  \fi%
}}
```

`\sharpsymbol` `\flatsymbol` When changing the sharp or flat symbol, change these macros rather than changing `\shrp` or `\flt`. This will ensure that other shortcuts like `#` and `&` will reflect your change.

```
\newcommand\sharpsymbol{\ensuremath{^\#}}
\newcommand\flatsymbol{\raise.5ex\hbox{{\SB@flatsize$\flat$}}}
```

`\shrp` `\flt` These macros typeset sharp and flat symbols.

```
\newcommand\shrp{\SB@accidental\sharpsymbol}
\newcommand\flt{\SB@accidental\flatsymbol}
```

\DeclareFlatSize The \flat math symbol is too small for properly typesetting chord names. (Its size was designed for staff notation not textual chord names.) The correct size for the symbol should be approximately 30% larger than the current superscript size, or 90% of the base font size b. However, simply computing $0.9b$ does not work well because most fonts do not render well in arbitrary sizes. To solve the problem, we must therefore choose an appropriate size individually for each possible base font size b. This is the solution adopted by the rest of LaTeX for such things. For example, LaTeX's \DeclareMathSizes macro defines an appropriate superscript size for each possible base font size. The macro below creates a similar macro that that defines an appropriate flat-symbol size for each possible base font size.

```
\newcommand\DeclareFlatSize[2]{%
  \expandafter\xdef\csname SB@flatsize@#1\endcsname{#2}%
}
\DeclareFlatSize\@vpt\@vpt
\DeclareFlatSize\@vipt\@vipt
\DeclareFlatSize\@viipt\@vipt
\DeclareFlatSize\@viiipt\@viipt
\DeclareFlatSize\@ixpt\@viiipt
\DeclareFlatSize\@xpt\@ixpt
\DeclareFlatSize\@xipt\@xpt
\DeclareFlatSize\@xiipt\@xipt
\DeclareFlatSize\@xivpt\@xiipt
\DeclareFlatSize\@xviipt\@xivpt
\DeclareFlatSize\@xxpt\@xviipt
\DeclareFlatSize\@xxvpt\@xxpt
```

\SB@flatsize Select the correct flat symbol size based on the current font size.

```
\newcommand\SB@flatsize{%
  \@ifundefined{SB@flatsize@\f@size}{}{%
    \expandafter\fontsize%
      \csname SB@flatsize@\f@size\endcsname\f@baselineskip%
    \selectfont%
  }%
}
```

In the following code, the \ch, \mch, \[, and ^ macros are each defined to be a single macro that then expands to the real definition. This is necessary because the top-level definitions of each must stay the same in order to allow the lyric-scanning code to uniquely identify them, yet their internal definitions must be redefined by code that turns chords and/or measure bars on and off. Such code redefines \SB@ch, \SB@mch, \SB@bracket, and \SB@rechord to effect a change of mode without touching the top-level definitions.

\ch \SB@ch \SB@ch@on \SB@@ch \SB@@@ch \SB@ch@off The \ch macro puts a chord atop a ligature without breaking the ligature. Normally this just means placing the chord midway over the unbroken ligature (ignoring the third argument completely). However, when a previous chord macro encounters it while scanning ahead in the input stream to parse its lyric, the \ch macro itself is not actually expanded at all. Instead, the chord macro scans ahead,

spots the \ch macro, gobbles it, and then steals its arguments, breaking the ligature with hyphenation. Thus, the \ch macro is only actually expanded when the ligature shouldn't be broken.

```
\newcommand\ch{\SB@ch}
\newcommand\SB@ch{}
\newcommand\SB@ch@on{\SB@begincname\SB@@ch}
\newcommand*\SB@@ch[1]{\SB@endcname\SB@@@ch{#1}}
\newcommand*\SB@@@ch[4]{\[\SB@noreplay{\hphantom{#2}}#1]#4}
\newcommand*\SB@ch@off[4]{#4}
```

\mch The \mch macro is like \ch except that it also introduces a measure bar.

\SB@mch \SB@mch@m \SB@mch@on \SB@@mch \SB@@@mch

```
\newcommand\mch{\SB@mch}
\newcommand\SB@mch{}
\newcommand*\SB@mch@m[4]{#2\measurebar#3}
\newcommand\SB@mch@on{\SB@begincname\SB@@mch}
\newcommand*\SB@@mch[1]{\SB@endcname\SB@@@mch{#1}}
\newcommand*\SB@@@mch[4]{#2\measurebar\[#1]#3}
```

\SB@activehat This macro must always contain the current definition of the ^ chord-replay active character, in order for the lyric scanner to properly identify it and insert proper hyphenation when necessary.

```
\newcommand\SB@activehat{%
  \ifmmode^\else\expandafter\SB@rechord\fi%
}
```

\SB@loadactives It's cumbersome to have to type \shrp, \flt, and \mbar every time you want a sharp, flat, or measure bar, so within verses and choruses we allow the hash, ampersand, and pipe symbols to perform the those functions too. It's also cumbersome to have to type something like \chord{Am}{lyric} to produce each chord. As an easier alternative, we here define \[Am] to typeset chords.

```
\newcommand\SB@loadactives{}
{
  \catcode`&\active
  \catcode`#\active
  \catcode`|\active
  \catcode`^\active
  \global\let&\flt
  \global\let#\shrp
  \global\let|\measurebar
  \global\let^\SB@activehat
  \gdef\SB@loadactives{%
    \catcode`^\ifchorded\active\else9 \fi%
    \catcode`|\ifmeasures\active\else9 \fi%
    \def\[{\SB@bracket}%
  }
}
```

16.12 Chord Replaying

\SB@trackch While inside a verse where the chord history is being remembered for future verses, \SB@trackch is true.

```
\newif\ifSB@trackch
```

\SB@cr@ Reserve token registers to record a history of the chords seen in a verse.

```
\newtoks\SB@cr@
\newtoks\SB@ctail
```

\SB@creg The following control sequence equals the token register being memorized into or replayed from.

```
\newcommand\SB@creg{}
```

\newchords Allocate a new chord-replay register to hold memorized chords.

```
\newcommand\newchords[1]{%
  \@ifundefined{SB@cr@#1}{%
    \expandafter\newtoks\csname SB@cr@#1\endcsname%
    \global\csname SB@cr@#1\endcsname{\\}%
  }{\SB@errdup{#1}}%
}
```

\memorize \SB@memorize Saying \memorize throws out any previously memorized list of chords and starts memorizing chords until the end of the current verse or chorus.

```
\newcommand\memorize{%
  \@ifnextchar[\SB@memorize{\SB@memorize[]}%
}
\newcommand\SB@memorize{}
\def\SB@memorize[#1]{%
  \@ifundefined{SB@cr@#1}{\SB@errreg{#1}}{%
    \SB@trackchtrue%
    \global\expandafter\let\expandafter\SB@creg%
      \csname SB@cr@#1\endcsname%
    \global\SB@creg{\\}%
  }%
}
```

\replay \SB@replay \SB@@replay Saying \replay stops any memorization and begins replaying memorized chords.

```
\newcommand\replay{\@ifnextchar[\SB@replay\SB@@replay}
\newcommand\SB@replay{}
\def\SB@replay[#1]{%
  \@ifundefined{SB@cr@#1}{\SB@errreg{#1}}{%
    \SB@trackchfalse%
    \global\expandafter\let\expandafter\SB@creg%
      \csname SB@cr@#1\endcsname%
    \global\SB@ctail\SB@creg%
  }%
}
\newcommand\SB@@replay{%
```

```
  \SB@trackchfalse%
  \global\SB@ctail\SB@creg%
}
```

`\SB@rechord` `\SB@@rechord` Replay the same chord that was in a previous verse.

```
\newcommand\SB@rechord{}
\newcommand\SB@@rechord{%
  \SB@ifempty\SB@ctail{%
    \SB@errreplay%
    \SB@toks{}%
    \let\SB@donext\@gobble%
  }{%
    \SB@lop\SB@ctail\SB@toks%
    \let\SB@donext\SB@chord%
    \let\SB@noreplay\@gobble%
  }%
  \expandafter\SB@donext\the\SB@toks]%
}
```

`\ifSB@nohat` The `\ifSB@nohat` conditional is set to false when a chord macro contains a `^` in its argument. This suppresses the recording mechanism momentarily so that replays will skip this chord.

```
\newif\ifSB@nohat
```

`\SB@noreplay` Sometimes material must be added to a chord but omitted when the chord is replayed. We accomplish this by enclosing such material in `\SB@noreplay` macros, which are set to `\@gobble` just before a replay and reset to `\@firstofone` at other times.

```
\newcommand\SB@noreplay{}
\let\SB@noreplay\@firstofone
```

16.13 Guitar Tablatures

The song book software not only supports chord names alone, but can also typeset guitar tablature diagrams. The macros for producing these diagrams are found here.

`\SB@fretwidth` Set the width of each vertical string in the tablature diagram.

```
\newlength\SB@fretwidth
\setlength\SB@fretwidth{6\p@}
```

`\SB@fretnum` Typeset a fret number to appear to the left of the diagram.

```
\newcommand\SB@fretnum[1]{{%
  \sffamily\fontsize\@xpt\@xpt\selectfont#1%
}}
```

`\SB@onfret` Typeset one string of one fret with ⟨*arg1*⟩ typeset overtop of it (usually a dot or nothing at all).

```
\newcommand\SB@onfret[1]{%
  \rlap{\hbox to\SB@fretwidth{\hfil\vrule\@height6\p@\hfil}}%
  \hbox to\SB@fretwidth{\hfil#1\hfil}%
}
```

`\SB@atopfret` Typeset material (given by ⟨*arg1*⟩) to be placed above a string in the tablature diagram.

```
\newcommand\SB@atopfret[1]{%
  \hbox to\SB@fretwidth{\hfil#1\hfil}%
}
```

`\SB@fretbar` Typeset a horizontal fret bar of width `\SB@dimen`.

```
\newcommand\SB@fretbar{%
  \nointerlineskip%
  \hbox to\SB@dimen{%
    \advance\SB@dimen-\SB@fretwidth%
    \advance\SB@dimen.4\p@%
    \hfil%
    \vrule\@width\SB@dimen\@height.4\p@\@depth\z@%
    \hfil%
  }%
  \nointerlineskip%
}
```

`\SB@topempty` Above a string in a tablature diagram there can be nothing, an ×, or an ∘.

`\SB@topX` `\SB@topO`

```
\newcommand\SB@topempty{\SB@atopfret\relax}
\newcommand\SB@topX{\SB@atopfret{%
  \hbox{%
    \kern-.2\p@%
    \fontencoding{OMS}\fontfamily{cmsy}%
    \fontseries{m}\fontshape{n}%
    \fontsize\@viipt\@viipt\selectfont\char\tw@%
    \kern-.2\p@%
  }%
}}
\newcommand\SB@topO{\SB@atopfret{%
  \vrule\@width\z@\@height4.3333\p@\@depth.8333\p@%
  \lower.74\p@\hbox{%
    \fontencoding{OMS}\fontfamily{cmsy}%
    \fontseries{m}\fontshape{n}%
    \fontsize\@xpt\@xpt\selectfont\char14%
  }%
}}
```

\SB@fretempty
\SB@frethit

On a string in a fret diagram there can be nothing or a filled circle.

```
\newcommand\SB@fretempty{\SB@onfret\relax}
\newcommand\SB@frethit{\SB@onfret{%
  \hbox{%
    \fontencoding{OMS}\fontfamily{cmsy}%
    \fontseries{m}\fontshape{n}%
    \fontsize\@xiipt\@xiipt\selectfont\char15%
  }%
}}
```

\SB@finger

If we're including fingering info in the tablature diagram, then below each string there might be a number.

```
\newcommand\SB@finger[1]{%
  \SB@atopfret{\sffamily\fontsize\@vipt\@vipt\selectfont#1}%
}
```

\ifSB@gettabind
\SB@tabindent

Lyrics under tablature diagrams look odd if they aren't aligned with the leftmost string of the diagram. To accomplish this, the following two macros record the amount by which a lyric under this tablature diagram must be indented to position it properly.

```
\newif\ifSB@gettabind\SB@gettabindfalse
\newdimen\SB@tabindent
```

\SB@targfret
\SB@targstr
\SB@targfing

Reserve some macro names in which to store the three pieces of the second argument to the \gtab macro. The first is for the fret number, the second is for the ⟨*strings*⟩ info, and the last is for the ⟨*fingering*⟩ info.

```
\newcommand\SB@targfret{}
\newcommand\SB@targstr{}
\newcommand\SB@targfing{}
```

In general \gtab macros often appear inside chord macros, which means that their arguments have already been scanned by the time the \gtab macro itself is expanded. This means that catcodes cannot be reassigned (without resorting to ε-TEX).

We therefore adopt the alternative strategy of converting each token in the ⟨*strings*⟩ and ⟨*fingering*⟩ arguments of a \gtab macro into a control sequence (using \csname). We can then temporarily assign meanings to those control sequences and replay the arguments to achieve various effects.

\SB@csify
\SB@@csify

Convert all tokens in the first argument to control sequences and store the resulting sequence into the macro given by the first argument. Store the length in tokens into counter register \SB@cnt.

```
\newcommand\SB@csify[2]{%
  \SB@toks{}%
  \SB@cnt\z@%
  \SB@@csify#2\SB@@csify%
  \edef#1{\the\SB@toks}%
}
```

```
\newcommand\SB@@csify[1]{%
  \ifx#1\SB@@csify\else%
    \advance\SB@cnt\@ne%
    \SB@toks\expandafter{\the\SB@toks\csname#1\endcsname}%
    \expandafter\SB@@csify%
  \fi%
}
```

\SB@gttop
\SB@gtinit
\SB@gtinc
\SB@gtset

Different meanings are assigned to digits, `X`'s, and `O`'s depending on whether we are currently typesetting the material overtop the diagram, the interior of the diagram, or the fingering numbers below the diagram. These meanings are set by `\SB@gttop`, `\SB@gtinit` & `\SB@gtinc`, and `\SB@gtset`, respectively.

```
\newcommand\SB@gttop{%
  \let\X\SB@topX\let\0\SB@topO\let\O\0\let\1\SB@topempty%
  \let\2\1\let\3\1\let\4\1\let\5\1%
  \let\6\1\let\7\1\let\8\1\let\9\1%
}
\newcommand\SB@gtinit{%
  \let\X\SB@fretempty\let\0\X\let\O\X\let\1\SB@frethit%
  \let\2\X\let\3\X\let\4\X\let\5\X%
  \let\6\X\let\7\X\let\8\X\let\9\X%
}
\newcommand\SB@gtinc{%
  \let\9\8\let\8\7\let\7\6\let\6\5\let\5\4%
  \let\4\3\let\3\2\let\2\1\let\1\0%
}
\newcommand\SB@gtset[2]{%
  \let\X#1\let\0\X\let\O\X%
  \def\1{#21}\def\2{#22}\def\3{#23}%
  \def\4{#24}\def\5{#25}\def\6{#26}%
  \def\7{#27}\def\8{#28}\def\9{#29}%
}
```

\SB@gtmax

To compute the height of the tablature diagram, we must identify the maximum fret number in the ⟨*strings*⟩ argument. This is accomplished by using the following macro in combination with `\SB@gtset` above.

```
\newcommand\SB@gtmax[1]{\ifnum\SB@cnt<#1\SB@cnt#1\fi}
```

\gtab
\SB@gtab

A `\gtab` macro begins by setting catcodes suitable for parsing a chord name as its first argument. If the macro is already inside a chord macro this has no effect, but if not this allows tokens like `#` and `&` to be used for sharp and flat. Colon is reset to a non-active character while processing the second argument to avoid a potential conflict with Babel French.

```
\newcommand\gtab{\SB@begincname\SB@gtab}
\newcommand*\SB@gtab[1]{%
  \SB@endcname%
  \begingroup%
    \catcode`:12\relax%
    \SB@@gtab{#1}%
```

```
}
```

`\SB@@gtab` Typeset a full tablature diagram. Text ⟨*arg1*⟩ is a chord name placed above the diagram. Text ⟨*arg2*⟩ consists of: (1) an optional fret number placed to the left of the diagram; (2) a sequence of tokens, each of which can be `X` (to place an × above the string), `0` or `O` (to place an ∘ above the string), or one of `1` through `9` (to place a filled circle on that string at the fret of the given number); and (3) an optional colon which, if present, precedes another sequence of tokens, each of which is either `0` (no fingering information for that string), or one of `1` through `4` (to place the given number under that string).

```
\newcommand*\SB@@gtab[2]{%
  \endgroup%
  \let\SB@targfret\@empty%
  \let\SB@targstr\@empty%
  \let\SB@targfing\@empty%
  \SB@tabargs#2:::\SB@tabargs%
  \ifx\SB@targstr\@empty%
    \def\SB@targstr{\0\0\0\0\0\0}%
  \fi%
  \ifvmode\leavevmode\fi%
  \vbox{%
    \normalfont\normalsize%
    \setbox\SB@box\hbox{%
      \thinspace{\printchord{#1\strut}}\thinspace%
    }%
    \setbox\SB@boxii\hbox{\SB@fretnum{\SB@targfret}}%
    \setbox\SB@boxiii\hbox{{\SB@gttop\SB@targstr}}%
    \hsize\wd\SB@box%
    \ifSB@gettabind%
      \global\SB@tabindent\wd\SB@boxii%
      \global\advance\SB@tabindent.5\SB@fretwidth%
      \global\advance\SB@tabindent-.5\p@%
    \fi%
    \SB@dimen\wd\SB@boxii%
    \advance\SB@dimen\wd\SB@boxiii%
    \ifdim\hsize<\SB@dimen%
      \hsize\SB@dimen%
    \else\ifSB@gettabind%
      \SB@dimenii\hsize%
      \advance\SB@dimenii-\SB@dimen%
      \divide\SB@dimenii\tw@%
      \global\advance\SB@tabindent\SB@dimenii%
    \fi\fi%
    \hbox to\hsize{\hfil\unhbox\SB@box\hfil}%
    \kern-\p@\nointerlineskip%
    \hbox to\hsize{%
      \hfil%
      \vtop{\kern\p@\kern2\p@\box\SB@boxii}%
      \vtop{%
```

```
        \SB@dimen\wd\SB@boxiii%
        \box\SB@boxiii%
        \SB@cnt\minfrets%
        \SB@gtset\relax\SB@gtmax\SB@targstr%
        \SB@gtinit%
        \loop%
          \SB@fretbar\hbox{\SB@targstr}%
          \advance\SB@cnt\m@ne%
        \ifnum\SB@cnt>\z@\SB@gtinc\repeat%
        \SB@fretbar%
        \ifx\SB@targsfing\@empty\else%
          \kern1.5\p@%
          \SB@gtset\SB@topempty\SB@finger%
          \hbox{\SB@targfing}%
        \fi%
      }%
      \hfil%
    }%
    \kern3\p@%
  }%
  \SB@gettabindfalse%
}
```

`\SB@tabargs` `\SB@ctoken` Break the second argument to a `\gtab` macro into three sub-arguments. The possible forms are: (a) ⟨*strings*⟩, (b) ⟨*fret*⟩:⟨*strings*⟩, (c) ⟨*strings*⟩:⟨*fingering*⟩, or (d) ⟨*fret*⟩:⟨*strings*⟩:⟨*fingering*⟩.

```
\newcommand\SB@ctoken{} \def\SB@ctoken{:}
\newcommand\SB@tabargs{}
\def\SB@tabargs#1:#2:#3:#4\SB@tabargs{%
  \def\SB@temp{#4}%
  \ifx\SB@temp\@empty%
    \SB@csify\SB@targstr{#1}%
  \else\ifx\SB@temp\SB@ctoken%
    \SB@csify\SB@targstr{#1}%
    \ifnum\SB@cnt>\@ne%
      \SB@cntii\SB@cnt%
      \SB@csify\SB@targfing{#2}%
      \SB@cnt\SB@cntii%
    \else%
      \def\SB@targfret{#1}%
      \SB@csify\SB@targstr{#2}%
    \fi%
  \else%
    \def\SB@targfret{#1}%
    \SB@csify\SB@targfing{#3}%
    \SB@csify\SB@targstr{#2}%
  \fi\fi%
}
```

16.14 Book Sectioning

The following macros divide the song book into distinct sections, each with different headers, different song numbering styles, different indexes, etc.

`\songchapter` Format the chapter header for a chapter in a song book. By default, chapter headers on a song book omit the chapter number, but do include an entry in the pdf index or table of contents. Thus, the chapter has a number; it's just not displayed at the start of the chapter.

```
\newcommand\songchapter{%
  \let\SB@temp\@seccntformat%
  \def\@seccntformat##1{}%
  \@startsection{chapter}{0}{\z@}%
    {3.5ex\@plus1ex\@minus.2ex}%
    {.4ex\let\@seccntformat\SB@temp}%
    {\sffamily\bfseries\LARGE\centering}%
}
```

`\songsection` Format the section header for a section in a song book. This is the same as for chapter headers except at the section level.

```
\newcommand\songsection{%
  \let\SB@temp\@seccntformat%
  \def\@seccntformat##1{}%
  \@startsection{section}{1}{\z@}%
    {3.5ex\@plus1ex\@minus.2ex}%
    {.4ex\let\@seccntformat\SB@temp}%
    {\sffamily\bfseries\LARGE\centering}%
}
```

`songs` Begin and end a book section. The argument is a list of indexes with which to associate songs in this section.

```
\newenvironment{songs}[1]{%
  \ifSB@songsenv\SB@errnse\fi%
  \gdef\SB@indexlist{#1}%
  \SB@chkidxlst%
  \stepcounter{SB@songsnum}%
  \setcounter{songnum}{1}%
  \let\SB@sgroup\@empty%
  \ifinner\else\ifdim\pagetotal>\z@%
    \null\nointerlineskip%
  \fi\fi%
  \songcolumns\SB@numcols%
  \SB@songsenvtrue%
}{%
  \commitsongs%
  \global\let\SB@indexlist\@empty%
  \ifinner\else\clearpage\fi%
  \SB@songsenvfalse%
}
```

Each songs section needs a unique number to aid in hyperlinking.

```
\newcounter{SB@songsnum}
```

16.15 Index Generation

The following macros generate the various types of indexes. At present there are four types:

1. A "large" index has a separate section for each capital letter and is printed in two columns.
2. A "small" index has only a single column, centered, and has no sections.
3. A "scripture" index has three columns and each entry has a comma-separated list of references.
4. An "author" index is like a large index except in bold and without the sectioning.

"Large" and "small" indexes will be chosen automatically based on the number of index entries when building a song index. The other two types are designated by the user.

As is typical of LaTeX indexes, generation of song book indexes requires two passes of document compilation. During the first pass, data files are generated with song titles, authors, and scripture references. An external program is then used to produce LaTeX source files from those data files. During the second pass of document compilation, those source files are imported to typeset all the indexes and display them in the document.

`\SB@indexlist` This macro records the comma-separated list of the indexes associated with the current book section.

```
\newcommand\SB@indexlist{}
```

`\SB@newindex` Define a new title, author, or scripture index.

`\SB@openindex`

```
\newcommand\SB@newindex[4]{%
  \expandafter\newcommand\csname SB@idxfilename@#3\endcsname{#4}%
  \expandafter\newcommand\csname SB@idxsel@#3\endcsname[3]{###1}%
  \expandafter\newcommand\csname SB@idxref@#3\endcsname{\thesongnum}%
  \AtBeginDocument{\SB@openindex{#2}{#3}{#4}}%
}
\newcommand\SB@openindex[3]{%
  \ifSB@genindexes\begingroup%
    \newwrite\SB@theindex%
    \immediate\openout\SB@theindex=#3.sxd%
    #1%
    \global\expandafter\let\csname SB@index@#2\endcsname\SB@theindex%
  \endgroup\fi%
}
```

\newindex
\SB@titleinit

Define a new title index. This causes a new file named ⟨*arg2*⟩`.sxd` to be created as the document processes. The identifier associated with this new index, and that will appear in any book sectioning commands that use it, is ⟨*arg1*⟩.

```
\newcommand\newindex{\SB@newindex1\SB@titleinit}
\@onlypreamble\newindex
\newcommand\SB@titleinit{%
  \immediate\write\SB@theindex{TITLE INDEX DATA FILE}%
}
```

\newscripindex
\SB@scripinit

Define a new scripture index. This is exactly like `\newindex` except that scripture references are added to ⟨*arg2*⟩`.sxd` instead of titles.

```
\newcommand\newscripindex{\SB@newindex2\SB@scripinit}
\@onlypreamble\newscripindex
\newcommand\SB@scripinit{%
  \immediate\write\SB@theindex{SCRIPTURE INDEX DATA FILE}%
}
```

\newauthorindex
\SB@authinit

Define a new author index. This is exactly like `\newindex` except that author info will be written to ⟨*arg2*⟩.sxd instead of title info.

```
\newcommand\newauthorindex{\SB@newindex3\SB@authinit}
\@onlypreamble\newauthorindex
\newcommand\SB@authinit{%
  \immediate\write\SB@theindex{AUTHOR INDEX DATA FILE}%
}
```

\indexsongsas

The following macro allows the user to change how songs are indexed on the right side of index entries. By default, the song's number is listed.

```
\newcommand\indexsongsas[1]{%
  \@ifundefined{SB@idxref@#1}%
    {\SB@errnoidx{#1}\@gobble}%
    {\expandafter\renewcommand\csname SB@idxref@#1\endcsname}%
}
```

\SB@percent

Assign a literal % character to `\SB@percent` in order to output it to `index.sxd` files.

```
\newcommand\SB@percent{}
{\catcode`\%=12\gdef\SB@percent{%}}
```

\authsepword
\authbyword
\authignoreword
\titleprefixword
\SB@idxcmd

The `songidx` index-generation program understands several different directives that each dictate various aspects of how index entries are parsed, sorted, and displayed. Such directives should typically appear at the start of the `.sxd` file just after the header line that identifies the type of index. To allow the user to specify these directives within the `.tex` source file, we here define a set of preamble macros that add directives to a token list. The token list is eventually committed at the end of the preamble once the index `.sxd` files are opened for output.

```
\newcommand\SB@idxcmd[2]{%
  \expandafter\gdef\expandafter#1\expandafter{%
    #1\immediate\write\SB@theindex{\SB@percent#2}%
```

```
  }%
}
\newcommand\authsepword[1]{\SB@idxcmd\SB@authinit{sep #1}}
\@onlypreamble\authsepword
\newcommand\authbyword[1]{\SB@idxcmd\SB@authinit{after #1}}
\@onlypreamble\authbyword
\newcommand\authignoreword[1]{\SB@idxcmd\SB@authinit{ignore #1}}
\@onlypreamble\authignoreword
\newcommand\titleprefixword[1]{\SB@idxcmd\SB@titleinit{prefix #1}}
\@onlypreamble\titleprefixword
```

`\SB@songwrites` Song index data cannot be written to the index files immediately as soon as it is declared by the document author. It must be deferred twice: First it is queued in the following box register until the the song box is finally constructed. When the song box is constructed, the queued material is injected into the top of the box as non-immediate write whatsits that are only expanded and written when the box finally reaches the output stream. This allows the index data to depend on things like the current page number, which is only decided at the last minute by the LaTeX output routines.

```
\newbox\SB@songwrites
```

`\SB@addtoindex` Queue data ⟨*arg2*⟩ associated with the current song for eventual writing to the index whose output stream is given by ⟨*arg1*⟩.

```
\newcommand\SB@addtoindex[2]{%
  \global\setbox\SB@songwrites\vbox{%
    \unvbox\SB@songwrites%
    \ifSB@genindexes%
      \protected@write{\csname SB@index@#1\endcsname}{}{#2}%
      \protected@write{\csname SB@index@#1\endcsname}{}%
        {\csname SB@idxref@#1\endcsname}%
      \protected@write{\csname SB@index@#1\endcsname}{}{%
        song\theSB@songsnum-\thesongnum.%
        \ifnum\c@section=\z@1\else2\fi}%
    \fi%
  }%
}
```

`\SB@addtoindexes` Add ⟨*arg1*⟩ to all title indexes, ⟨*arg2*⟩ to all scripture indexes, and ⟨*arg3*⟩ to all author indexes.

```
\newcommand\SB@addtoindexes[3]{%
  \@for\SB@temp:=\SB@indexlist\do{%
    \SB@addtoindex\SB@temp%
      {\csname SB@idxsel@\SB@temp\endcsname{#1}{#2}{#3}}%
  }%
}
```

\SB@addtotitles Add ⟨*arg1*⟩ to all title indexes, but leave other indexes unaffected.

```
\newcommand\SB@addtotitles[1]{%
  \@for\SB@temp:=\SB@indexlist\do{%
    \csname SB@idxsel@\SB@temp\endcsname%
      {\SB@addtoindex\SB@temp{#1}}{}{}%
  }%
}
```

\SB@chkidxlst Check the current list of indexes and flag an error if any are undefined.

```
\newcommand\SB@chkidxlst{%
  \let\SB@temp\SB@indexlist%
  \let\SB@indexlist\@empty%
  \@for\SB@tempii:=\SB@temp\do{%
    \@ifundefined{SB@idxsel@\SB@tempii}{\SB@errnoidx\SB@tempii}{%
      \ifx\SB@indexlist\@empty%
        \SB@toks\expandafter{\SB@tempii}%
      \else%
        \SB@toks\expandafter\expandafter\expandafter{%
          \expandafter\SB@indexlist\expandafter,\SB@tempii}%
      \fi%
      \edef\SB@indexlist{\the\SB@toks}%
    }%
  }%
}
```

\indexentry
\SB@idxentry
\SB@@idxentry

\SB@addtoindexes will be called automatically for each song in a section. However, \indexentry may be called by the user in order to add an alternative index entry for the given song. Usually this is done to index the song by its first line or some other memorable line in a chorus or verse somewhere.

```
\newcommand\indexentry{\@ifnextchar[{\SB@idxentry*}{\SB@@idxentry*}}
\newcommand\SB@idxentry{}
\def\SB@idxentry#1[#2]#3{{%
  \def\SB@indexlist{#2}%
  \SB@chkidxlst%
  \SB@addtoindexes{#1#3}{#3}{#3}%
}}
\newcommand\SB@@idxentry[2]{\SB@addtotitles{#1#2}}
```

\indextitleentry \indextitleentry may be used to add an alternate title for the song to the index. (The only difference between the effects of \indexentry and \indextitleentry is that the latter are italicized in the rendered index and the former are not.)

```
\newcommand\indextitleentry{%
  \@ifnextchar[{\SB@idxentry{}}{\SB@@idxentry{}}%
}
```

\SB@idxtitlebox Define a box to hold the index title.

```
\newbox\SB@idxtitlebox
```

`\SB@idxlineskip` Set the spacing between lines in an index.

```
\newcommand\SB@idxlineskip[1]{%
  \vskip#1\p@\@plus#1\p@\@minus#1\p@%
}
```

When rendering an index entry $X \ldots Y$ that is too long to fit on one physical line, we must break text X and/or Y up into multiple lines. Text X should be typeset as a left-justified paragraph with a right margin of about 2em; however, it's final line must not be so long that it cannot fit even the first item of list Y. Text Y should be typeset as a right-justified paragraph whose first line begins on the last line of X. However, breaking Y up the way paragraphs are normally broken up doesn't work well because that causes most of Y to be crammed into the first few lines, leaving the last line very short. This looks strange and is hard to read. It looks much better to instead break Y up in such a way that the portion of Y that is placed on each line is of approximately equal width (subject to the constraint that we don't want to introduce any more lines than are necessary). This makes it visually clear that all of these lines are associated with X. The following code performs the width computations that do this horizontal-balancing of text.

`\SB@ellipspread` Typeset an index entry of the form $X \ldots Y$. In the common case, the entire entry fits on one line so we just typeset it in the usual way. If it doesn't fit on one line, we call `\SB@balancerows` for a more sophisticated treatment.

```
\newcommand\SB@ellipspread[2]{%
  \begingroup%
    \SB@dimen\z@%
    \def\SB@temp{#1}%
    \SB@toks{#2}%
    \setbox\SB@box\hbox{{%
      \SB@temp%
      \leaders\hbox to.5em{\hss.\hss}\hskip2em\@plus1fil%
      {\the\SB@toks}%
    }}%
    \ifdim\wd\SB@box>\hsize%
      \SB@balancerows%
    \else%
      \hbox to\hsize{\unhbox\SB@box}\par%
    \fi%
  \endgroup%
}
```

`\SB@balancerows` Typeset an index entry of the form $X \ldots Y$ that doesn't fit on one line, where X is the content of macro `\SB@temp` and Y is the content of token register `\SB@toks`.

First, we must pre-compute the width w_1 of the final line of X when X is typeset as a left-justified paragraph, storing it in `\SB@dimenii`. This is necessary because in order to force TEX to typeset the first line of Y at some chosen width w_2, we must insert leaders of width $c - w_1 - w_2$ into the paragraph between X and Y, where c is the column width.

Computing this width w_1 is a bit tricky. We must tell TEX that the last line of X must not be so long that it does not even have room for the first item of Y. Thus, we must strip off the first item of Y and add it (or a non-breaking space of equivalent width) to the end of X to typeset the paragraph. Then we use `\lastbox` to pull off the final line and check its width.

```
\newcommand\SB@balancerows{%
  \edef\SB@tempii{\the\SB@toks}%
  \setbox\SB@box\vbox{%
    \SB@toks\expandafter{\expandafter\\\the\SB@toks\\}%
    \SB@lop\SB@toks\SB@toks%
    \settowidth\SB@dimen{\the\SB@toks}%
    \advance\SB@dimen-.5em%
    \leftskip.5cm%
    {\hbadness\@M\hfuzz\maxdimen%
     \hskip-.5cm\relax\SB@temp\unskip\nobreak%
     \hskip\SB@dimen\nobreak%
     \rightskip2em\@plus1fil\par}%
    \setbox\SB@box\lastbox%
    \setbox\SB@box\hbox{%
      \unhbox\SB@box%
      \unskip\unskip\unpenalty%
      \unpenalty\unskip\unpenalty%
    }%
    \expandafter%
  }%
  \expandafter\SB@dimenii\the\wd\SB@box\relax%
```

Next, compute the smallest width w_2 such that the index entry text produced by `\SB@multiline` with `\SB@dimen`$=w_2$ has no more lines than with `\SB@dimen` set to the maximum available width for the right-hand side. This effectively horizontal-balances the right-hand side of the index entry text, making all lines of Y roughly equal in width without introducing any extra lines.

```
  \SB@dimen\hsize%
  \advance\SB@dimen-.5cm%
  \setbox\SB@box\vbox{%
    \SB@multiline{\hbadness\@M\hfuzz\maxdimen}%
  }%
  \SB@dimeniii.5\SB@dimen%
  \SB@dimeniv\SB@dimeniii%
  \loop%
    \SB@dimeniv.5\SB@dimeniv%
    \setbox\SB@boxii\vbox{%
      \SB@dimen\SB@dimeniii%
      \SB@multiline{\hbadness\@M\hfuzz\maxdimen}%
    }%
    \ifnum\SB@cnt<\@M%
      \ifdim\ht\SB@boxii>\ht\SB@box%
        \advance\SB@dimeniii\SB@dimeniv%
      \else%
```

```
        \SB@dimen\SB@dimeniii%
        \advance\SB@dimeniii-\SB@dimeniv%
      \fi%
    \else%
      \advance\SB@dimeniii\SB@dimeniv%
    \fi%
  \ifdim\SB@dimeniv>2\p@\repeat%
  \setbox\SB@box\box\voidb@x%
  \setbox\SB@boxii\box\voidb@x%
```

Finally, typeset the results based on the quantities computed above.

```
  \SB@multiline\relax%
}
```

`\SB@multiline` Create a paragraph containing text $X \ldots Y$ where X is the content of `\SB@temp`, Y is the content of `\SB@tempii`, and Y is restricted to width `\SB@dimen` (but may span multiple lines of that width). Dimen register `\SB@dimenii` must be set with the expected width of the final line of X. The first argument contains any parameter definitions that should be in effect when X is processed.

Note that the expansion of `\SB@tempii`, which may contain `\SB@idxitemsep`, depends on `\SB@dimen`. Therefore, the redefinition of `\SB@dimen` at the start of this macro must not be removed!

```
\newcommand\SB@multiline[1]{%
  \begingroup%
    \SB@dimen-\SB@dimen%
    \advance\SB@dimen\hsize%
    \SB@dimenii-\SB@dimenii%
    \advance\SB@dimenii\SB@dimen%
    {#1\hskip-.5cm\relax\SB@temp\unskip\nobreak%
     \SB@maxmin\SB@dimenii<{1.5em}%
     \leftskip.5cm\rightskip2em\@plus1fil%
     \interlinepenalty\@M%
     \leaders\hbox to.5em{\hss.\hss}\hskip\SB@dimenii\@plus1fill%
     \nobreak{\SB@tempii\kern-2em}%
     \par\global\SB@cnt\badness}%
  \endgroup%
}%
```

`\SB@idxitemsep` If text Y in index entry $X \ldots Y$ has multiple items in a list, those items should be separated by `\\` macros instead of by commas. The `\\` macro will be assigned the definition of `\SB@idxitemsep` during index generation, which produces the comma along with the complex spacing required if Y ends up being broken into multiple lines. In particular, it forces each wrapped line of Y to be right-justified with left margin at least `\SB@dimen`.

```
\newcommand\SB@idxitemsep{%
  ,\kern-2em\penalty-8\hskip2.33em\@minus.11em%
  \hskip-\SB@dimen\@plus-1fill%
  \vadjust{}\nobreak%
  \hskip\SB@dimen\@plus1fill\relax%
```

```
}
```

The following set of macros and environments are intended for use in the `.sbx` files that are automatically generated by an index-generating program; they shouldn't normally appear in the user's `.tex` or `.sbd` files directly. However, they are named as exported macros (no `@` symbols) since they are used outside the package code and are therefore not stricly internal.

`idxblock` Some indexes are divided into blocks (e.g., one for each letter of the alphabet or one for each book of the bible). Each such block should be enclosed between `\begin{idxblock}{X}` and `\end{idxblock}` lines, where X is the title of the block. The actual definition of the `idxblock` environment is set within the initialization code for each type of index (below).

```
\newenvironment{idxblock}[1]{}{}
```

`\idxentry` `\idxaltentry` Within each `idxblock` environment there should be a series of `\idxentry` and/or `\idxaltentry` macros, one for each line of the index. Again, the exact definitions of these macros will vary between index types.

```
\newcommand\idxentry[2]{}
\newcommand\idxaltentry[2]{}
```

`SB@lgidx` `SB@smidx` Some indexes actually have two definitions for each `idxblock` environment—one for use when there are few enough entries to permit a small style index, and another for use in a large style index. These macros will be redefined appropriately within the initialization code for each type of index.

```
\newenvironment{SB@lgidx}[1]{}{}
\newenvironment{SB@smidx}[1]{}{}
```

`\SB@idxsetup` Set various parameters for a multicolumn index environment.

```
\newcommand\SB@idxsetup[1]{%
  \hsize\SB@colwidth%
  \parskip\z@skip\parfillskip\z@skip\parindent\z@%
  \baselineskip\f@size\p@\@plus\p@\@minus\p@%
  \lineskiplimit\z@\lineskip\p@\@plus\p@\@minus\p@%
  \hyphenpenalty\@M\exhyphenpenalty\@M%
}
```

`\SB@makeidxcolumn` Break off enough material from `\SB@box` to create one column of the index.

```
\newcommand\SB@makeidxcolumn[1]{%
  \ifdim\ht\SB@box=\z@%
    \hskip\hsize\relax%
  \else%
    \splittopskip\z@skip\splitmaxdepth\maxdepth%
    \vsplit\SB@box to\SB@dimen%
    \global\setbox\SB@box\vbox{%
      \SB@idxsetup{#1}%
      \splitbotmark%
      \unvbox\SB@box%
```

```
    }%
  \fi%
}
```

`\SB@oneidxpage` Construct one full page of the index. The definition of `\SB@oneidxpage` is generated dynamically based on the type of index and number of columns.

```
\newcommand\SB@oneidxpage{}
```

`\SB@displayindex` Create an index with title ⟨*arg2*⟩ and with ⟨*arg1*⟩ columns (must be a literal constant). Input the index contents from external file ⟨*arg3*⟩, which is expected to be a TEX file.

```
\newcommand\SB@displayindex[3]{%
  \ifsongindexes\begingroup%
    \SB@colwidth\hsize%
    \advance\SB@colwidth-#1\columnsep%
    \advance\SB@colwidth\columnsep%
    \divide\SB@colwidth#1%
    \setbox\SB@idxtitlebox\vbox{%
      \let\SB@temp\songsection%
      \ifx\chapter\undefined\else%
        \ifx\chapter\relax\else%
          \let\SB@temp\songchapter%
        \fi%
      \fi%
      \SB@temp{#2}%
    }%
```

The `.sbx` index file might not exist (e.g., if this is the first pass through the TEX compiler). If it exists, first try typesetting its content as a small index (one column, centered, with no divisions).

```
    \IfFileExists{\csname SB@idxfilename@#3\endcsname.sbx}{%
      \ifx\hyperlink\undefined\let\hyperlink\@secondoftwo\fi%
      \ifx\hyperlink\relax\let\hyperlink\@secondoftwo\fi%
      \global\setbox\SB@box\vbox{%
        \null%
        \vfil%
        \unvcopy\SB@idxtitlebox%
        \vskip.5in\@minus.3in\relax%
        \hbox to\hsize{%
          \hfil%
          \vbox{%
            \hsize\SB@colwidth%
            \renewenvironment{idxblock}[1]%
              {\begin{SB@smidx}{####1}}{\end{SB@smidx}}%
            \let\\\SB@idxitemsep%
            \input{\csname SB@idxfilename@#3\endcsname.sbx}%
          }%
          \hfil%
        }%
        \vskip\z@\@plus2fil\relax%
```

```
    }%
```

Test whether the resulting small index fits within one page. If not, re-typeset it as a large index.

```
    {\vbadness\@M\vfuzz\maxdimen%
     \splitmaxdepth\maxdepth\splittopskip\z@skip%
     \global\setbox\SB@boxii\vsplit\SB@box to\textheight}%
    \ifvoid\SB@box%
      \box\SB@boxii%
    \else%
      \global\setbox\SB@box\vbox{%
        \renewenvironment{idxblock}[1]%
          {\begin{SB@lgidx}{####1}}{\end{SB@lgidx}}%
        \let\\\SB@idxitemsep%
        \SB@idxsetup{#1}%
        \input{\csname SB@idxfilename@#3\endcsname.sbx}%
        \unskip%
      }%
      \SB@toks{\SB@makeidxcolumn{#1}}%
      \SB@cnt#1\relax%
      \loop\ifnum\SB@cnt>\@ne%
        \SB@toks\expandafter{\the\SB@toks%
          \kern\columnsep\SB@makeidxcolumn{#1}}%
        \advance\SB@cnt\m@ne%
      \repeat%
      \edef\SB@oneidxpage{\the\SB@toks}%
      \unvbox\SB@idxtitlebox%
      \vskip.2in\relax%
      \nointerlineskip%
      \null%
      \nointerlineskip%
      \SB@cnt\vbadness\vbadness\@M%
      \SB@dimenii\vfuzz\vfuzz\maxdimen%
      \loop%
        \SB@dimen\textheight%
        \ifinner\else\kern\z@\advance\SB@dimen-\pagetotal\fi%
        \global\setbox\SB@boxii\copy\SB@box%
        \global\setbox\SB@boxiii\hbox{\SB@oneidxpage}%
        \ifdim\ht\SB@box>\z@%
          \box\SB@boxiii%
          \vfil\break%
      \repeat%
      \SB@dimenii\ht\SB@boxii%
      \divide\SB@dimenii#1\relax%
      \SB@maxmin\SB@dimen>\SB@dimenii%
      \loop%
        \global\setbox\SB@box\copy\SB@boxii%
        \global\setbox\SB@boxiii\hbox{\SB@oneidxpage}%
        \ifdim\ht\SB@box>\z@%
          \advance\SB@dimen\p@%
```

```
        \repeat%
        \box\SB@boxiii%
        \global\setbox\SB@boxii\box\voidb@x%
        \vbadness\SB@cnt\vfuzz\SB@dimenii%
      \fi%
    }%
```

If the `.sbx` file doesn't exist, then instead typeset a page with a message on it indicating that the document must be compiled a second time in order to generate the index.

```
    {%
      \vbox to\textheight{%
        \vfil%
        \unvbox\SB@idxtitlebox%
        \vskip1em\relax%
        \hbox to\hsize{\hfil[Index not yet generated.]\hfil}%
        \vskip\z@\@plus2fil\relax%
      }%
    }%
    \clearpage%
  \endgroup\fi%
}
```

`\showindex` Create an index with title ⟨*arg2*⟩ based on the data associated with index identifier ⟨*arg3*⟩ (which was passed to `\newindex`). Optional argument ⟨*arg1*⟩ specifies the number of columns. This macro calls the appropriate index-creation macro depending on the type of index that ⟨*arg3*⟩ was declared to be.

```
\newcommand\showindex[3][0]{%
  \@ifundefined{SB@idxsel@#3}{\SB@errnoidx{#3}}{%
    \expandafter\let\expandafter\SB@temp\csname SB@idxsel@#3\endcsname%
    \SB@cnt#1\relax%
    \ifnum\SB@cnt<\@ne\SB@cnt\SB@temp232\relax\fi%
    \expandafter\SB@temp%
    \expandafter\SB@maketitleindex%
    \expandafter\SB@makescripindex%
    \expandafter\SB@makeauthorindex%
    \expandafter{\the\SB@cnt}%
    {#2}{#3}%
  }%
}
```

`\SB@maketitleindex` Create a song title index. ⟨*arg1*⟩ is a column count, ⟨*arg2*⟩ is the title, and ⟨*arg3*⟩ is the index identifier (which was passed to `\newindex`).

```
\newcommand\SB@maketitleindex{%
  \renewenvironment{SB@lgidx}[1]{
    \hbox{\SB@colorbox\idxbgcolor{\vbox{%
      \hbox to\idxheadwidth{{\idxheadfont\relax##1}\hfil}%
    }}}%
    \nobreak\vskip3\p@\@plus2\p@\@minus2\p@\nointerlineskip%
```

```
  }{\penalty-50\vskip5\p@\@plus5\p@\@minus4\p@}%
  \renewenvironment{SB@smidx}[1]{}{}%
  \renewcommand\idxentry[2]{%
    \SB@ellipspread{\idxtitlefont\relax\ignorespaces##1\unskip}%
                   {{\idxrefsfont\relax##2}}%
  }%
  \renewcommand\idxaltentry[2]{%
    \SB@ellipspread{\idxlyricfont\relax\ignorespaces##1\unskip}%
                   {{\idxrefsfont\relax##2}}%
  }%
  \SB@displayindex%
}
```

`\SB@idxcolhead` In a scripture index, this macro remembers the current book of the bible we're in so that new columns can be headed with "Bookname (continued)".

```
\newcommand\SB@idxcolhead{}
```

`\SB@idxheadsep` Add vertical space following the header line that begins (or continues) a section of a scripture index.

```
\newcommand\SB@idxheadsep{{%
  \SB@dimen4\p@%
  \advance\SB@dimen-\prevdepth%
  \SB@maxmin\SB@dimen<\z@%
  \SB@dimenii\SB@dimen%
  \SB@maxmin\SB@dimenii>\p@%
  \vskip\SB@dimen\@plus\p@\@minus\SB@dimenii%
}}
```

`\SB@idxcont` Typeset the "Bookname (continued)" line that continues a scripture index section when it spans a column break.

```
\newcommand\SB@idxcont[1]{%
  \hbox to\hsize{{\idxcont{#1}}\hfil}%
  \nobreak%
  \SB@idxheadsep\nointerlineskip%
}
```

`\SB@makescripindex` Create a scripture index. ⟨*arg1*⟩ is a column count, ⟨*arg1*⟩ is the title, and ⟨*arg2*⟩ is the index identifier (which was passed to `\newscripindex`).

```
\newcommand\SB@makescripindex{%
  \renewenvironment{SB@lgidx}[1]{%
    \gdef\SB@idxcolhead{##1}%
    \hbox to\hsize{{\idxbook{##1}}\hfil}%
    \nobreak%
    \SB@idxheadsep\nointerlineskip%
  }{%
    \mark{\noexpand\relax}%
    \penalty-20\vskip3\p@\@plus3\p@\relax%
  }%
  \renewenvironment{SB@smidx}[1]
```

```
    {\begin{SB@lgidx}{##1}}{\end{SB@lgidx}}%
  \renewcommand\idxentry[2]{%
    \SB@ellipspread{\hskip.25cm\idxscripfont\relax##1}%
                   {{\idxrefsfont\relax##2}}%
    \SB@toks\expandafter{\SB@idxcolhead}%
    \mark{\noexpand\SB@idxcont{\the\SB@toks}}%
  }%
  \renewcommand\idxaltentry[2]{\SB@erridx{a scripture}}%
  \SB@displayindex%
}
```

`\SB@makeauthorindex` Create an author index. ⟨*arg1*⟩ is a column count, ⟨*arg2*⟩ is the title, and ⟨*arg2*⟩ is the index identifier (which was passed to `\newauthindex`).

```
\newcommand\SB@makeauthorindex{%
  \renewenvironment{SB@lgidx}[1]{}{}%
  \renewenvironment{SB@smidx}[1]{}{}%
  \renewcommand\idxentry[2]{%
    \SB@ellipspread{{\idxauthfont\relax\sfcode`.\@m##1}}%
                   {{\idxrefsfont##2}}%
  }%
  \renewcommand\idxaltentry[2]{\SB@erridx{an author}}%
  \SB@displayindex%
}
```

16.16 Error Messages

We break error messages out into separate macros here in order to reduce the length (in tokens) of the more frequently used macros that do actual work. This can result in a small speed improvement on slower machines.

`\SB@Error` `\SB@Warn` All errors and warnings will be reported as coming from package "songs".

```
\newcommand\SB@Error{\PackageError{songs}}
\newcommand\SB@Warn{\PackageWarning{songs}}
```

`\SB@errspos`

```
\newcommand\SB@errspos{%
  \SB@Error{Illegal \protect\songpos\space argument}{The argume%
  nt to \protect\songpos\space must be a number from 0 to 3.}%
}
```

`\SB@errnse`

```
\newcommand\SB@errnse{%
  \SB@Error{Nested songs environments are not supported}{End th%
  e previous songs environment before beginning the next one.}%
}
```

\SB@errpl

```
\newcommand\SB@errpl{%
  \SB@Error{\protect\includeonlysongs\space not permitted with%
  in a songs environment}{\protect\includeonlysongs\space can o%
  nly be used in the document preamble or between songs environ%
  ments in the document body.}%
}
```

\SB@warnigen

```
\newcommand\SB@warnigen{%
  \PackageWarning{songs}{Indexes shown but index-generation inh%
  ibited. Index files may be out of date.}{Indexes were initial%
  ly turned off by the end of the document preamble, which mean%
  s that the auxiliary files used to keep the indexes up-to-da%
  te were not generated. However, indexes were turned on withi%
  n the document body using \protect\indexeson, which means th%
  e indexes shown in the resulting document are being created f%
  rom outdated files. To correct the problem, be sure that inde%
  xes are turned on by the end of the preamble so that the auxi%
  liary files will be updated.}%
}
```

\SB@errrtopt

```
\newcommand\SB@errrtopt{%
  \SB@Error{Cannot display chords in a rawtext dump}{You have u%
  sed the rawtext option in the \protect\usepackage\space lin%
  e and have either used the chorded option as well or have use%
  d the \protect\chordson\space macro subsequently.}%
}
```

\SB@warnrc

```
\newcommand\SB@warnrc{%
  \SB@Warn{The \protect\repchoruses\space feature will not wor%
  k when the number of columns is set to zero}%
}
```

\SB@errboo

```
\newcommand\SB@errboo{%
  \SB@Error{Encountered \protect\beginsong\space without seein%
  g an \protect\endsong\space for the previous song}%
  {Song \thesongnum\space might be missing a%
  n \protect\endsong\space line.}%
}
```

\SB@errbor

```
\newcommand\SB@errbor{%
  \SB@Error{Encountered \protect\beginsong\space without seein%
  g an \protect\endscripture\space for the preceding scriptur%
  e quotation}{A scripture quotation appearing after son%
```

```
  g \thesongnum\space might be missing a%
  n \protect\endscripture\space line.}%
}
```

\SB@erreov

```
\newcommand\SB@erreov{%
  \SB@Error{Encountered \protect\endsong\space without seein%
  g an \protect\endverse\space for the preceding verse}{Son%
  g \thesongnum\space has a \protect\beginverse\space%
  line with no matching \protect\endverse\space line.}%
}
```

\SB@erreoc

```
\newcommand\SB@erreoc{%
  \SB@Error{Encountered \protect\endsong\space without seein%
  g an \protect\endchorus\space for the preceding chorus}{Son%
  g \thesongnum\space has a \protect\beginchorus\space%
  line with no matching \protect\endchorus\space line.}%
}
```

\SB@erreor

```
\newcommand\SB@erreor{%
  \SB@Error{Encountered \protect\endsong\space without seein%
  g an \protect\endscripture for the preceding scripture quot%
  e}{A scripture quote appearing before song \thesongnum\space%
  ended with \protect\endsong\space instead of wit%
  h \protect\endscripture.}%
}
```

\SB@erreot

```
\newcommand\SB@erreot{%
  \SB@Error{Encountered \protect\endsong\space with no matchin%
  g \protect\beginsong}{Before song \thesongnum\space there wa%
  s an \protect\endsong\space with no matchin%
  g \protect\beginsong.}%
}
```

\SB@errbvv

```
\newcommand\SB@errbvv{%
  \SB@Error{Encountered \protect\beginverse\space without seein%
  g an \protect\endverse\space for the preceding verse}{Son%
  g \thesongnum\space might have a verse that has n%
  o \protect\endendverse\space line.}%
}
```

\SB@errbvc

```
\newcommand\SB@errbvc{%
  \SB@Error{Encountered \protect\beginverse\space without seein%
  g an \protect\endchorus\space for the preceding chorus}{Son%
  g \thesongnum\space might have a chorus that has n%
  o \protect\endchorus\space line.}%
}
```

\SB@errbvt

```
\newcommand\SB@errbvt{%
  \SB@Error{Encountered \protect\beginverse\space without firs%
  t seeing a \protect\beginsong\space line}{Before son%
  g \thesongnum, there is a \protect\beginverse\space line no%
  t contained in any song.}%
}
```

\SB@errevc

```
\newcommand\SB@errevc{%
  \SB@Error{Encountered \protect\endverse\space while process%
  ing a chorus}{Song \thesongnum\space might hav%
  e a \protect\beginchorus\space concluded by a%
  n \protect\endverse\space instead of an \protect\endchorus.}%
}
```

\SB@errevo

```
\newcommand\SB@errevo{%
  \SB@Error{Encountered \protect\endverse\space without firs%
  t seeing a \protect\beginverse}{Song \thesongnum\space m%
  ight have an \protect\endverse\space with no matchin%
  g \protect\beginverse.}%
}
```

\SB@errevt

```
\newcommand\SB@errevt{%
  \SB@Error{Encountered an \protect\endverse\space outside o%
  f any song}{Before song \thesongnum, there is a%
  n \protect\endverse\space line not preceded b%
  y a \protect\beginsong\space line.}%
}
```

\SB@erretex

```
\newcommand\SB@erretex{%
  \SB@Error{The \protect\repchoruses\space feature requires e-%
  TeX compatibility}{Your version of LaTeX2e does not appear t%
  o be e-TeX compatible. Find a distribution that includes e-T%
  eX support in order to use this feature.}%
}
```

\SB@errbcv

```
\newcommand\SB@errbcv{%
  \SB@Error{Encountered \protect\beginchorus\space without see%
  ing an \protect\endverse\space for the preceding verse}{Son%
  g \thesongnum\space might hav%
  e a \protect\beginverse\space with no match%
  ing \protect\endverse.}%
}
```

\SB@errbcc

```
\newcommand\SB@errbcc{%
  \SB@Error{Encountered \protect\beginchorus\space without see%
  ing an \protect\endchorus\space for the preceding chorus}%
  {Song \thesongnum\space might have a \protect\beginchorus%
  \space with no matching \protect\endchorus.}%
}
```

\SB@errbct

```
\newcommand\SB@errbct{%
  \SB@Error{Encountered \protect\beginchorus\space without see%
  ing a \protect\beginsong\space line first}{After son%
  g \thesongnum\space there is a \protect\beginchorus\space%
  line outside of any song.}%
}
```

\SB@errecv

```
\newcommand\SB@errecv{%
  \SB@Error{Encountered an \protect\endchorus\space while proc%
  essing a verse}{Song \thesongnum\space might hav%
  e a \protect\beginverse\space concluded by \protect\endchorus%
  \space instead of \protect\endverse.}%
}
```

\SB@erreco

```
\newcommand\SB@erreco{%
  \SB@Error{Encountered \protect\endchorus\space without firs%
  t seeing a \protect\beginchorus}{Song \thesongnum\space m%
  ight have an \protect\endchorus\space with no match%
  ing \protect\beginchorus.}%
}
```

\SB@errect

```
\newcommand\SB@errect{%
  \SB@Error{Encountered an \protect\endchorus\space outside o%
  f any song}{Before song \thesongnum, there is a%
  n \protect\endchorus\space line not preceded b%
  y a \protect\beginsong\space line.}%
}
```

\SB@errbro

```
\newcommand\SB@errbro{%
  \SB@Error{Missing \protect\endsong}%
  {Nested song and intersong environments are not supported%
  . Song \thesongnum\space might be missing a%
  n \protect\endsong\space line.}%
}
```

\SB@errbrr

```
\newcommand\SB@errbrr{%
  \SB@Error{Nested intersong environments are not supported}%
  {A scripture quote or other intersong environment before s%
  ong \thesongnum\space is missing its ending line.}%
}
```

\SB@errero

```
\newcommand\SB@errero{%
  \SB@Error{Encountered an \protect\endscripture\space whil%
  e processing a song}{Song \thesongnum\space ends wit%
  h \protect\endscripture\space when it should end wit%
  h \protect\endsong.}%
}
```

\SB@errert

```
\newcommand\SB@errert{%
  \SB@Error{Encountered an \protect\endscripture\space with%
  out first seeing a \protect\beginscripture}{Before son%
  g \thesongnum, there is an \protect\endscripture\space w%
  ith no matching \protect\beginscripture.}%
}
```

\SB@errscrip

```
\newcommand\SB@errscrip[1]{%
  \SB@Error{Encountered a \protect#1\space outside a scriptu%
  re quote}{\protect#1\space can only appear betwee%
  n \protect\beginscripture\space an%
  d \protect\endscripture\space lines.}%
}
```

\SB@errchord

```
\newcommand\SB@errchord{%
  \SB@Error{Song \thesongnum\space seems to have chord%
  s that appear outside of any verse or chorus}{All chords a%
  nd lyrics should appear between \protect\beginverse\space%
  and \protect\endverse, or between \protect\beginchorus\space%
  and \protect\endchorus.}%
}
```

\SB@errreplay

```
\newcommand\SB@errreplay{%
  \SB@Error{Replayed chord has no matching chord}{Son%
  g \thesongnum\space uses \protect^ more times than the%
  re are chords in the previously memorized verse.}%
}
```

\SB@errreg

```
\newcommand\SB@errreg[1]{%
  \SB@Error{Unknown chord-replay register name: #1}{Chord-re%
  play registers must be declared with \protect\newchords.}%
}
```

\SB@errdup

```
\newcommand\SB@errdup[1]{%
  \SB@Error{Duplicate definition of chord-replay register%
  : #1}{\protect\newchords\space was used to declare the sa%
  me chord-replay register twice.}%
}
```

\SB@errmbar

```
\newcommand\SB@errmbar{%
  \SB@Error{Song \thesongnum\space seems to have measur%
  e bars that appear outside of any verse or chorus}{All mea%
  sure bars (produced with \protect\mbar\space or |) must ap%
  pear between \protect\beginverse\space an%
  d \protect\endverse, or between \protect\beginchorus\space%
  and \protect\endchorus.}%
}
```

\SB@errtab

```
\newcommand\SB@errtab{%
  \SB@Error{Invalid argument to \protect\gtab\space macro. R%
   eplacing it with \protect\0.}{Valid arguments consist onl%
   y of: X, O, 0, 1, 2, 3, or 4.}%
}
```

\SB@errnoidx

```
\newcommand\SB@errnoidx[1]{%
  \SB@Error{Unknown index identifier: #1}{This index identifie%
  r was never declared using \protect\newindex.}%
}
```

\SB@erridx

```
\newcommand\SB@erridx[1]{%
  \SB@Error{\protect\idxaltentry\space not allowed in #1 index}%
  {This error should not occur. The index generation routines ha%
  ve malfunctioned. Try deleting all temporary files and then re%
  compiling.}%
}
```

16.17 Option Processing

\ifchorded
\iflyric
\ifslides
\ifmeasures
\ifpartiallist
\ifrepchorus
\iftranscapos
\ifnolyrics
\ifrawtext
\ifpdfindex
\ifsongindexes
\ifSB@colorboxes
\ifSB@genindexes
\ifSB@omitscrip

Reserve conditionals for all of the various option settings. We wait to define these since if any are used earlier than this, it is an error in the package code, and we'd rather get an error than continue.

```
\newif\ifchorded
\newif\iflyric\lyrictrue
\newif\ifslides
\newif\ifmeasures
\newif\ifpartiallist
\newif\ifrepchorus
\newif\iftranscapos
\newif\ifnolyrics
\newif\ifrawtext
\newif\ifpdfindex\pdfindextrue
\newif\ifsongindexes\songindexestrue
\newif\ifSB@colorboxes\SB@colorboxestrue
\newif\ifSB@genindexes\SB@genindexestrue
\newif\ifSB@omitscrip
```

\nolyrics

The `\nolyrics` macro is just shorthand for `\nolyricstrue`.

```
\newcommand\nolyrics{}
\let\nolyrics\nolyricstrue
```

Finally we're ready to process all of the package options. This is delayed until near the end because the option processing code needs to execute various macros found in the previous sections.

```
\SB@chordson
\ProcessOptions\relax
```

If we're not generating a pdf, then don't generate the pdf index.

```
\ifSB@pdf\else\pdfindexfalse\fi
```

\SB@colorbox

Include the colors package and define colors, if requested.

```
\ifSB@colorboxes
  \RequirePackage{color}
  \definecolor{SongbookShade}{gray}{.80}
  \newcommand\SB@colorbox[2]{%
    \ifx\@empty#1%
      \vbox{%
        \kern3\p@%
        \hbox{\kern3\p@{#2}\kern3\p@}%
        \kern3\p@%
      }%
    \else%
      \colorbox{#1}{#2}%
    \fi%
  }
\else
```

```
  \newcommand\SB@colorbox[2]{\vbox{%
    \kern3\p@%
    \hbox{\kern3\p@{#2}\kern3\p@}%
    \kern3\p@%
  }}
\fi
```

16.18 Rawtext Mode

If generating raw text, most of what has been defined previously is ignored in favor of some very specialized macros that write all the song lyrics to a text file.

```
\ifrawtext
  \newwrite\SB@txtout
  \immediate\openout\SB@txtout=\jobname.txt
  \newif\ifSB@doEOL
  {\catcode`\^^M12 %
   \catcode`\^^J12 %
   \gdef\SB@printEOL{\ifSB@doEOL^^M^^J\fi}}
  {\catcode`#12\gdef\SB@hash{#}}
  {\catcode`&12\gdef\SB@amp{&}}
  \renewcommand\SB@@@beginsong{%
    \begingroup%
      \def\'{}\def\`{}\def\v{}\def\u{}\def\={}\def\^{}%
      \def\.{}\def\H{}\def\~{}\def\"{}\def\t{}%
      \def\copyright{(c)}%
      \let~\space%
      \let\par\SB@printEOL%
      \let\#\SB@hash%
      \let\&\SB@amp%
      \catcode`|9 %
      \catcode`*9 %
      \catcode`^9 %
      \def\[##1]{}%
      \resettitles%
      \immediate\write\SB@txtout{\thesongnum. \songtitle}%
      \nexttitle%
      \foreachtitle{\immediate\write\SB@txtout{(\songtitle)}}%
      \ifx\songauthors\@empty\else%
         \immediate\write\SB@txtout{\songauthors}%
      \fi%
      \ifx\SB@rawrefs\@empty\else%
         \immediate\write\SB@txtout{\SB@rawrefs}%
      \fi%
      \immediate\write\SB@txtout{}%
      \SB@doEOLfalse%
      \obeylines%
  }
  \renewcommand\SB@endsong{%
      \SB@doEOLtrue%
```

```
      \immediate\write\SB@txtout{\songcopyright\space%
        \songlicense\SB@printEOL}%
    \endgroup%
    \SB@insongfalse%
    \stepcounter{songnum}%
  }
  \def\SB@parsesrefs#1{\def\songrefs{#1}}
  \long\def\beginverse#1#2\endverse{%
    \SB@doEOLtrue\begingroup%
      \def\textnote##1{##1}%
      \def\SB@temp{#1}%
      \def\SB@star{*}%
      \ifx\SB@temp\SB@star%
        \immediate\write\SB@txtout{\@gobble#2}%
      \else%
        \immediate\write\SB@txtout{#2}%
      \fi%
    \endgroup\SB@doEOLfalse}
  \long\def\beginchorus#1\endchorus{%
    \SB@doEOLtrue\begingroup%
      \def\textnote##1{##1}%
      \immediate\write\SB@txtout{Chorus:#1}%
    \endgroup\SB@doEOLfalse}
  \long\def\beginscripture#1\endscripture{}
  \def\musicnote#1{}
  \def\textnote#1{%
    \SB@doEOLtrue%
    \immediate\write\SB@txtout{#1\SB@printEOL}%
    \SB@doEOLfalse}
  \def\brk{}
  \def\rep#1{(x#1)}
  \def\echo#1{(#1)}
  \def\mbar#1#2{}
  \def\lrep{}
  \def\rrep{}
  \def\nolyrics{}
  \renewcommand\memorize[1][]{}
  \renewcommand\replay[1][]{}
\fi
```

16.19 Codeline Index

Numbers underlined refer to the code line where the corresponding entry is defined; numbers in roman refer to the code lines where the entry is used.

A

B

C

D

J

L

M

N

O

P

R

S

T

U

V

W