The **songs** package*

Kevin W. Hamlen

March 10, 2023

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, handouts, and a variety of specialized song indexes are 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

The **songs** package is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. A copy of the license can be found in §**??**.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License in §**??** for details. A copy of the license can also be obtained by writing to the Free Software Foundation, Inc., 51 Franklin Street, 5th Floor, Boston, MA 02110-1301, USA.

---

*This manual documents **songs** v3.1, dated 2018/09/12, © 2018 Kevin W. Hamlen, and distributed under version 2 the GNU General Public License as published by the Free Software Foundation.

3   Sample Document

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. 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}

\noversenumbers

\begin{document}

\songsection{Worship Songs}

\begin{songs}{}
\beginsong{Doxology}[by={Louis Bourgeois and Thomas Ken},
                     sr={Revelation 5:13},
                     cr={Public domain.}]
\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, run LaTeX (`pdflatex` is recommended):

`pdflatex mybook.tex`

(where `mybook.tex` is the name of the source document above). The final document is named `mybook.pdf` if you use `pdflatex` or `mybook.dvi` if you use regular `latex`.

Note that compiling a document that includes indexes requires extra steps. See §**??** for details.

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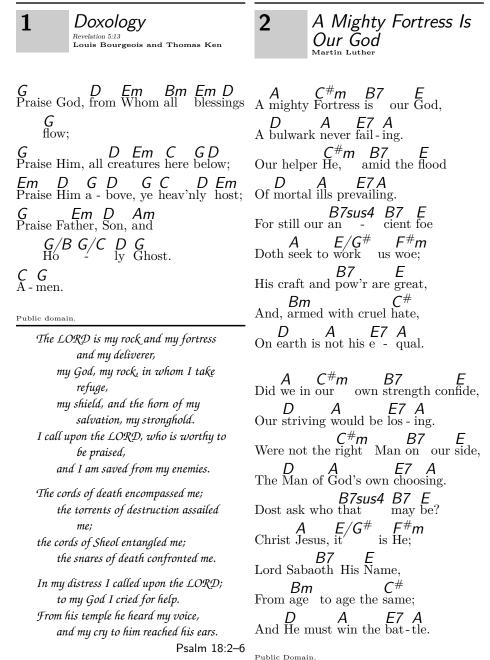
Worship Songs

1 Doxology

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

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

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:

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. The `slides` and `chorded` options can be used together to create chorded slides. If no output options are specified, `chorded` is the default.

lyric
chorded
slides
rawtext

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

Raw text output yields an ascii text file named ⟨*jobname*⟩`.txt` (where ⟨*jobname*⟩ is the root filename) containing lyrics without chords. This can be useful for importing song books into another program, such as a spell-checker.

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

\chordson
\chordsoff
\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.

Measure Bars. The songs package includes a facility for placing measure bars in chord books (see §**??**). 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.

nomeasures
showmeasures
\measureson
\measuresoff

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

transposecapos

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.

noindexes
\indexeson
\indexesoff
nopdfindex

PDF bookmark entries and hyperlinks can be suppressed with the `nopdfindex` option. For finer control of PDF indexes, see §**??**.

Scripture Quotations. The `noscripture` option omits scripture quotations (see §**??**) from the output. You can also turn scripture quotations on or off in the middle of the document by using `\scriptureon` or `\scriptureoff`, respectively.

noscripture
\scriptureon
\scriptureoff

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

\includeonlysongs   **Partial Song Sets.**   Often it is useful to be able to extract a subset of songs from the master document—e.g. to create a handout or set of overhead slides for a specific worship service. To do this, you can type `\includeonlysongs{⟨songlist⟩}` in the document preamble (i.e., before the `\begin{document}` line), where ⟨*songlist*⟩ is a comma-separated list of the song numbers to include. 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 (§**??**), 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*⟩.

The `\includeonlysongs` macro only reorders songs within each `songs` environment (see §**??**), not between different `songs` environments. It also cannot be used in conjunction with the `rawtext` option.

5   Songs

5.1   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 ⟨*id*⟩'s (see §**??**)—one identifier for each index that is to include songs in this song set. Between the `\begin{songs}` and `\end{songs}` lines of a song section only songs (see below) or inter-song environments (see §**??**) may appear. No text in a `songs` environment may appear outside of these environments.

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

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

Songs should appear only within `songs` environments (see above) unless you are supplying your own page-builder (see §**??**).

In the `\beginsong` line, ⟨*titles*⟩ is one or more 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 an optional comma-separated list of key-value pairs (keyvals) of the form ⟨*key*⟩=⟨*value*⟩. 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~2023 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 §**??**. You can also create your own keyvals (see §**??**).

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= **Licensing Info.** Licensing information is provided by li={⟨*license*⟩}, where
\setlicense ⟨*license*⟩ is any text. Licensing information is displayed in fine print under the song just after the copyright information (if any). Alternatively, writing \setlicense{⟨*license*⟩} anywhere between the \beginsong and \endsong lines is equivalent to using li={⟨*license*⟩} in the \beginsong line.

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

Figure 2: Formal syntax rules for song scripture references

When many songs in a book are covered by a common 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.

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 script (see §**??**) 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 **??**. In those syntax rules, ⟨*chapter*⟩ and ⟨*verse*⟩ stand for 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*⟩).

5.2 Verses and Choruses

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

and a chorus begins and ends with:

```
\beginchorus
⋮
\endchorus
```

Verses are numbered (unless `\noversenumbers` has 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. This can be 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`.

5.3 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; they are omitted from lyric books. The ⟨*chordname*⟩ may 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* $E^\flat$ Am
peace and joy

If whitespace (a space or ⟨*return*⟩) 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* $E^\flat$ Am
peace and joy

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* $F^\#sus4$ A
e - ternal

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

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

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

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

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

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

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

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

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

Without the braces, the word "eternal" would be pushed out away from the chord so that the chord would appear only over the partial word "th'".

\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. Instrumental solos should typically not appear in lyric books, so such lines should usually also be surrounded by \ifchorded and \fi (see §**??**).

\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*⟩}

9

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*   my *Gmaj7sus4* life

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

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

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*   me free *F G Am*

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 in chord macros, 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}`

5.4   Replaying Chords and Choruses

^   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 of 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

$\overset{G}{\text{This}}$ is the $\overset{C}{\text{first}}$ $\overset{G}{\text{verse.}}$

$\overset{G}{\text{The second verse}}$ $\overset{C}{}$ has the same $\overset{G}{\text{chords.}}$

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

$\overset{G}{\text{The third}}$ $\overset{C}{\text{verse has a}}$ $\overset{Cm}{\text{new}}$ $\overset{G}{\text{chord.}}$

Replaying can be used in combination with automatic transposition to produce modulated verses. See §?? for an example.

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

$\overset{G}{\text{The third}}$ $\overset{C}{\text{chord will}}$ $\overset{Cm}{\text{not}}$ be $\overset{G}{\text{replayed.}}$

$\overset{G}{\text{When replaying, the}}$ $\overset{C}{\text{unmemorized chord is}}$ $\overset{G}{\text{skipped.}}$

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.

11

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 chords 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.

`\repchoruses`   **Replaying 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 subsequent 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`   Writing `\norepchoruses` turns off chorus repetition for subsequent songs.

If you need finer control over where replayed choruses appear, use the conditional macros covered in §**??** instead of `\repchoruses`. For example, to manually insert a chorus into only slide books at a particular point (without affecting other versions of your book), you could write:

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

and copy and paste the desired chorus into the middle.

5.5   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 §**??**. 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`.

5.6   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* Alleluia! *(Alleluia!)*

`\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. 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* ‖:Alleluia!:‖(×4)

5.7 Measure Bars

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

`Alle|\[G]luia`   *produces*   Alle|luia G

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

`\meter`   The first measure bar in a song 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

Sing to the |heavens, ye |saints of |old!

5.8 Textual Notes

`\textnote` Aside from verses and choruses, songs can also contain textual notes that provide
`\musicnote` 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 a fret on which they should put their capos. 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 §**??** for more information on automatic chord transposition.

5.9   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 typographical 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*   difficult

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 §**??**), 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*   difficult

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*   difficult

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.

6   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 an optional digit from 2 to 9 placed to the left of the diagram.

⟨*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*⟩ is an optional 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*

To create a barre chord in which one finger is extended across multiple strings, use parentheses () or brackets [] in the ⟨*strings*⟩ argument to group the barred strings. Each such group will draw a barre on the lowest numbered fret it contains. For example:

\gtab{C7}{X(3535X):013140}   *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.

16

Tablatures Within Macros Macros that produce tablatures must not bury the colons that separate the ⟨*fret*⟩, ⟨*strings*⟩, and ⟨*fingering*⟩ arguments within other macros, and it's safest to always include both colons to avoid ambiguities related to optional argument parsing. For example,

```
\newcommand{\mystrings}{X4412X}
\newcommand{\myfingers}{X3412X}
\newcommand{\mychord}{\gtab{C\shrp}{:\mystrings:\myfingers}}
```

works as expected. But omitting the colon before `\mystrings` in the definition of `\mychord` confuses `\gtab` into thinking `\mystrings` is the ⟨*fret*⟩ argument, and writing code like `\gtab{C\shrp}{\allargs}` with `\allargs` defined to something with colons results in an error, because it confuses `\gtab` into thinking that `\allargs` is only the ⟨*strings*⟩ argument.

7   Automatic Transposition

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

`\transpose{`⟨*n*⟩`}`

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

`\preferflats` **Enharmonics.** When using `\transpose` to automatically transpose the chords
`\prefersharps` 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*♭ 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 §**??**) 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

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

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

\trchordformat **Both Keys.** By default, when chords are automatically transposed using
\transpose, only the transposed chords are printed. However, in some cases
you may wish to print the old chords and the transposed chords together so that
musicians playing transposing and non-transposing instruments can play from
the same piece of music. This can be achieved by redefining the \trchordformat
macro, which receives 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 **Changing Note Names.** In many countries it is common to use the solfedge
\alphascale 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

$$\text{\textbackslash notenames}\{\langle nameA\rangle\}\{\langle nameB\rangle\}\dots\{\langle nameG\rangle\}$$

where each of $\langle nameA\rangle$ through $\langle nameG\rangle$ 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   The songs package can also automatically convert one set of note names to
\notenamesout 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 does not 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}}
```

`\notrans` Transposition can be suppressed within material that would otherwise be transposed by using the `\notrans` macro. For example, writing

```
\transposehere{G = \notrans{G}}
```

would typeset a transposed *G* followed by a non-transposed *G* chord. This does not suppress note name conversion (see `\notenames`). To suppress both transposition and note name conversion, just use braces (e.g., `{G}` instead of `\notrans{G}`).

`\gtabtrans` **Transposing Guitar Tablatures.** The songs package cannot automatically transpose tablature diagrams (see §**??**). Therefore, when automatic transposition is taking place, only the chord names of `\gtab` macros are displayed (and transposed); the diagrams are omitted. To change this default, redefine the `\gtabtrans` macro, whose two arguments are the two arguments to `\gtab`. For example, to display original tablatures without transposing them even when transposition has been turned on, write

```
\renewcommand{\gtabtrans}[2]{\gtab{\notrans{#1}}{#2}}
```

To transpose the chord name but not the diagram under it, replace `\notrans{#1}` with simply `#1` in the above. To restore the default behavior, write

```
\renewcommand{\gtabtrans}[2]{\transposehere{#1}}
```

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:

```
\begin{intersong}
⋮
\end{intersong}
```

Material contributed in an `intersong` environment is subject to the same column-breaking rules as songs (see §**??**), but all other formatting is up to you. 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:

```
\begin{intersong*}
⋮
\end{intersong*}
```

This starts a new page if the current page already has column-width material in it.

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 §**??**), 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 **Starting a Scripture Quotation.** A special form of intersong block typesets a
\endscripture 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 §**??**).

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

\Acolon
\Bcolon
Tuplets. To typeset biblical poetry the way it appears in most bibles, begin each line with either `\Acolon` or `\Bcolon`. A-colons are typeset flush with the left margin, while B-colons are indented. Any lines too long to fit are wrapped with double-width hanging indentation. 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   Chapters and Sections

\songsection
\songchapter
Song books can be divided into chapters and sections using all the usual macros provided by LaTeX (e.g., `\chapter`, `\section`, etc.) and by other macro packages. In addition, the songs package provides two helpful built-in sectioning macros:

```
\songchapter{⟨title⟩}
\songsection{⟨title⟩}
```

which act like LaTeX's `\chapter` and `\section` commands except that they center the ⟨*title*⟩ text in sans serif font and omit the chapter/section number. The `\songchapter` macro only works in document classes that support `\chapter` (e.g., the book class).

10   Indexes

10.1   Index Creation

\newindex
\newauthorindex
\newscripindex
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 (no font or formatting macros, since those cannot be used in pdf bookmark indexes). The `[`⟨*columns*⟩`]` part is optional; if specified it dictates the number of columns if the index can't fit in a single column. For example, for a 2-column title index, write:

> `\showindex[2]{Index of Song Titles}{mainindex}`

10.2   Index Entries

Every song automatically gets entries in the current `songs` environment's list of title index(es) (see §**??**). 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` You can also create index entries by saying `\indexentry[⟨indexes⟩]{⟨lyrics⟩}`
`\indextitleentry` (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.

10.3 Compiling

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

    pdflatex mybook.tex

Second, use the `songidx.lua` script to generate an index for each index that you declared with `\newindex`, `\newauthorindex`, or `\newscripindex`. The script can be launched using LuaTeX, using the following syntax:

    texlua songidx.lua [-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 to generate indexes sorted according to a Christian English bible, execute:

    texlua songidx.lua titlfile.sxd titlfile.sbx
    texlua songidx.lua authfile.sxd authfile.sbx
    texlua songidx.lua -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

11 Customizing the Book

11.1 Song and Verse Numbering

`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 its text is exported to auxiliary index generation files where it is sorted.

\printsongnum   To change the formatting of song numbers as they appear at the beginning of each song, 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}}
```

\printversenum   To change the formatting of verse numbers as they appear at the beginning of each verse, 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 §**??** 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), with minimal line spacing. You can change this default by redefining `\stitlefont`. For example, to cause titles to be typeset in a roman font with lines spaced 20 points apart, you could define

```
\renewcommand{\stitlefont}{
   \rmfont\Large\baselineskip=20pt\lineskiplimit=0pt
}
```

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

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

`\notebgcolor`   The colors of shaded boxes containing textual notes and song numbers can be
`\snumbgcolor`   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`

\afterpreludeskip
\beforepostludeskip

The vertical distance between the song's body and its prelude and postlude material is controlled by skips `\afterpreludeskip` and `\beforepostludeskip`. This glue can be made stretchable for centering effects. For example, to cause each song body to be centered on the page with one song per page, you could write:

> `\songcolumns{1}`
> `\spenalty=-10000`
> `\afterpreludeskip=2pt plus 1fil`
> `\beforepostludeskip=2pt plus 1fil`

\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`

\clineparams

To change the vertical distance between chords and the lyrics below them, redefine the `\clineparams` macro with a definition that adjusts the LaTeX parameters `\baselineskip`, `\lineskiplimit`, and `\lineskip`. For example, to cause the baselines of chords and their lyrics to be 12 points apart with at least 1 point of space between the bottom of the chord and the top of the lyric, you could write:

> `\renewcommand{\clineparams}{`
>   `\baselineskip=12pt`
>   `\lineskiplimit=1pt`
>   `\lineskip=1pt`
> `}`

\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`:

27

```
\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   By default, \extendprelude displays the song's authors and scripture refer-
\showauthors   ences using the macros \showauthors and \showrefs. The following definition
\showrefs   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 in-
formation 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 §**??** 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 pre-formatted paragraph the way it appears in the default song header blocks.

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

\makeprelude   For complete control over the appearance of the header and footer material that
\makepostlude   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.

As a simple example, the following causes each song to start with its number and title(s), centered, in a large, boldface font, and then centers the rest of the prelude material (e.g., references and authors) below that (using \extendprelude).

```
\renewcommand\makeprelude{%
  \resettitles
  \centering
  {\Large\bfseries\thesongnum. \songtitle\par
   \nexttitle\foreachtitle{(\songtitle)\par}}%
  \extendprelude
}
```

\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 site. 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 left-justify or center the lines of verses or choruses, redefine `\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 §**??** 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 §**??** 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
rawtext	the rawtext option is active
transcapos	the transposecapos option is active
nolyrics	the \nolyrics macro is in effect
pagepreludes	the \pagepreludes 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 **??**. 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.

30

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`.

\pagepreludes Song preludes (i.e., the material atop each song, including the title) are typeset by default at column width. Writing `\pagepreludes` typesets subsequent preludes at page width atop fresh pages, with the rest of the song in multiple columns beneath its title. (To prohibit separation of songs from their preludes, it also sets `\songpos` to 0.)

\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 typesetting 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

11.6.1 Index Appearance

Index Titles. To customize the appearance of index titles, redefine the \songsection and/or \songchapter macros from §**??**. For example, to use LaTeX's built-in \section and \chapter macros instead, you could write:

```
\renewcommand{\songchapter}{\chapter}
\renewcommand{\songsection}{\section}
```

\sepindexestrue
\sepindexesfalse   **Layout and page divisions.** Indexes are by default typeset on separate pages, and when an index is sufficiently small, it is centered on the page in one column. To disable these defaults, write \sepindexesfalse. This causes indexes to avoid using unnecessary vertical space or starting unnecessary new pages. To re-enable the defaults, use \sepindexestrue.

\idxheadwidth   The \idxheadwidth length defines the width of the shaded boxes that begin each alphabetic block of a large title index. Setting it to 0pt suppresses the boxes entirely. For example, to set the width of those boxes to 1 centimeter, you could define

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

\idxrefsfont   **Fonts and colors.** 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` Title indexes contain entries for song titles and also entries for notable
`\idxlyricfont` 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}`

`\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 bold-
face, 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)}`

11.6.2 Entry References

`\indexsongsas` 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.

33

11.6.3 PDF Bookmarks and Links

`\songtarget`
`\songlink`
Each `\beginsong` environment adds a PDF bookmark (if generating a PDF) and hyperlink target (if using the `hyperref` package) for the song by invoking `\songtarget` with two arguments: (1) a suggested PDF bookmark level, and (2) a link target name. Links in indexes to these targets are created by `\songlink`, which also gets two arguments: (1) the link target name (same as the second argument to `\songtarget`), and (2) the text to be linked.

Redefine these macros to customize or suppress these bookmarks, targets, and links. For example, to enable both bookmarks and links (the default behavior) use:

```
\renewcommand{\songtarget}[2]
   {\pdfbookmark[#1]{\thesongnum. \songtitle}{#2}}
\renewcommand{\songlink}[2]{\hyperlink{#1}{#2}}
```

To enable links but not bookmarks, use:

```
\renewcommand{\songtarget}[2]{\hypertarget{#2}{\relax}}
\renewcommand{\songlink}[2]{\hyperlink{#1}{#2}}
```

To disable both bookmarks and links, use:

```
\renewcommand{\songtarget}[2]{}
\renewcommand{\songlink}[2]{#2}
```

11.6.4 Sort Order

The alphabetic ordering of entries in title and author indexes is dictated by the computer system on which the songs software is installed. Different languages and regions have different sorting conventions, so the `songidx` Lua script delegates decisions about order to your operating system. If the default ordering proves inadequate, you can modify it by changing your operating system's *locale* (see your system's local help files). Alternatively, you can explicitly tell the `songidx` program which locale to use in one of three ways:

- *Windows:* Edit the `generate.bat` file in the `Sample` folder (or your working folder) with any plain text editor (e.g., Vim or Notepad). Near the top, find the line that says `SET locale=`. After the `=`, type any valid locale name. For a list of valid locale names on Windows, please see the "Language name abbreviation" column of Microsoft's online National Language Support (NLS) API Reference:

    http://msdn.microsoft.com/en-us/goglobal/bb896001.aspx

- *Unix:* Create an environment variable named `SONGIDX_LOCALE` and set it equal to the desired locale name. The command `locale -a` lists all valid locale names on most Unix systems.

- *Command-line:* If you are executing the `songidx` script manually, use the `-l` option to specify the locale:

    texlua songidx -l sv_SE myindex.sxd myindex.sbx

11.6.5 Special Words In Song Info

The following macros control how certain keywords are treated when parsing and sorting index entries. They only affect indexes that have already been declared, so put them strictly after all your index creation commands (see §**??**).

`\titleprefixword`   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`   When parsing author index entries, the word "and" is recognized by the `songidx` script 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.

`\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 §**??**. To access the current song's number or the current verse's number, use `\thesongnum` or `\theversenum` (see §**??**). 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 §**??** 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   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 Song♭ook 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 Song♭ook package. It includes an excellent list of other resources that might be of interest to creators of song books.

14 GNU General Public License

<div align="center">

Terms and Conditions For
Copying, Distribution and Modification

</div>

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.

15 Implementation

The following provides the verbatim implementation of the songs LaTeX package, along with 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 §**??**. To find the implementation of any particular macro, the index at the end of this document should prove helpful.

The unwary TeXer may wonder at the rather large size of the implementation. The volume and complexity of the code stems mainly from the following challenging features:

- Putting chords above lyrics fully automatically requires building an entire lyric-parser in LaTeX (see §**??**).

- Avoiding page-turns within songs without prohibiting column-breaks requires building a completely new page-breaking algorithm (see §**??**).

- The package must be able to generate a daunting number of document variants from a common source: lyric-only books, chorded books, digital slides, transparency slides, selected song subsets, transposed songs, and combinations of the above. This is like putting six or more packages into one.

- Song book indexes are far more complex than those for a prose book. See §**??** for some of the difficulties involved.

15.1 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 whether ε-TeX primitives are available.

```
1 \newif\ifSB@etex
2 \ifx\eTeXversion\undefined\else
3   \ifx\eTeXversion\relax\else
4     \SB@etextrue
5     \ifx\e@alloc\@undefined
6       \IfFileExists{etex.sty}{\RequirePackage{etex}}{}
7     \fi
8   \fi
9 \fi
```

`\ifSB@pdf` Detect whether we're generating a pdf file, since this affects the treatment of hyperlinks and bookmark indexes.

```
10 \newif\ifSB@pdf\SB@pdffalse
11 \IfFileExists{ifpdf.sty}{\RequirePackage{ifpdf}\ifpdf\SB@pdftrue\fi}{
12   \ifx\pdfoutput\undefined\else
13     \ifx\pdfoutput\relax\else
14       \ifnum\pdfoutput<\@ne\else
15         \SB@pdftrue
```

```
16        \fi
17      \fi
18    \fi
19 }
```

\ifSB@preamble Some macros have different effects depending on when they're used in the preamble or in the document body, so we need a conditional that remembers whether we're still in the preamble. It gets initialized to true and later changed to false once the body begins.

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

\ifSB@test \
\ifSB@testii \
\SB@temp \
\SB@tempii \
\SB@tempiii \
\SB@tempiv \
\SB@tempv

Reserve some control sequence names for scratch use.

```
22 \newif\ifSB@test
23 \newif\ifSB@testii
24 \newcommand\SB@temp{}
25 \newcommand\SB@tempii{}
26 \newcommand\SB@tempiii{}
27 \newcommand\SB@tempiv{}
28 \newcommand\SB@tempv{}
```

\SB@newcount \
\SB@newdimen \
\SB@newbox \
\SB@newtoks \
\SB@newwrite

Create macros for safely allocating count, dimen, box, token, and write registers with detection for name-clashes. For some reason, the default allocation macros provided by the LaTeX kernel do not detect name-clashes(!), which means that packages that use them might accidentally overwrite our registers, causing all sorts of problems. But at least we can do our best to avoid overwriting their registers.

```
29 \newcommand\SB@newcount[1]{\@ifdefinable#1{\newcount#1}}
30 \newcommand\SB@newdimen[1]{\@ifdefinable#1{\newdimen#1}}
31 \newcommand\SB@newbox[1]{\@ifdefinable#1{\newbox#1}}
32 \newcommand\SB@newtoks[1]{\@ifdefinable#1{\newtoks#1}}
33 \newcommand\SB@newwrite[1]{\@ifdefinable#1{\newwrite#1}}
```

\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.

```
34 \SB@newdimen\SB@dimen
35 \SB@newdimen\SB@dimenii
36 \SB@newdimen\SB@dimeniii
37 \SB@newdimen\SB@dimeniv
38 \SB@newbox\SB@box
39 \SB@newbox\SB@boxii
40 \SB@newbox\SB@boxiii
41 \SB@newtoks\SB@toks
42 \SB@newcount\SB@cnt
43 \SB@newcount\SB@cntii
44 \newlength\SB@skip
```

\SB@envbox Also reserve a slightly less volatile box register for per-environment use. In scripture environments it holds the scripture citation. In indexes it holds the index title text.

```
45 \SB@newbox\SB@envbox
```

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

```
46 \RequirePackage{keyval}
```

`\SB@app` Utility macro: Append some text to the definition of another macro.

```
47 \newcommand\SB@app[3]{%
48   \expandafter#1\expandafter#2\expandafter{#2#3}%
49 }
```

15.2 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.

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

`\stitlefont` Define the font style to use for formatting song titles.

```
51 \newcommand\stitlefont{%
52   \sffamily\ifslides\Huge\else\slshape\Large\fi%
53 }
```

`\versefont` By default, verses, choruses, and textual notes just allow the `\lyricfont` style to
`\chorusfont` continue. Meter numbers are in tiny, sans-serif, upright font. Echo parts toggle
`\notefont` slanted and upright fonts.
`\meterfont`
```
54 \newcommand\versefont{}
55 \newcommand\chorusfont{}
56 \newcommand\notefont{}
57 \newcommand\meterfont{\tiny\sffamily\upshape}
```

`\echofont` Echo parts 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.

```
58 \newcommand\echofont{%
59   \ifdim\fontdimen\@ne\font>\z@\upshape\else\slshape\fi%
60 }
```

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

```
61 \newcommand\scripturefont{%
62   \usefont{OT1}{pzc}{mb}{it}%
63   \shiftdblquotes{-1.1\p@}\z@{-2\p@}\z@%
64 }
```

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

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

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

```
66 \newcommand\snumbgcolor{SongbookShade}
67 \newcommand\notebgcolor{SongbookShade}
68 \newcommand\idxbgcolor{SongbookShade}
```

`\versejustify`	Verses and choruses are both left-justified with hanging indentation equal to
`\chorusjustify`	`\parindent`.

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

`\notejustify` Textual notes are fully justified when they are too long to fit in a single line.

```
71 \newcommand\notejustify{%
72   \advance\baselineskip\p@\relax%
73   \leftskip\z@skip\rightskip\z@skip%
74   \parfillskip\@flushglue\parindent\z@%
75 }
```

`\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`.

```
76 \newcommand\placenote[1]{%
77   \leftskip\z@skip\rightskip\@flushglue\SB@cbarshift%
78   \noindent#1\par%
79 }
```

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

```
80 \newcounter{songnum}
81 \newcounter{versenum}
```

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

```
82 \renewcommand\thesongnum{\songnumstyle{songnum}}
83 \newcommand\songnumstyle{}
84 \let\songnumstyle\arabic
```

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

```
85 \renewcommand\theversenum{\versenumstyle{versenum}}
86 \newcommand\versenumstyle{}
87 \let\versenumstyle\arabic
```

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

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

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

```
89 \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.

```
90 \newcommand\placeversenum[1]{%
91   \hskip-\leftskip\hskip-\parindent\relax%
92   \box#1%
93 }
```

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

```
94 \newcommand\everyverse{}
95 \newcommand\everychorus{}
```

\printchord Define the printing style for chords.

```
96 \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.

```
97 \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.

```
98 \newlength\versesep
99 \versesep123456789sp\relax
```

\afterpreludeskip Users can specify the amount of vertical space that separates song prelude and
\beforepostludeskip postlude material from the body of the song by adjusting the following two macros.

```
100 \newlength\afterpreludeskip
101 \afterpreludeskip=2\p@\@plus4\p@
102 \newlength\beforepostludeskip
103 \beforepostludeskip=2\p@\@plus4\p@
```

\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.

```
104 \newlength\baselineadj
105 \baselineadj\z@skip
```

\clineparams The spacing between chords and the lyrics below them can be adjusted by changing
the values of \baselineskip, \lineskiplimit, and \lineskip within the follow-
ing macro. By default, \baselineskip is set to 2 points smaller than the height
of the current (lyric) font, and \lineskiplimit and \lineskip are set so that
chords intrude at most 2 points into the lyric below them. This helps to keep
chords tight with lyrics.

```
106 \newcommand\clineparams{%
107   \baselineskip\f@size\p@%
108   \advance\baselineskip-2\p@%
109   \lineskiplimit-2\p@%
110   \lineskip-2\p@%
111 }
```

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

```
112 \parindent.25in
```

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

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

\songnumwidth Set the width of the song number boxes that begin each song. We guess a suitable width by typesetting the text "999."

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

\versenumwidth Set the width that is reserved for normal-sized verse numbers. (Verse numbers wider than this will indent the first line of lyrics.)

```
117 \newlength\versenumwidth
118 \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.

```
119 \newlength\cbarwidth
120 \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.

```
121 \newlength\sbarheight
122 \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 (1000).

```
123 \interlinepenalty\@m
```

\vvpenalty
\ccpenalty
\vcpenalty
\cvpenalty
\brkpenalty
The following count registers define the line-breaking 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 $1000 - 800 = 200$.

```
124 \SB@newcount\vvpenalty\vvpenalty200
125 \SB@newcount\ccpenalty\ccpenalty200
126 \SB@newcount\vcpenalty\vcpenalty200
127 \SB@newcount\cvpenalty\cvpenalty200
128 \SB@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.

```
129 \SB@newcount\spenalty\spenalty100
```

\songmark
\versemark
\chorusmark

The user can redefine the following macros to add TeX marks for each song, each verse, or each chorus. Such marks are used by LaTeX to define page headers and footers.

```
130 \newcommand\songmark{}
131 \newcommand\versemark{}
132 \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.

```
133 \newcommand\extendprelude{\showrefs\showauthors}
134 \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.

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

\idxtitlefont

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

```
136 \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.

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

\idxscripfont

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

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

\idxauthfont

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

```
139 \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.

```
140 \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.

```
141 \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.

```
142 \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.

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

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

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

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

```
147 \SB@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.

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

15.3   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 §**??** 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.

```
149 \DeclareOption{slides}{\slides}
150 \newcommand\slides{%
151   \slidestrue%
152   \def\lyricfont{\normalfont\huge}%
153   \def\chorusfont{\slshape}%
154   \def\versejustify{\justifycenter}%
155   \let\chorusjustify\versejustify
156   \def\placenote##1{\justifycenter\noindent##1\par}%
157   \scriptureoff%
158   \onesongcolumn%
159   \ifSB@preamble\ifSB@chordedspec\else\SB@chordsoff\fi\fi%
160   \spenalty-\@M%
161   \let\colbotglue\@flushglue%
162   \setlength\cbarwidth\z@%
163   \setlength\sbarheight\z@%
164 }
```

\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.

```
165 \newcommand\justifyleft{%
166   \leftskip\parindent%
167   \ifSB@inverse\advance\leftskip\versenumwidth\fi%
168   \SB@cbarshift%
169   \parindent-\parindent%
170 }
```

\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.

```
171 \newcommand\justifycenter{%
172   \centering\SB@cbarshift\rightskip\leftskip%
173   \def\placeversenum##1{%
174     \hskip-\leftskip\hskip-\parindent\relax%
175     \hangindent-\wd##1\hangafter\m@ne%
176     \box##1\hfil%
177   }%
178 }
```

unouter *(Default: off)*   Several macros provided by the songs package are, by default,
\SB@outer 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`.

```
179 \newcommand\SB@outer{\outer}
180 \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.)

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

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.)

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

noindexes *(Default: off)*   Suppress generation of index files and displaying of in-document
\indexeson indexes. The `\indexeson` and `\indexesoff` macros can be used elsewhere to
\indexesoff toggle display of indexes. Index-regeneration will occur if indexes are turned on by the end of the document.

```
183 \DeclareOption{noindexes}{\indexesoff}
184 \newcommand\indexeson{\songindexestrue}
185 \newcommand\indexesoff{\songindexesfalse}
```

nopdfindex *(Default: off)*   Suppress creation of PDF bookmark entries and hyperlinks.

```
186 \DeclareOption{nopdfindex}{%
187   \let\songtarget\@gobbletwo%
188   \let\songlink\@secondoftwo%
189 }
```

`\ifSB@measurespec` The `showmeasures` and `chorded` options interact in the sense that by default,
`\ifSB@chordedspec` 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 whether each of these options has been explicitly specified by the user or whether it is still in a default state.

```
190 \newif\ifSB@measurespec
191 \newif\ifSB@chordedspec
```

`chorded` *(Default: chorded)*   Determines whether chords should be shown. This option
`lyric` can be set in the `\usepackage` line or toggled elsewhere with the `\chordson` and
`\chordson` `\chordsoff` macros. Chords cannot be turned on in conjunction with the `rawtext`
`\chordsoff` option. If chords are turned on by the end of the preamble, no attempt will be
`\SB@chordson` made to balance columns on each page.
`\SB@chordsoff`
```
192 \DeclareOption{chorded}{\chordson}
193 \DeclareOption{lyric}{\chordsoff}
194 \newcommand\chordson{\SB@chordedspectrue\SB@chordson}
195 \newcommand\chordsoff{\SB@chordedspectrue\SB@chordsoff}
196 \newcommand\SB@chordson{%
197   \ifrawtext%
198     \SB@errrtopt%
199   \else%
200     \chordedtrue\lyricfalse%
201     \let\SB@bracket\SB@chord%
202     \let\SB@rechord\SB@@rechord%
203     \let\SB@ch\SB@ch@on%
204     \ifSB@measurespec%
205       \ifmeasures\SB@measureson\else\SB@measuresoff\fi%
206     \else%
207       \SB@measureson%
208     \fi%
209     \ifSB@preamble\def\colbotglue{\z@\@plus.5\textheight}\fi%
210     \SB@setbaselineskip%
211   \fi%
212 }
213 \newcommand\SB@chordsoff{%
214   \chordedfalse\lyrictrue%
215   \def\SB@bracket##1]{\ignorespaces}%
216   \let\SB@rechord\relax%
217   \let\SB@ch\SB@ch@off%
218   \ifSB@measurespec%
219     \ifmeasures\SB@measureson\else\SB@measuresoff\fi%
220   \else%
221     \SB@measuresoff%
222   \fi%
223   \ifSB@preamble\let\colbotglue\z@skip\fi%
224   \SB@setbaselineskip%
225 }
```

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

```
226 \DeclareOption{showmeasures}{\measureson}
227 \DeclareOption{nomeasures}{\measuresoff}
228 \newcommand\measureson{\SB@measurespectrue\SB@measureson}
229 \newcommand\measuresoff{\SB@measurespectrue\SB@measuresoff}
230 \newcommand\SB@measureson{%
231   \measurestrue%
232   \let\SB@mbar\SB@makembar%
233   \ifchorded%
234     \let\SB@mch\SB@mch@on%
235   \else%
236     \let\SB@mch\SB@mch@m%
237   \fi%
238   \ifSB@inverse\SB@loadactives\fi%
239   \ifSB@inchorus\SB@loadactives\fi%
240 }
241 \newcommand\SB@measuresoff{%
242   \measuresfalse%
243   \let\SB@mbar\@gobbletwo%
244   \ifchorded%
245     \let\SB@mch\SB@ch@on%
246   \else%
247     \let\SB@mch\SB@ch@off%
248   \fi%
249   \ifSB@inverse\SB@loadactives\fi%
250   \ifSB@inchorus\SB@loadactives\fi%
251 }
```

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.

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

noscripture *(Default: off)* Inhibits the display of scripture quotes. This option can also be
\scriptureon toggled on and off anywhere with the `\scriptureon` and `\scriptureoff` macros.
\scriptureoff
```
253 \DeclareOption{noscripture}{\SB@omitscriptrue}
254 \newcommand\scriptureon{\SB@omitscripfalse}
255 \newcommand\scriptureoff{\SB@omitscriptrue}
```

onesongcolumn *(Default: onesongcolumn is the default if generating slides or rawtext, twosong-*
twosongcolumns *columns otherwise)* The number of columns per page is specified using the
\onesongcolumn following package options and macros. In `rawtext` mode it must remain set to one
\twosongcolumns column per page. The entire page-making system can be turned off by setting the
\songcolumns 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.

```
256 \DeclareOption{twosongcolumns}{\SB@numcols\tw@}
257 \DeclareOption{onesongcolumn}{\SB@numcols\@ne}
258 \newcommand\songcolumns[1]{%
259   \SB@cnt#1\relax%
260   \ifnum\SB@cnt=\SB@numcols\else%
261     \ifSB@preamble\else{\SB@clearpage}\fi%
```

```
262    \fi%
263    \SB@numcols\SB@cnt%
264    \ifnum\SB@numcols>\z@%
265      \SB@colwidth-\columnsep%
266      \multiply\SB@colwidth\SB@numcols%
267      \advance\SB@colwidth\columnsep%
268      \advance\SB@colwidth\textwidth%
269      \divide\SB@colwidth\SB@numcols%
270    \else%
271      \ifrepchorus\SB@warnrc\fi%
272    \fi%
273 }
274 \newcommand\onesongcolumn{\songcolumns\@ne}
275 \newcommand\twosongcolumns{\songcolumns\tw@}
```

\includeonlysongs   Display only a select list of songs and ignore the rest.
\songlist
```
276 \newcommand\songlist{}
277 \newcommand\includeonlysongs[1]{%
278    \ifSB@songsenv\SB@errpl\else%
279      \partiallisttrue%
280      \renewcommand\songlist{#1}%
281    \fi%
282 }
```

\nosongnumbers   The user can turn off song numbering with the following macro.
```
283 \newcommand\nosongnumbers{\setlength\songnumwidth\z@}
```

\noversenumbers   The user can turn off verse numbering with the following macro.
```
284 \newcommand\noversenumbers{%
285    \renewcommand\printversenum[1]{}%
286    \setlength\versenumwidth\z@%
287 }
```

\repchoruses   Using \repchoruses causes choruses to be automatically repeated on subsequent
\norepchoruses   pages of the song. The feature requires ε-TeX because the supporting code needs
an extended mark register class.
```
288 \ifSB@etex
289   \newcommand\repchoruses{%
290     \ifnum\SB@numcols<\@ne\SB@warnrc\fi%
291     \repchorustrue%
292   }
293 \else
294   \newcommand\repchoruses{\SB@erretex}
295 \fi
296 \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.
```
297 \newcommand\sepverses{%
298    \vvpenalty-\@M%
299    \ccpenalty100 %
300    \vcpenalty\vvpenalty%
```

```
301    \cvpenalty\vvpenalty%
302    \let\colbotglue\@flushglue%
303 }
```

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

```
304 \AtBeginDocument{
```

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

```
305    \SB@setversesep
```

Initialize page layout algorithm.

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

Macros used after this point occur outside the preamble.

```
307    \SB@preamblefalse
308 }
```

15.4 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.

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

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

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

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

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

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

```
313 \SB@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.

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

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

```
315 \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. an integer (positive or zero) indicating whether box *b* should be fully emptied and committed as columns (if positive), or whether its final less-than-column-height remainder should be reserved as an in-progress column (if zero);

2. the box *b* to split;

3. a count register *i* equaling the column index (zero or greater) where the content of *b* is to begin; and

4. the desired column height.

Box *b* is split and *i* is incremented until *i* reaches `\SB@numcols` or *b* is emptied. If *b* is emptied and the first argument is 0, 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. Otherwise, if *b* is emptied and the first argument is positive, the final column is contributed and *i* is set to one greater than the index of that column. (If *i* reaches `\SB@numcols` before *b* is emptied, the first argument is ignored.)

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.

The implementation takes two special steps to avoid pre-committing in-progress columns (when the first macro argument is zero): First, the final split that empties box *b* is "undone" by reverting to a backup copy made before each split. Second, any underfull box warnings for this final split are suppressed by temporarily adding infinite-stretch `\vfil` glue to the bottom of the box. This strategy preserves underfull and overfull box warnings for the columns that are actually committed, but suppresses faux warnings for the last split that is undone.

```
316 \newcommand\SB@mkpage[4]{%
317   \ifvoid#2\else\begingroup%
318     \edef\SB@temp{\ifnum#2=\SB@box\SB@boxii\else\SB@box\fi}%
319     \edef\SB@tempii{\ifnum#2=\SB@boxiii\SB@boxii\else\SB@boxiii\fi}%
320     \splitmaxdepth\maxdepth\splittopskip\z@skip%
321     \ifnum#1=\z@\global\setbox#2\vbox{\unvbox#2\vfil}\fi%
322     \loop\ifnum#3<\SB@numcols%
323       \ifnum#1=\z@\setbox\SB@tempii\copy#2\fi%
324       \setbox\SB@temp\vsplit#2to#4\relax%
325       \ifvoid#2%
326         \ifnum#1=\z@%
327           \global\setbox#2\box\SB@tempii%
328         \else%
329           \SB@updatepage%
330           \global\advance#3\@ne%
331         \fi%
332         #3\SB@numcols%
333       \else%
334         \SB@updatepage%
335         \global\advance#3\@ne%
336         \ifrepchorus\ifvoid\SB@chorusbox\else%
337           \SB@insertchorus#2%
338         \fi\fi%
339       \fi%
```

```
340        \repeat%
341        \ifnum#1=\z@\global\setbox#2\vbox{\unvbox#2\unskip}\fi%
342    \endgroup\fi%
343 }
```

\SB@migrate Migrate a mark out of a recently split vertical list, but do not insert superfluous
empty marks that may override previous marks.

```
344 \newcommand\SB@migrate[1]{%
345    \SB@toks\expandafter{#1}%
346    \edef\SB@temp{\the\SB@toks}%
347    \ifx\SB@temp\@empty\else\mark{\the\SB@toks}\fi%
348 }
```

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

```
349 \newcommand\SB@updatepage{%
350    \global\setbox\SB@mrkbox\vbox{%
351        \unvbox\SB@mrkbox%
352        \SB@migrate\splitfirstmark%
353        \SB@migrate\splitbotmark%
354    }%
355    \global\setbox\SB@pgbox\hbox{%
356        \SB@dimen\SB@colwidth%
357        \advance\SB@dimen\columnsep%
358        \multiply\SB@dimen\SB@colnum%
359        \advance\SB@dimen-\wd\SB@pgbox%
360        \unhbox\SB@pgbox%
361        \ifdim\SB@dimen=\z@\else\hskip\SB@dimen\relax\fi%
362        \box\SB@temp%
363    }%
364 }
```

\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.

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

\SB@output This is the main output routine for the page-builder. It repeatedly calls \SB@mkpage,
emitting pages as they are completed, until the remaining content of box
\SB@colbox is not enough to fill a column. If the macro argument is 0, this
final, in-progress column is left unfinished, pending future contributions. If the
argument is positive, the final material is committed as a column. If the argument
is two or greater, the entire in-progress page is also committed and the column
number reset.

```
366 \newcommand\SB@output[1]{%
367    \ifnum\SB@numcols>\z@\begingroup%
368        \loop%
369            \SB@dimen\textheight%
370            \ifinner\else\advance\SB@dimen-\pagetotal\fi%
371            \SB@mkpage#1\SB@colbox\SB@colnum\SB@dimen%
372            \SB@testfalse\SB@testiitrue%
373            \ifnum#1>\@ne\ifvoid\SB@colbox\ifnum\SB@colnum>\z@%
374                \SB@testtrue\SB@testiifalse%
```

```
375        \fi\fi\fi%
376        \ifnum\SB@colnum<\SB@numcols\SB@testiifalse\else\SB@testtrue\fi%
377        \ifSB@test%
378          \unvbox\SB@mrkbox%
379          \ifinner\else\kern\z@\fi%
380          \box\SB@pgbox%
381          \ifinner\else\vfil\break\vskip\vsize\relax\fi%
382          \global\SB@colnum\z@%
383        \fi%
384      \ifSB@testii\repeat%
385    \endgroup\else%
386      \unvbox\SB@colbox\unskip%
387    \fi%
388 }
```

<code>\SB@putboxes</code> 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.

```
389 \newcommand\SB@putboxes[1]{%
390    \SB@dimen\ifnum\SB@numcols>\z@\ht\SB@colbox\else\p@\fi%
391    #1\SB@colbox%
392    \ifdim\SB@dimen>\z@%
393      \SB@breakpoint\spenalty%
394      \ifdim\sbarheight>\z@%
395        \vskip-\sbarheight\relax%
396      \fi%
397    \fi%
398    #1\SB@songbox%
399 }
```

<code>\SB@nextcol</code> 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 <code>\@flushglue</code>, this forces a break that leaves whitespace at the bottom of the column. When it's <code>\colbotglue</code>, it acts like a natural column break chosen by the page-breaker. However, if the current column is empty, <code>\@flushglue</code> is always used so that an empty column will result.

```
400 \newcommand\SB@nextcol[2]{%
401    \ifnum#1>\z@%
402      \ifnum\SB@numcols>\z@%
403        \global\setbox\SB@colbox\vbox{%
404          \SB@cnt#1\relax%
405          \SB@dimen\ht\SB@colbox%
406          \unvbox\SB@colbox%
407          \unskip%
408          \ifdim\SB@dimen>\z@%
409            \vskip#2\relax%
410            \break%
411            \advance\SB@cnt\m@ne%
412          \fi%
413          \loop\ifnum\SB@cnt>\z@%
414            \nointerlineskip%
415            \null%
416            \vfil%
```

```
417          \break%
418          \advance\SB@cnt\m@ne%
419        \repeat%
420      }%
421      \SB@output1%
422    \else%
423      \ifnum\lastpenalty=-\@M\null\fi%
424      \break%
425    \fi%
426  \fi%
427 }
```

\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).

```
428 \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. The number of column breaks is determined by temporarily setting \SB@updatepage to \SB@droppage and then calling the \SB@mkpage algorithm under various conditions to see how many columns it would contribute if we start the current song at various positions.

```
429 \newcommand\SB@@selectcol{%
430   \begingroup%
431     \SB@cnt\z@%
432     \vbadness\@M\vfuzz\maxdimen%
433     \let\SB@updatepage\SB@droppage%
434     \SB@dimen\textheight%
435     \ifinner\else\advance\SB@dimen-\pagetotal\fi%
436     \setbox\SB@boxii\vbox{\SB@putboxes\unvcopy}%
437     \SB@cntii\SB@colnum%
438     \SB@mkpage0\SB@boxii\SB@cntii\SB@dimen%
439     \SB@spos%
440     \global\SB@cnt\SB@cnt%
441   \endgroup%
442   \SB@nextcol\SB@cnt\colbotglue%
443 }
```

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

```
444 \newcommand\SB@spbegnew{%
445   \setbox\SB@boxiii\copy\SB@songbox%
446   \SB@cntii\z@%
447   \SB@mkpage0\SB@boxiii\SB@cntii\textheight%
448 }
```

\SB@spextold Tentatively extend the song previously typeset on the current even page to the next odd page to see whether 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.

```
449 \newcommand\SB@spextold{%
450   \ifodd\c@page\else%
451     \SB@cntii\z@%
452     \SB@mkpage0\SB@boxii\SB@cntii\textheight%
453   \fi%
454 }
```

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

```
455 \newcommand\SB@spextnew{%
456   \SB@cntii\z@%
457   \SB@mkpage0\SB@boxiii\SB@cntii\textheight%
458 }
```

\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.

```
459 \newcommand\SB@spdblpg{%
460   \ifnum\SB@cntii<\SB@numcols%
461     \SB@cnt\SB@numcols%
462     \advance\SB@cnt-\SB@colnum%
463     \if@twoside\ifodd\c@page\else%
464       \advance\SB@cnt\SB@numcols%
465     \fi\fi%
466   \fi%
467 }
```

\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.

```
468 \newcommand\SB@sposi{%
469   \ifnum\SB@cntii<\SB@numcols\else\if@twoside%
470     \SB@spextold%
471   \fi\fi%
472   \ifnum\SB@cntii<\SB@numcols\else%
473     \SB@spbegnew%
474     \ifnum\SB@cntii<\SB@numcols\else\if@twoside%
475       \SB@spextnew%
476     \fi\fi%
477     \SB@spdblpg%
478   \fi%
479 }
```

\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.

```
480 \newcommand\SB@sposii{%
481   \ifnum\SB@cntii<\SB@numcols\else%
482     \SB@spbegnew%
483     \ifnum\SB@cntii<\SB@numcols%
484       \SB@cnt\SB@numcols%
485       \advance\SB@cnt-\SB@colnum%
```

```
486    \else%
487      \if@twoside%
488        \SB@spextold%
489        \ifnum\SB@cntii<\SB@numcols\else%
490          \SB@spextnew%
491          \SB@spdblpg%
492        \fi%
493      \fi%
494    \fi%
495  \fi%
496 }
```

\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.

```
497 \newcommand\SB@sposiii{%
498   \ifnum\SB@cntii>\SB@colnum%
499     \SB@cnt\SB@colnum%
500     \advance\SB@cnt\@ne%
501     \ifnum\SB@cnt<\SB@numcols%
502       \setbox\SB@boxiii\copy\SB@songbox%
503       \SB@mkpage0\SB@boxiii\SB@cnt\SB@dimen%
504       \advance\SB@cnt\m@ne%
505     \fi%
506     \ifnum\SB@cnt>\SB@colnum%
507       \SB@cnt\z@%
508       \SB@sposii%
509     \else%
510       \SB@cnt\@ne%
511     \fi%
512   \fi%
513 }
```

\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.

```
514 \newcommand\songpos[1]{%
515   \ifcase#1%
516     \let\SB@selectcol\relax%
517     \let\SB@spos\relax%
518   \or%
519     \let\SB@selectcol\SB@@selectcol%
520     \let\SB@spos\SB@sposi%
521   \or%
522     \let\SB@selectcol\SB@@selectcol%
523     \let\SB@spos\SB@sposii%
524   \or%
525     \let\SB@selectcol\SB@@selectcol%
526     \let\SB@spos\SB@sposiii%
527   \else%
528     \SB@errspos%
529   \fi%
530 }
```

`\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.

```
531 \newcommand\SB@spos{}
532 \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). The `\SB@colbox` is tested for zero height and depth rather than voidness, since sometimes it contains zero-length `\splittopskip` glue.

```
533 \newcommand\SB@clearpage{%
534   \SB@testtrue%
535   \ifvoid\SB@pgbox%
536     \ifdim\ht\SB@colbox=\z@\ifdim\dp\SB@colbox=\z@%
537       \SB@testfalse%
538     \fi\fi%
539   \fi%
540   \ifSB@test%
541     \SB@cnt\SB@numcols%
542     \advance\SB@cnt-\SB@colnum%
543     \SB@nextcol\SB@cnt\lastcolglue%
544     \SB@output2%
545   \fi%
546 }
```

`\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.

```
547 \newcommand\SB@cleardpage{%
548   \SB@clearpage%
549   \if@twoside\ifodd\c@page%
550     \SB@nextcol\SB@numcols\@flushglue%
551     \SB@output2%
552   \fi\fi%
553 }
```

`\SB@stype` There are two song content submission types: column- and page-submissions. Page-submissions are page-width and go atop fresh pages unless the current page has only page-width material so far. Column-submissions are column-width and start a new page only when the current page is full. This macro gets set to the desired type for the current submission. Mostly it stays set to the default column-submission type.

```
554 \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.

```
555 \newcommand\SB@stypcol{%
556   \ifnum\SB@numcols>\z@%
557     \SB@selectcol%
558     \global\setbox\SB@colbox\vbox{\SB@putboxes\unvbox}%
559     \SB@output0%
560   \else%
```

```
561      \unvbox\voidb@x%
562      \SB@breakpoint\spenalty%
563      \ifdim\sbarheight>\z@%
564        \vskip-\sbarheight\relax%
565      \fi%
566      \unvbox\SB@songbox%
567    \fi%
568 }
```

\SB@styppage Page-submissions go directly to the top of the nearest fresh page unless the current page has all page-width material so far.

Implementation notes: The `\null` is needed because the page builder consults `\pagetotal`, which isn't updated by TeX until a box is contributed (`\unvbox` doesn't count). Both `\nointerlineskip`s are needed because `\unvbox` fails to update `\prevdepth`, and it doesn't make sense to inherit its value from whatever preceeded this contribution. Authors who want interline glue must therefore insert it explicitly at the bottom of their contributed text.

```
569 \newcommand\SB@styppage{%
570   \ifnum\SB@numcols>\z@%
571     \SB@clearpage%
572     \unvbox\SB@songbox%
573     \nointerlineskip\null%
574   \else%
575     \unvbox\SB@songbox%
576   \fi%
577   \nointerlineskip%
578 }
```

\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`.

```
579 \newcommand\SB@sgroup{}
580 \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.

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

\SB@clearpboxes This dynamically constructed macro clears the content of all boxes created by the workings of `\includeonlysongs`.

```
582 \newcommand\SB@clearpboxes{}
```

\SB@partbox Save a box of full-song or chorus material for later output when producing a partial list using `\includeonlysongs`.

```
583 \newcommand\SB@partbox[1]{%
584   \SB@newbox#1%
585   \SB@app\gdef\SB@clearpboxes{\setbox#1\box\voidb@x}%
586   \global\setbox#1\box%
587 }
```

\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.

```
588 \newcommand\SB@submitpart{%
589   \ifx\SB@sgroup\@empty\else%
590     \SB@testfalse
591     \@for\SB@temp:=\songlist\do{\ifx\SB@temp\SB@sgroup\SB@testtrue\fi}%
592     \ifSB@test%
593       \edef\SB@tempii{\SB@sgroup @\the\SB@groupcnt}%
594       \expandafter\SB@partbox
595         \csname songbox@\SB@tempii\endcsname\SB@songbox%
596       \global\expandafter\let%
597         \csname stype@\SB@tempii\endcsname\SB@stype%
598       \ifrepchorus\ifvoid\SB@chorusbox\else%
599         \expandafter\SB@partbox
600 \csname chbox@\SB@tempii\endcsname\SB@chorusbox%
601       \fi\fi%
602     \fi%
603     \global\advance\SB@groupcnt%
604       \ifnum\SB@groupcnt<\z@\m@ne\else\@ne\fi%
605   \fi%
606   \setbox\SB@songbox\box\voidb@x%
607   \setbox\SB@chorusbox\box\voidb@x%
608 }
```

\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.)

```
609 \newcommand\SB@submitsong{%
610   \ifpartiallist\SB@submitpart\else\SB@stype\fi%
611 }
```

\SB@submitenv Submit the \SB@envbox box as a page-width contribution.

```
612 \newcommand\SB@submitenv{%
613   \begingroup%
614     \let\SB@songbox\SB@envbox%
615     \SB@styppage%
616   \endgroup%
617 }
```

\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.

```
618 \newcommand*\SB@songlistbrk{brk}
619 \newcommand*\SB@songlistnc{nextcol}
620 \newcommand*\SB@songlistcp{sclearpage}
621 \newcommand*\SB@songlistcdp{scleardpage}
```

\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.

```
622 \newcommand\commitsongs{%
623   \ifpartiallist%
624     \ifnum\SB@numcols>\z@%
625       \@for\SB@temp:=\songlist\do{%
626         \ifx\SB@temp\SB@songlistnc\SB@nextcol\@ne\@flushglue\else%
627         \ifx\SB@temp\SB@songlistbrk\SB@nextcol\@ne\colbotglue\else%
628         \ifx\SB@temp\SB@songlistcp\SB@clearpage\else%
629         \ifx\SB@temp\SB@songlistcdp\SB@cleardpage\else%
630           \SB@groupcnt\m@ne\SB@finloop%
631           \SB@groupcnt\z@\SB@finloop%
632         \fi\fi\fi\fi%
633       }%
634     \else%
635       \@for\SB@temp:=\songlist\do{%
636         \ifx\SB@temp\SB@songlistnc\vfil\break\else%
637         \ifx\SB@temp\SB@songlistbrk\break\else%
638         \ifx\SB@temp\SB@songlistcp\clearpage\else%
639         \ifx\SB@temp\SB@songlistcdp%
640           \clearpage%
641           \ifodd\c@page\null\newpage\fi%
642         \else%
643           \SB@groupcnt\m@ne\SB@finloop%
644           \SB@groupcnt\z@\SB@finloop%
645         \fi\fi\fi\fi%
646       }%
647     \fi%
648     \SB@clearpboxes%
649   \fi%
650   \SB@clearpage%
651 }
```

\SB@finloop While contributing saved material included by **\includeonlysongs**, this macro contributes each series of boxes grouped together as part of a **songgroup** environment.

```
652 \newcommand\SB@finloop{%
653   \loop\edef\SB@tempii{\SB@temp @\the\SB@groupcnt}%
654       \expandafter\ifx%
655         \csname songbox@\SB@tempii\endcsname\relax\else%
656   \setbox\SB@songbox\expandafter\copy%
657       \csname songbox@\SB@tempii\endcsname%
658   \expandafter\ifx\csname chbox@\SB@tempii\endcsname\relax%
659     \repchorusfalse%
660   \else%
661     \repchorustrue%
662     \setbox\SB@chorusbox\expandafter\copy%
663       \csname chbox@\SB@tempii\endcsname%
664   \fi%
665   \csname stype@\SB@tempii\endcsname%
666   \advance\SB@groupcnt\ifnum\SB@groupcnt<\z@\m@ne\else\@ne\fi%
667   \repeat%
668 }
```

67

Insert a chorus into the first marked spot in the 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.

```
669 \newcommand\SB@insertchorus[1]{{%
670   \vbadness\@M\vfuzz\maxdimen%
671   \setbox\SB@box\copy#1%
672   \setbox\SB@box\vsplit\SB@box to\maxdimen%
673   \edef\SB@temp{\splitfirstmarks\SB@nocmarkclass}%
674   \ifx\SB@temp\SB@nocmark\else%
675     \edef\SB@temp{\splitfirstmarks\SB@cmarkclass}%
676     \ifx\SB@temp\SB@cmark%
677       \SB@dimen4096\p@%
678       \SB@dimenii\maxdimen%
679       \SB@dimeniii\SB@dimen%
680       \loop%
681         \SB@dimeniii.5\SB@dimeniii%
682         \setbox\SB@box\copy#1%
683         \setbox\SB@box\vsplit\SB@box to\SB@dimen%
684         \edef\SB@temp{\splitfirstmarks\SB@cmarkclass}%
685         \ifx\SB@temp\SB@cmark%
686           \SB@dimenii\SB@dimen%
687           \advance\SB@dimen-\SB@dimeniii%
688         \else%
689           \advance\SB@dimen\SB@dimeniii%
690         \fi%
691       \ifdim\SB@dimeniii>2\p@\repeat%
692       \setbox\SB@box\vsplit#1to\SB@dimenii%
693       \global\setbox#1\vbox{%
694         \unvbox\SB@box\unskip%
695         \SB@inversefalse\SB@prevversetrue\SB@stanzabreak%
696         \SB@putbox\unvcopy\SB@chorusbox%
697         \SB@inversetrue\SB@prevversefalse\SB@stanzabreak%
698         \unvbox#1%
699       }%
```

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.

```
700     \else\ifx\SB@temp\SB@lastcmark%
701       \global\setbox#1\vbox{%
702         \unvbox#1%
703         \unskip%
704         \ifdim\sbarheight>\z@%
705           \setbox\SB@box\lastbox%
706           \unskip\unpenalty%
707         \fi%
708         \setbox\SB@box\lastbox%
```

68

```
709        \unskip\unskip%
710        \SB@inversefalse\SB@prevversetrue\SB@stanzabreak%
711        \marks\SB@nocmarkclass{\SB@nocmark}%
712        \unvcopy\SB@chorusbox%
713        \vskip\versesep\vskip\beforepostludeskip\relax%
714        \nointerlineskip\box\SB@box%
715        \ifdim\sbarheight>\z@%
716          \nobreak\vskip2\p@\@plus\p@%
717          \hrule\@height\sbarheight\@width\SB@colwidth%
718        \fi%
719      }%
720    \fi\fi%
721  \fi%
722 }}
```

\nextcol End the current column (inserting vertical space as needed). This differs from column breaks produced with \brk, which does not introduce any empty vertical space.

```
723 \newcommand\nextcol{%
724   \@ifstar{\SB@nextcol\@ne\@flushglue}%
725           {\ifpartiallist\else\SB@nextcol\@ne\@flushglue\fi}%
726 }
```

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

```
727 \newcommand\sclearpage{%
728   \@ifstar\SB@clearpage{\ifpartiallist\else\SB@clearpage\fi}%
729 }
```

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

```
730 \newcommand\scleardpage{%
731   \@ifstar\SB@cleardpage{\ifpartiallist\else\SB@cleardpage\fi}%
732 }
```

15.5 Songs

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

\SB@lop The following macros were adapted from Donald Knuth's *The TEXbook*, for manip-
\SB@@lop ulating lists of the form *item1**item2**...**itemN*\\.
\SB@emptylist
\SB@ifempty

```
733 \newcommand\SB@lop[1]{\expandafter\SB@@lop\the#1\SB@@lop#1}
734 \newcommand\SB@@lop{}
735 \def\SB@@lop\\#1\\#2\SB@@lop#3#4{\global#3{\\#2}\global#4{#1}}
736 \newcommand*\SB@emptylist{\\}
737 \newcommand\SB@ifempty[3]{%
738   \edef\SB@temp{\the#1}%
739   \ifx\SB@temp\SB@emptylist#2\else#3\fi%
740 }
```

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

```
741 \SB@newtoks\SB@titlelist
742 \SB@newtoks\SB@titletail
```

69

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

              743 \newcommand\songtitle{}

\resettitles Initialize the title list iterator.

              744 \newcommand\resettitles{%
              745   \global\SB@titletail\SB@titlelist%
              746   \nexttitle%
              747 }

\nexttitle   Advance the title list iterator to the next title.

              748 \newcommand\nexttitle{%
              749   \SB@ifempty\SB@titletail{%
              750   \global\let\songtitle\relax%
              751   }{%
              752   \SB@lop\SB@titletail\SB@toks%
              753   \edef\songtitle{\the\SB@toks}%
              754   }%
              755 }

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

              756 \newcommand\foreachtitle[1]{%
              757   \ifx\songtitle\relax\else%
              758   \loop#1\nexttitle\ifx\songtitle\relax\else\repeat%
              759   \fi%
              760 }

\ifSB@insong   To help the user locate errors, keep track of which environments we're inside
\ifSB@intersong and immediately signal an error if someone tries to use a song command inside a
\ifSB@inverse   scripture quotation, etc.
\ifSB@inchorus 761 \newif\ifSB@songsenv\SB@songsenvfalse
                762 \newif\ifSB@insong\SB@insongfalse
                763 \newif\ifSB@intersong\SB@intersongfalse
                764 \newif\ifSB@inverse\SB@inversefalse
                765 \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.

              766 \newcommand\SB@closeall{}

\SB@rawrefs   The current song's scripture references, authors, copyright info, and copyright
\songauthors license information are stored in these macros.
\songcopyright 767 \newcommand\SB@rawrefs{}
\songlicense 768 \newcommand\songauthors{}
              769 \newcommand\songcopyright{}
              770 \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.

              771 \newcommand\songrefs{}

70

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

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

\newsongkey Defining a new key for `\beginsong` is just like the `keyval` package's `\define@key`
\SB@clearbskeys 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.)

```
773 \newcommand\SB@clearbskeys{}
774 \newcommand\newsongkey[2]{%
775   \SB@app\gdef\SB@clearbskeys{#2}%
776   \define@key{beginsong}{#1}%
777 }
```

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.

```
778 \newsongkey{sr}{\def\SB@rawrefs{}\gdef\songrefs{}}
779              {\def\SB@rawrefs{#1}\SB@parsesrefs{#1}}
780 \newsongkey{by}{\def\songauthors{}}{\def\songauthors{#1}}
781 \newsongkey{cr}{\def\songcopyright{}}{\def\songcopyright{#1}}
782 \newsongkey{li}{\setlicense{}}{\setlicense{#1}}
783 \newsongkey{index}{}{\indexentry{#1}}
784 \newsongkey{ititle}{}{\indextitleentry{#1}}
```

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

```
785 \newenvironment{song}{\beginsong}{\SB@endsong}
786 \newcommand\beginsong[1]{%
787   \ifSB@insong\SB@errboo\SB@closeall\fi%
788   \ifSB@intersong\SB@errbor\SB@closeall\fi%
789   \SB@insongtrue%
790   \def\SB@closeall{\endsong}%
791   \SB@parsetitles{#1}%
792   \global\setbox\SB@songwrites\box\voidb@x%
793   \SB@clearbskeys%
794   \@ifnextchar[\SB@bskvfmt\SB@@beginsong%
795 }
796 \newcommand\SB@@beginsong{%
797   \@ifnextchar\bgroup\SB@bsoldfmt\SB@@@beginsong%
798 }
799 \newcommand\SB@bsoldfmt[3]{%
800   \SB@bskvfmt[sr={#1},by={#2},cr={#3}]%
801 }
802 \newcommand\SB@bskvfmt{}
803 \def\SB@bskvfmt[#1]{%
804   \setkeys{beginsong}{#1}%
805   \SB@@@beginsong%
806 }
```

\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 reside.

```
807 \newcommand\SB@@@beginsong{%
808   \global\SB@stanzafalse%
809   \setbox\SB@chorusbox\box\voidb@x%
810   \SB@gotchorusfalse%
811   \setbox\SB@songbox\vbox\bgroup\begingroup%
812     \ifnum\SB@numcols>\z@\hsize\SB@colwidth\fi%
813     \leftskip\z@skip\rightskip\z@skip%
814     \parfillskip\@flushglue\parskip\z@skip%
815     \SB@raggedright%
816     \global\SB@transposefactor\z@%
817     \global\SB@cr@{\\}%
818     \protected@edef\@currentlabel{\p@songnum\thesongnum}%
819     \setcounter{versenum}{1}%
820     \SB@prevversetrue%
821     \meter44%
822     \resettitles%
823     \SB@addtoindexes\songtitle\SB@rawrefs\songauthors%
824     \nexttitle%
825     \foreachtitle{\expandafter\SB@addtotitles\expandafter{\songtitle}}%
826     \resettitles%
827     \lyricfont\relax%
828     \SB@setbaselineskip%
829 }
```

\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.

```
830 \newcommand\SB@endsong{%
831   \ifSB@insong%
832       \ifSB@inverse\SB@erreov\endverse\fi%
833       \ifSB@inchorus\SB@erreoc\endchorus\fi%
834       \global\SB@skip\versesep%
835       \unskip%
836       \ifrepchorus\ifvoid\SB@chorusbox\else%
837         \ifSB@prevverse\ifvnumbered%
838           \marks\SB@cmarkclass{\SB@lastcmark}%
839         \fi\fi%
840       \fi\fi%
841     \endgroup\egroup%
842     \begingroup%
843       \ifnum\SB@numcols>\z@%
844         \hsize\ifpagepreludes\textwidth\else\SB@colwidth\fi%
845       \fi%
846       \leftskip\z@skip\rightskip\z@skip%
847       \parfillskip\@flushglue\parskip\z@skip\parindent\z@%
848       \global\setbox\SB@envbox\vbox{%
```

```
849          \songmark%
850          \unvbox\SB@songwrites%
851          \ifpagepreludes\else\ifdim\sbarheight>\z@%
852            \hrule\@height\sbarheight\@width\hsize%
853            \nobreak\vskip5\p@\relax%
854          \fi\fi%
855          \resettitles%
856          \begingroup%
857            \songtarget{\ifnum\c@section=\z@1\else2\fi}%
858                      {song\theSB@songsnum-\thesongnum}%
859          \endgroup%
860          \vbox{\makeprelude}%
861          \nobreak\vskip\SB@skip%
862          \vskip\afterpreludeskip\relax%
863        }%
864      \ifnum\SB@numcols>\z@\hsize\SB@colwidth\fi%
865      \global\setbox\SB@songbox\vbox{%
866        \ifpagepreludes\else\unvbox\SB@envbox\fi%
867        \unvbox\SB@songbox%
868        \nobreak\vskip\SB@skip%
869        \vskip\beforepostludeskip\relax%
870        \nointerlineskip%
871        \vbox{\makepostlude}%
872        \ifdim\sbarheight>\z@%
873          \nobreak\vskip2\p@\@plus\p@%
874          \nointerlineskip%
875          \hbox{\vrule\@height\sbarheight\@width\hsize}%
876        \fi%
877      }%
878    \endgroup%
879    \SB@insongfalse%
880    \edef\SB@sgroup{\thesongnum}%
881    \global\SB@groupcnt\z@%
882    \ifpagepreludes\SB@submitenv\fi%
883    \SB@submitsong%
884    \ifnum\SB@grouplvl=\z@\let\SB@sgroup\@empty\fi%
885    \stepcounter{songnum}%
886  \else%
887    \ifSB@intersong\SB@erreor\SB@closeall%
888    \else\SB@erreot\fi%
889  \fi%
890 }
```

Set the \baselineskip to an appropriate line height.

```
891 \newcommand\SB@setbaselineskip{%
892  \SB@dimen\f@size\p@%
893  \baselineskip\SB@dimen\relax%
894  \ifchorded%
895    \setbox\SB@box\hbox{{\printchord{ABCDEFG\shrp\flt/j7}}}%
896    \advance\baselineskip\ht\SB@box%
897    \advance\baselineskip2\p@%
898  \fi%
899  \ifslides%
900    \advance\baselineskip.2\SB@dimen\@plus.5\SB@dimen%
```

```
901        \@minus.2\SB@dimen%
902    \else%
903      \advance\baselineskip\z@\@plus.1\SB@dimen\relax%
904    \fi%
905    \advance\baselineskip\baselineadj%
906 }
```

\SB@setversesep   Set the \versesep to an appropriate amount if has not already been explicitly set
by the user.

```
907 \newcommand\SB@setversesep{%
908    \SB@dimen123456789sp%
909    \edef\SB@temp{\the\SB@dimen}%
910    \edef\SB@tempii{\the\versesep}%
911    \ifx\SB@temp\SB@tempii%
912      \begingroup%
913        \lyricfont\relax%
914        \SB@dimen\f@size\p@%
915        \ifchorded%
916          \setbox\SB@box\hbox{{\printchord{ABCDEFG\shrp\flt/j7}}}%
917          \advance\SB@dimen\ht\SB@box%
918        \fi%
919        \ifslides%
920          \global\versesep1.2\SB@dimen\@plus.3\SB@dimen%
921          \@minus.3\SB@dimen%
922        \else%
923          \global\versesep.75\SB@dimen\@plus.25\SB@dimen%
924          \@minus.13\SB@dimen%
925        \fi%
926      \endgroup%
927    \fi%
928 }
```

\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.

Note that if you are redefining \makeprelude, you can probably replace
everything below with something much simpler. The code below is lengthy because
it accommodates all of the many different options that various authors may adjust
to customize their books. If you redefine it, you can replace all of this with smaller,
more specialized programming that just outputs the prelude format you desire.

```
929 \newcommand\makeprelude{%
930    \resettitles%
```

In slides mode, the title, references, and authors are simply centered on the page
with no song number. Only the first of the song titles is included. The references
and authors only span the middle 50% of the page, since letting them span the
whole page width stretches them out too much and makes their fine print too hard
to read.

```
931    \ifslides%
932      \hbox to\hsize{{\hfil\stitlefont\relax\songtitle\hfil}}%
933      \vskip5\p@%
934      \hbox to\hsize{%
935        \hfil%
936        \vbox{%
```

```
937        \divide\hsize\tw@\parskip\p@\relax%
938        \centering\small\extendprelude%
939      }%
940      \hfil%
941    }%
942  \else%
```

In non-slides mode, we write the song number in a shaded box to the left (if `\songnumwidth` is positive) and everything else in left-justified paragraphs to the right of it (or centered if `\pagepreludes` is on). The height of the shaded box that contains the song number depends on which is higher: the natural height of the song number, or everything else that goes to the right of it. To find out which is higher, we start by putting the song number in its own box (`\SB@boxii`).

```
943    \ifdim\songnumwidth>\z@%
944      \setbox\SB@boxii\hbox{{\SB@colorbox\snumbgcolor{%
945        \hbox to\songnumwidth{%
946          \printsongnum{\thesongnum}\hfil%
947        }%
948      }}}%
949    \fi%
```

Now we know the width w of the song number box, so we typeset everything else in a box (`\SB@box`) of width $c - w$, where c is the column width. (If `\pagepreludes` is on, we instead use width $c - 2w$ so that the material stays centered on the page.)

```
950    \setbox\SB@box\vbox{%
951      \ifdim\songnumwidth>\z@%
952        \SB@dimen\wd\SB@boxii%
953        \advance\SB@dimen3\p@%
954        \ifpagepreludes\multiply\SB@dimen\tw@\fi%
955        \advance\hsize-\SB@dimen%
956      \fi%
957      \ifpagepreludes\centering\else\SB@raggedright\fi%
958      \offinterlineskip\lineskip\p@%
959      {\stitlefont\relax%
960       \songtitle\par%
961       \nexttitle%
962       \foreachtitle{(\songtitle)\par}}%
963      \ifdim\prevdepth=\z@\kern\p@\fi%
964      \parskip\p@\relax\tiny%
965      \extendprelude%
966      \kern\z@%
967    }%
```

If the song number is being printed (i.e., `\songnumwidth` is positive), and its height is greater than the height of the other material, then we just put `\SB@boxii` and `\SB@box` side-by-side. If the song number is being printed but its height is less, then we re-typeset it at height equal to the other material, and place the boxes side-by-side. Finally, if the song number is not being printed at all, we just unbox `\SB@box` onto the vertical list.

```
968      \ifdim\songnumwidth>\z@%
969        \hbox{%
970          \ifdim\ht\SB@boxii>\ht\SB@box%
971            \box\SB@boxii%
972            \kern3\p@%
```

```
973            \vtop{\box\SB@box}%
974          \else%
975            \SB@colorbox\snumbgcolor{\vbox to\ht\SB@box{{%
976              \hbox to\songnumwidth{%
977                \printsongnum{\thesongnum}\hfil%
978              }\vfil%
979            }}}%
980            \kern3\p@%
981            \box\SB@box%
982          \fi%
983        }%
984      \else%
985        \unvbox\SB@box%
986      \fi%
987    \fi%
988 }
```

\makepostlude   Generate the material that ends each song. The default implementation just
                 prints the copyright and licensing information (if any) as a single, left-justified,
                 non-indentended paragraph in fine print.

```
989 \newcommand\makepostlude{%
990    \SB@raggedright\baselineskip\z@skip\parskip\z@skip\parindent\z@%
991    \tiny\extendpostlude%
992 }
```

\showauthors   Display the author information in the prelude. This macro is only called by
                 \extendprelude, which is only called by \makeprelude; so if you redefine either
                 of those, you don't need this. The default implementation prints the authors in
                 boldface and shortens the spacing after periods so that they don't look like ends of
                 sentences.

```
993 \newcommand\showauthors{%
994    \setbox\SB@box\hbox{\bfseries\sfcode`.\@m\songauthors}%
995    \ifdim\wd\SB@box>\z@\unhbox\SB@box\par\fi%
996 }
```

\showrefs   Display the scripture references in the prelude. This macro is only called by
                 \extendprelude, which is only called by \makeprelude; so if you redefine either
                 of those, you don't need this. The default implementation prints the scripture
                 references in slanted (oblique) font.

```
997 \newcommand\showrefs{%
998    \setbox\SB@box\hbox{\slshape\songrefs\vphantom,}%
999    \ifdim\wd\SB@box>\z@\unhbox\SB@box\par\fi%
1000 }
```

\SB@next   Several macros use \futurelet to look ahead in the input stream, and then take
\SB@donext   various actions depending on what is seen. In these macros, \SB@next is assigned
\SB@dothis   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.

```
1001 \newcommand\SB@next{}
1002 \newcommand\SB@donext{}
1003 \newcommand\SB@dothis{}
```

`\SB@nextname` Sometimes when scanning ahead we `\string`ify the name of the next token. When that happens, the name is stored in this macro for safekeeping.

```
1004 \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.

```
1005 \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.

```
1006 \newcommand\SB@parsetitles[1]{%
1007   \begingroup%
1008     \global\SB@titlelist{\\}%
1009     \SB@toks{}%
1010     \let\\\SB@titlesep%
1011     \SB@pthead#1\SB@endparse%
1012   \endgroup%
1013 }
```

`\SB@pthead` While processing tokens at the head of a title, we skip over all spaces until we
`\SB@@pthead` reach a non-space token.
`\SB@@@pthead`
```
1014 \newcommand\SB@pthead{\futurelet\SB@next\SB@@pthead}
1015 \newcommand\SB@@pthead{%
1016   \ifcat\noexpand\SB@next\@sptoken%
1017     \expandafter\SB@@@pthead%
1018   \else%
1019     \expandafter\SB@ptmain%
1020   \fi%
1021 }
1022 \newcommand\SB@@@pthead{%
1023   \afterassignment\SB@pthead%
1024   \let\SB@next= }
```

`\SB@ptloop` The iterator of the title parser loop just scans the next token.

```
1025 \newcommand\SB@ptloop{\futurelet\SB@next\SB@ptmain}
```

`\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.

```
1026 \newcommand\SB@ptmain{%
1027   \ifcat\noexpand\SB@next\@sptoken%
1028     \let\SB@donext\SB@ptsp%
1029   \else\ifcat\noexpand\SB@next\bgroup%
1030     \let\SB@donext\SB@ptbg%
1031   \else\ifx\SB@next\SB@endparse%
1032     \global\SB@titlelist\expandafter{\the\SB@titlelist\\}%
```

77

```
1033      \let\SB@donext\@gobble%
1034    \else\ifx\SB@next\\%
1035      \SB@toks{}%
1036      \def\SB@donext{\SB@ptstep\SB@pthead}%
1037    \else%
1038      \def\SB@donext{\SB@ptstep\SB@ptloop}%
1039    \fi\fi\fi\fi%
1040    \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.

```
1041 \newcommand\SB@ptstep[2]{%
1042    \global\SB@titlelist\expandafter\expandafter\expandafter{%
1043      \expandafter\the\expandafter\SB@titlelist\the\SB@toks#2}%
1044    \SB@toks{}%
1045    #1}
```

\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.

```
1046 \newcommand\SB@ptbg[1]{\SB@ptstep\SB@ptloop{{#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.

```
1047 \newcommand\SB@ptsp{
1048    \SB@appendsp\SB@toks%
1049    \afterassignment\SB@ptloop%
1050    \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.

```
1051 \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.

```
1052 \newcommand\SB@endparse{%
1053    \SB@Error{Title parsing failed}{This error should not occur.}%
1054 }
```

\SB@parserefs Assign the \songrefs macro 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`, respec-
tively. (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 are inserted because some of these tokens might end up being arguments to multi-byte unicode character macros 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.

```
1055 \newcommand\SB@parsesrefs[1]{%
1056   \begingroup%
1057     \SB@toks{\begingroup\SB@sractives}%
1058     \SB@prloop#1\SB@endparse%
1059     \xdef\songrefs{\the\SB@toks\endgroup}%
1060   \endgroup%
1061 }
```

\SB@prloop The main loop of the scripture reference scanner identifies each space, hyphen, and
\SB@prstep comma for special treatment.
\SB@@prstep
```
1062 \newcommand\SB@prloop{\futurelet\SB@next\SB@prstep}
1063 \newcommand\SB@prstep{%
1064   \ifcat\noexpand\SB@next A%
1065     \expandafter\SB@prcpy%
1066   \else%
1067     \expandafter\SB@@prstep%
1068   \fi%
1069 }
1070 \newcommand\SB@@prstep{%
1071   \ifcat\noexpand\SB@next\@sptoken%
1072     \let\SB@donext\SB@prspace%
1073   \else\ifx\SB@next-%
1074     \let\SB@donext\SB@prhyphen%
1075   \else\ifx\SB@next,%
1076     \let\SB@donext\SB@prcomma%
1077   \else\ifx\SB@next\SB@endparse%
1078     \let\SB@donext\@gobble%
1079   \else\ifcat\noexpand\SB@next\bgroup%
1080     \let\SB@donext\SB@prgr%
1081   \else%
1082     \let\SB@donext\SB@prcpy%
1083   \fi\fi\fi\fi\fi%
1084   \SB@donext%
1085 }
```

\SB@prcpy Anything that isn't one of the special tokens above, and anything in a group, is
\SB@prgr   copied without modification.
```
1086 \newcommand\SB@prcpy[1]{\SB@toks\expandafter{\the\SB@toks#1}\SB@prloop}
1087 \newcommand\SB@prgr[1]{\SB@toks\expandafter{\the\SB@toks{#1}}\SB@prloop}
```

`\SB@prcomma` Commas and hyphens are replaced with active equivalents.
`\SB@prhyphen`
```
1088 \newcommand\SB@prcomma[1]{}
1089 {\catcode'\,\active
1090  \gdef\SB@prcomma#1{\SB@toks\expandafter{\the\SB@toks,}\SB@prloop}}
1091 \newcommand\SB@prhyphen[1]{}
1092 {\catcode'-\active
1093  \gdef\SB@prhyphen#1{\SB@toks\expandafter{\the\SB@toks-}\SB@prloop}}
```

`\SB@prspace` Spaces are made active as well, but doing so requires some specialized code since
`\SB@@prspace` they cannot be consumed as implicit macro arguments.
```
1094 \newcommand\SB@prspace[1]{}
1095 {\obeyspaces
1096 \gdef\SB@prspace{\SB@toks\expandafter{\the\SB@toks␣}\SB@@prspace}}
1097 \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.
```
1098 \newcommand\SB@sractives{}
1099 {\catcode'\,\active\catcode'-\active\obeyspaces%
1100 \gdef\SB@sractives{%
1101 \let,\SB@srcomma\let-\SB@srhyphen\let␣\SB@srspace%
1102 \SB@srspacing}%
1103 }
```

`\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.
```
1104 \newcommand\SB@srspacing{%
1105  \nonfrenchspacing\sfcode'\;=1500\sfcode'\,=1250\relax%
1106 }
```

`\SB@srcomma` Commas not already followed by whitespace are appended with a thin, non-breaking
`\SB@@srcomma` space.
```
1107 \newcommand\SB@srcomma{,\futurelet\SB@next\SB@@srcomma}
1108 \newcommand\SB@@srcomma{%
1109   \ifx\SB@next\SB@srspace\else%
1110     \nobreak\thinspace%
1111   \fi%
1112 }
```

`\SB@srhyphen` Hyphens that are not already part of a ligature (an en- or em-dash) become
`\SB@@srhyphen` en-dashes.
`\SB@srdash`
`\SB@@srdash`
```
1113 \newcommand\SB@srhyphen{\futurelet\SB@next\SB@@srhyphen}
1114 \newcommand\SB@@srhyphen{%
1115   \ifx\SB@next\SB@srhyphen\expandafter\SB@srdash\else--\fi%
1116 }
1117 \newcommand\SB@srdash[1]{\futurelet\SB@next\SB@@srdash}
1118 \newcommand\SB@@srdash{%
1119   \ifx\SB@next\SB@srhyphen---\expandafter\@gobble\else--\fi%
1120 }
```

`\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).

```
1121 \newcommand\SB@srspace{\futurelet\SB@next\SB@@srspace}
1122 \newcommand\SB@@srspace{%
1123   \let\SB@donext\relax%
1124   \ifx\SB@next\SB@srspace\else%
1125     \ifnum\spacefactor>\@m%
1126       \ifnum\spacefactor>1499 %
1127         \ifcat\noexpand\SB@next0%
1128           \let\SB@donext\SB@srcso%
1129         \else%
1130           \penalty-5\enskip%
1131         \fi%
1132       \else%
1133         \space%
1134       \fi%
1135     \else%
1136       \nobreak\space%
1137     \fi%
1138   \fi%
1139   \SB@donext%
1140 }
```

`\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.

```
1141 \newcommand\SB@srcso[1]{\futurelet\SB@temp\SB@@srcso}
1142 \newcommand\SB@@srcso{%
1143   \ifx\SB@temp\SB@srspace%
1144     \penalty-5\enskip%
1145   \else%
1146     \space%
1147   \fi%
1148   \SB@next%
1149 }
```

15.6   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.

```
1150 \newif\ifSB@stanza
```

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

```
1151 \newcommand\SB@stanzabreak{%
1152   \ifhmode\par\fi%
1153   \ifSB@stanza%
1154     \SB@breakpoint{%
1155       \ifSB@inverse%
1156         \ifSB@prevverse\vvpenalty\else\cvpenalty\fi%
1157       \else%
1158         \ifSB@prevverse\vcpenalty\else\ccpenalty\fi%
1159       \fi%
1160     }%
1161     \vskip\versesep%
1162   \fi%
1163 }
```

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

```
1164 \newcommand\SB@breakpoint[1]{%
1165   \begingroup%
1166     \ifnum#1<\@M%
1167       \SB@skip\colbotglue\relax%
1168       \SB@skip-\SB@skip%
1169     \else%
1170       \SB@skip\z@skip%
1171     \fi%
1172     \advance\SB@skip\lastskip%
1173     \unskip%
1174     \nobreak%
1175     \ifnum#1<\@M%
1176       \vskip\colbotglue\relax%
1177       \penalty#1%
1178     \fi%
1179     \vskip\SB@skip%
1180   \endgroup%
1181 }
```

\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 descenders.

```
1182 \newcommand\SB@putbox[2]{%
1183   \begingroup%
1184     \SB@dimen\dp#2%
1185     #1#2%
1186     \setbox\SB@box\hbox{{\lyricfont\relax p}}%
1187     \ifdim\SB@dimen<\dp\SB@box%
1188       \advance\SB@dimen-\dp\SB@box%
1189       \vskip-\SB@dimen%
1190     \fi%
1191     \setbox\SB@box\box\voidb@x%
1192   \endgroup%
1193 }
```

\SB@obeylines   Within verses and choruses we would like to use \obeylines so that each ⟨return⟩ 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 ⟨*return*⟩ as well.

```
1194 \newcommand\SB@obeylines{%
1195   \let\par\SB@par%
1196   \obeylines%
1197   \let\@par\SB@@par%
1198 }
```

`\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.)

```
1199 \newcommand\SB@par{%
1200   \ifhmode%
1201     \SB@cnt\interlinepenalty%
1202     \interlinepenalty\@M%
1203     \endgraf%
1204     \interlinepenalty\SB@cnt%
1205     \ifSB@inchorus%
1206       \ifdim\cbarwidth>\z@\nobreak\else\SB@ilpenalty\fi%
1207     \else%
1208       \SB@ilpenalty%
1209     \fi%
1210   \fi%
1211 }
```

`\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.

```
1212 \newcommand\SB@ilpenalty{%
1213   \ifnum\interlinepenalty=\z@\else%
1214     \penalty\interlinepenalty%
1215   \fi%
1216 }
```

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

```
1217 \newcommand\SB@@par{\let\par\SB@par\par}
```

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

```
1218 \SB@newdimen\SB@parindent
```

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

```
1219 \newcommand\SB@everypar{}
```

83

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

```
1220 \newcommand\SB@raggedright{%
1221   \SB@parindent\parindent%
1222   \raggedright%
1223   \parindent\SB@parindent%
1224 }
```

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

```
1225 \newif\ifvnumbered
```

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

```
1226 \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.

```
1227 \CheckCommand\verse{%
1228   \let\\\@centercr%
1229   \list{}{%
1230     \itemsep\z@%
1231     \itemindent-1.5em%
1232     \listparindent\itemindent%
1233     \rightmargin\leftmargin%
1234     \advance\leftmargin1.5em%
1235   }%
1236   \item\relax%
1237 }
```

verse Begin a new verse. This can be done by beginning a verse environment or by using
verse* the \beginverse macro. The latter must check for a trailing star to determine
\beginverse whether 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.

```
1238 \renewenvironment{verse}
1239   {\vnumberedtrue\SB@beginverse}
1240   {\SB@endverse}
1241 \newenvironment{verse*}
1242   {\vnumberedfalse\SB@beginverse}
1243   {\SB@endverse}
1244 \newcommand\beginverse{%
1245   \begingroup%
1246     \SB@loadactives%
1247     \@ifstar{\endgroup\vnumberedfalse\SB@beginverse}%
1248             {\endgroup\vnumberedtrue\SB@beginverse}%
1249 }
```

\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.

```
1250 \newcommand\SB@beginverse{%
1251   \ifSB@insong%
1252     \ifSB@inverse\SB@errbvv\endverse\fi%
1253     \ifSB@inchorus\SB@errbvc\endchorus\fi%
1254   \else%
1255     \SB@errbvt\beginsong{Unknown Song}%
1256   \fi%
1257   \ifrepchorus\ifvoid\SB@chorusbox\else%
1258     \SB@gotchorustrue%
1259     \ifSB@prevverse\ifvnumbered%
1260       \marks\SB@cmarkclass{\SB@cmark}%
1261     \fi\fi%
1262   \fi\fi%
1263   \SB@inversetrue%
1264   \def\SB@closeall{\endverse\endsong}%
1265   \SB@stanzabreak%
1266   \versemark\nobreak%
1267   \global\SB@stanzatrue%
1268   \SB@ifempty\SB@cr@memorize{\replay[]}%
1269   \setbox\SB@box\vbox\bgroup\begingroup%
1270     \ifvnumbered%
1271       \protected@edef\@currentlabel{\p@versenum\theversenum}%
1272       \def\SB@everypar{%
1273         \setbox\SB@box\hbox{{\printversenum{\theversenum}}}%
1274         \ifdim\wd\SB@box<\versenumwidth%
1275           \setbox\SB@box%
1276           \hbox to\versenumwidth{\unhbox\SB@box\hfil}%
1277         \fi%
1278         \ifchorded\vrule\@height\baselineskip\@width\z@\@depth\z@\fi%
1279         \placeversenum\SB@box%
1280         \gdef\SB@everypar{}%
1281       }%
1282     \else%
1283       \def\SB@everypar{%
1284         \ifchorded\vrule\@height\baselineskip\@width\z@\@depth\z@\fi%
1285         \gdef\SB@everypar{}%
1286       }%
1287     \fi%
1288     \everypar{\SB@everypar\everypar{}}%
1289     \versefont\relax\SB@setbaselineskip\versejustify%
1290     \SB@loadactives%
1291     \SB@obeylines%
1292     \penalty12345 %
1293     \everyverse\relax%
1294 }
```

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

```
1295 \newcommand\SB@endverse{%
1296   \ifSB@insong%
```

```
1297      \ifSB@inverse%
1298          \unpenalty%
1299        \endgroup\egroup%
1300        \SB@putbox\unvbox\SB@box%
1301        \SB@inversefalse%
1302        \def\SB@closeall{\endsong}%
1303        \ifvnumbered\stepcounter{versenum}\fi%
1304        \SB@prevversetrue%
1305      \else\ifSB@inchorus\SB@errevc\endchorus%
1306      \else\SB@errevo\fi\fi%
1307    \else%
1308      \SB@errevt%
1309    \fi%
1310 }
```

\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.

```
1311 \newif\ifSB@chorustop
```

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

```
1312 \SB@newbox\SB@chorusbox
```

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

```
1313 \newif\ifSB@gotchorus
```

\SB@cmarkclass   The \repeatchoruses feature requires the use of two extended mark classes
\SB@nocmarkclass 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.

```
1314 \ifSB@etex
1315   \@ifundefined{newmarks}{
1316     \@ifundefined{newmark}{
1317       \mathchardef\SB@cmarkclass83
1318       \mathchardef\SB@nocmarkclass84
1319     }{
1320       \newmark\SB@cmarkclass
1321       \newmark\SB@nocmarkclass
1322     }
1323   }{
1324     \newmarks\SB@cmarkclass
1325     \newmarks\SB@nocmarkclass
1326   }
1327 \fi
```

\SB@cmark   To determine where choruses should be inserted when \repchoruses is active,
\SB@lastcmark three kinds of marks are inserted into song boxes: \SB@cmark is used to mark places
\SB@nocmark   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!

```
1328 \newcommand*\SB@cmark{SB@cmark}
1329 \newcommand*\SB@lastcmark{SB@lastcmark}
1330 \newcommand*\SB@nocmark{SB@nocmark}
```

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

```
1331 \newenvironment{chorus}{\beginchorus}{\SB@endchorus}
1332 \newcommand\beginchorus{%
1333   \ifSB@insong
1334     \ifSB@inverse\SB@errbcv\endverse\fi%
1335     \ifSB@inchorus\SB@errbcc\endchorus\fi%
1336   \else%
1337     \SB@errbct\beginsong{Unknown Song}%
1338   \fi%
1339   \SB@inchorustrue%
1340   \def\SB@closeall{\endchorus\endsong}%
1341   \SB@chorustoptrue%
1342   \vnumberedfalse%
1343   \SB@stanzabreak%
1344   \chorusmark%
1345   \ifrepchorus%
1346     \ifSB@gotchorus\else\ifSB@prevverse\else%
1347       \global\setbox\SB@chorusbox\vbox{%
1348         \unvbox\SB@chorusbox%
1349         \SB@stanzabreak%
1350         \chorusmark%
1351       }%
1352     \fi\fi%
1353   \fi%
1354   \global\SB@stanzatrue%
1355   \replay[]%
1356   \SB@@beginchorus%
1357   \everychorus\relax%
1358 }
```

`\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.

```
1359 \newcommand\SB@@beginchorus{%
1360   \ifrepchorus\marks\SB@cmarkclass{}\fi%
1361   \setbox\SB@box\vbox\bgroup\begingroup%
1362     \ifchorded%
1363       \def\SB@everypar{%
1364         \vrule\@height\baselineskip\@width\z@\@depth\z@%
1365         \gdef\SB@everypar{}%
```

```
1366        }%
1367        \everypar{\SB@everypar\everypar{}}%
1368      \fi%
1369      \chorusfont\relax\SB@setbaselineskip\chorusjustify%
1370      \SB@loadactives%
1371      \SB@obeylines%
1372      \penalty12345 %
1373 }
```

\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.

```
1374 \newcommand\SB@endchorus{%
1375   \ifSB@insong%
1376     \ifSB@inchorus%
1377         \unpenalty%
1378       \endgroup\egroup%
1379       \SB@inchorusfalse%
1380       \def\SB@closeall{\endsong}%
1381       \setbox\SB@box\vbox{%
1382         \SB@chorusbar\SB@box%
1383         \SB@putbox\unvbox\SB@box%
1384       }
1385       \ifrepchorus\ifSB@gotchorus\else%
1386         \global\setbox\SB@chorusbox\vbox{%
1387           \unvbox\SB@chorusbox%
1388           \unvcopy\SB@box%
1389         }%
1390       \fi\fi%
1391       \unvbox\SB@box%
1392       \SB@prevversefalse%
1393     \else\ifSB@inverse\SB@errecv\endverse%
1394     \else\SB@erreco\fi\fi%
1395   \else%
1396     \SB@errect%
1397   \fi%
1398 }
```

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

```
1399 \newcommand\SB@cbarshift{%
1400   \ifSB@inchorus\ifdim\cbarwidth>\z@%
1401     \advance\leftskip\cbarwidth%
1402     \advance\leftskip5\p@\relax%
1403   \fi\fi%
1404 }
```

\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.

```
1405 \newcommand\SB@chorusbar[1]{%
1406   \ifdim\cbarwidth>\z@%
1407     \SB@dimen\ht#1%
1408     \SB@dimenii\dp#1%
```

```
1409       \advance\SB@dimen%
1410         \ifSB@chorustop\ifchorded\else2\fi\fi\SB@dimenii%
1411       \SB@skip\SB@dimen\relax%
1412       \SB@computess\SB@skip1\@plus#1%
1413       \SB@computess\SB@skip{-1}\@minus#1%
1414       \nointerlineskip\null\nobreak%
1415       \leaders\vrule\@width\cbarwidth\vskip\SB@skip%
1416       \ifSB@chorustop\ifchorded\else%
1417         \advance\SB@skip-\SB@dimenii%
1418       \fi\fi%
1419       \nobreak\vskip-\SB@skip%
1420     \fi%
1421 }
```

\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.

```
1422 \newcommand\SB@computess[4]{%
1423   \begingroup%
1424     \vbadness\@M\vfuzz\maxdimen%
1425     \SB@dimen4096\p@%
1426     \setbox\SB@box\vbox spread#2\SB@dimen{\unvcopy#4}%
1427     \ifnum\badness=\z@%
1428       \global\advance#1\z@#31fil\relax%
1429     \else%
1430       \SB@dimenii\SB@dimen%
1431       \loop%
1432         \SB@dimenii.5\SB@dimenii%
1433         \ifnum\badness<100 %
1434           \advance\SB@dimen\SB@dimenii%
1435         \else
1436           \advance\SB@dimen-\SB@dimenii%
1437         \fi%
1438         \setbox\SB@box\vbox spread#2\SB@dimen{\unvcopy#4}%
1439         \ifnum\badness=100 \SB@dimenii\z@\fi%
1440       \ifdim\SB@dimenii>.1\p@\repeat%
1441       \ifdim\SB@dimen<.1\p@\SB@dimen\z@\fi%
1442       \global\advance#1\z@#3\SB@dimen\relax%
1443     \fi%
1444   \endgroup%
1445 }
```

\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 gen-
erating a non-partial list of songs. When generating a partial list, \brk between

89

songs is ignored.

```
1446 \newcommand\brk{%
1447   \ifSB@insong%
1448     \ifhmode\penalty-5 \else%
1449       \unpenalty%
1450       \ifSB@inchorus%
1451         \ifdim\cbarwidth=\z@%
1452           \ifrepchorus\marks\SB@cmarkclass{}\fi%
1453           \SB@breakpoint\brkpenalty%
1454         \else%
1455           \endgroup\egroup%
1456           \ifrepchorus\ifSB@gotchorus\else%
1457             \global\setbox\SB@chorusbox\vbox{%
1458               \unvbox\SB@chorusbox%
1459               \SB@chorusbar\SB@box%
1460               \unvcopy\SB@box%
1461               \SB@breakpoint\brkpenalty%
1462             }%
1463           \fi\fi%
1464           \SB@chorusbar\SB@box%
1465           \unvbox\SB@box%
1466           \SB@breakpoint\brkpenalty%
1467           \SB@chorustopfalse%
1468           \SB@@beginchorus%
1469         \fi%
1470       \else%
1471         \SB@breakpoint\brkpenalty%
1472       \fi%
1473     \fi%
1474   \else%
1475     \ifpartiallist\else\SB@nextcol\@ne\colbotglue\fi%
1476   \fi%
1477 }
```

\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.

```
1478 \newcommand\SB@boxup[1]{%
1479   \setbox\SB@box\hbox{{\notefont\relax#1}}%
1480   \SB@dimen\wd\SB@box%
1481   \advance\SB@dimen6\p@%
1482   \advance\SB@dimen\leftskip%
1483   \advance\SB@dimen\rightskip%
1484   \ifdim\SB@dimen>\hsize%
1485     \vbox{{%
1486       \advance\hsize-6\p@%
1487       \advance\hsize-\leftskip%
1488       \advance\hsize-\rightskip%
1489       \notejustify%
1490       \unhbox\SB@box\par%
1491       \kern\z@%
1492     }}%
1493   \else%
1494     \vbox{\box\SB@box\kern\z@}%
```

```
1495    \fi%
1496 }
```

\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.

```
1497 \newcommand\textnote[1]{%
1498    \ifhmode\par\fi%
1499    \ifnum\lastpenalty=12345\else%
1500      \ifSB@inverse%
1501        \vskip2\p@\relax%
1502      \else\ifSB@inchorus%
1503        \vskip2\p@\relax%
1504      \else\ifSB@stanza%
1505        \nobreak\vskip\versesep%
1506      \fi\fi\fi%
1507    \fi%
1508    \begingroup%
1509      \everypar{}%
1510      \ifchorded\chordedfalse\SB@setbaselineskip\chordedtrue\fi%
1511      \placenote{\SB@colorbox\notebgcolor{\SB@boxup{#1}}}%
1512    \endgroup%
1513    \nobreak%
1514    \ifSB@inverse%
1515      \vskip2\p@\relax%
1516    \else\ifSB@inchorus%
1517      \vskip2\p@\relax%
1518    \else\ifSB@stanza\else%
1519      \nobreak\vskip\versesep%
1520    \fi\fi\fi%
1521 }
```

\musicnote Create a textual note for musicians.

```
1522 \newcommand\musicnote[1]{\ifchorded\textnote{#1}\fi}
```

\echo Typeset an echo part in the lyrics. Echo parts are in a user-customizable font and
\SB@echo parenthesized.
\SB@@echo   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 re-scan 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.)

```
1523 \ifSB@etex
1524   \newcommand\echo{\begingroup\@sanitize\SB@echo}
1525   \newcommand\SB@echo[1]{%
1526     \endgroup%
1527     \begingroup%
1528       \echofont\relax%
1529       \endlinechar\m@ne%
1530       \scantokens{(#1)}%
1531     \endgroup%
1532   }
1533 \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.

```
1534   \newcommand\echo{%
1535     \begingroup%
1536       \echofont\relax%
1537       \afterassignment\SB@echo%
1538       \setbox\SB@box\hbox%
1539   }
1540   \newcommand\SB@echo{\aftergroup\SB@@echo(}
1541   \newcommand\SB@@echo{\unhbox\SB@box)\endgroup}
1542 \fi
```

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

```
1543 \newcommand\rep[1]{%
1544   (\raise.25ex\hbox{%
1545     \fontencoding{OMS}\fontfamily{cmsy}\selectfont\char\tw@%
1546   }#1)%
1547 }
```

15.7 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.

```
1548 \newenvironment{songgroup}{%
1549   \ifnum\SB@grouplvl=\z@%
1550     \edef\SB@sgroup{\thesongnum}%
1551     \global\SB@groupcnt\m@ne%
1552   \fi%
```

```
1553     \advance\SB@grouplvl\@ne%
1554 }{%
1555     \advance\SB@grouplvl\m@ne%
1556     \ifnum\SB@grouplvl=\z@\let\SB@sgroup\@empty\fi%
1557 }
```

\SB@grouplvl Count the songgroup environment nesting depth.

```
1558 \SB@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.

```
1559 \newenvironment{intersong}{%
1560     \ifSB@insong\SB@errbro\SB@closeall\fi%
1561     \ifSB@intersong\SB@errbrr\SB@closeall\fi%
1562     \setbox\SB@chorusbox\box\voidb@x%
1563     \SB@intersongtrue%
1564     \def\SB@closeall{\end{intersong}}%
1565     \setbox\SB@songbox\vbox\bgroup\begingroup%
1566        \ifnum\SB@numcols>\z@\hsize\SB@colwidth\fi%
1567        \ifdim\sbarheight>\z@%
1568          \hrule\@height\sbarheight\@width\hsize%
1569          \nobreak%
1570        \fi%
1571 }{%
1572     \ifSB@intersong
1573        \ifdim\sbarheight>\z@%
1574          \ifhmode\par\fi%
1575          \SB@skip\lastskip%
1576          \unskip\nobreak\vskip\SB@skip%
1577          \hbox{\vrule\@height\sbarheight\@width\hsize}%
1578        \fi%
1579        \endgroup\egroup%
1580        \ifSB@omitscrip%
1581          \setbox\SB@songbox\box\voidb@x%
1582        \else%
1583          \SB@submitsong%
1584        \fi%
1585        \SB@intersongfalse%
1586     \else%
1587        \ifSB@insong\SB@errero\SB@closeall\else\SB@errert\fi%
1588     \fi%
1589 }
```

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

```
1590 \newenvironment{intersong*}{%
1591     \ifSB@insong\SB@errbro\SB@closeall\fi%
1592     \ifSB@intersong\SB@errbrr\SB@closeall\fi%
1593     \setbox\SB@chorusbox\box\voidb@x%
1594     \SB@intersongtrue%
1595     \def\SB@closeall{\end{intersong*}}%
1596     \setbox\SB@songbox\vbox\bgroup\begingroup%
1597 }{%
```

```
1598    \ifSB@intersong%
1599      \endgroup\egroup%
1600      \ifSB@omitscrip%
1601        \setbox\SB@songbox\box\voidb@x%
1602      \else%
1603        \def\SB@stype{\SB@styppage}%
1604        \SB@submitsong%
1605        \def\SB@stype{\SB@stypcol}%
1606      \fi%
1607      \SB@intersongfalse%
1608    \else%
1609      \ifSB@insong\SB@errero\SB@closeall\else\SB@errert\fi%
1610    \fi%
1611 }
```

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.

```
1612 \newenvironment{scripture}{\beginscripture}{\SB@endscripture}
1613 \newcommand\beginscripture[1]{%
1614    \begin{intersong}%
1615      \SB@parsesrefs{#1}%
1616      \setbox\SB@envbox\hbox{{\printscrcite\songrefs}}%
1617      \def\SB@closeall{\endscripture}%
1618      \nobreak\vskip5\p@%
1619      \SB@parindent\parindent\parindent\z@%
1620      \parskip\z@skip\parfillskip\@flushglue%
1621      \leftskip\SB@parindent\rightskip\SB@parindent\relax%
1622      \scripturefont\relax%
1623      \baselineskip\f@size\p@\@plus\p@\relax%
1624      \advance\baselineskip\p@\relax%
1625      \emergencystretch.3em%
1626 }
```

\SB@endscripture   End a scripture quotation.

```
1627 \newcommand\SB@endscripture{%
1628    \ifSB@intersong
1629      \scitehere%
1630      \ifhmode\par\fi%
1631      \vskip-3\p@%
1632    \end{intersong}%
1633    \fi%
1634 }
```

\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.

```
1635 \newcommand\scitehere{%
1636   \ifSB@intersong%
1637     \ifvoid\SB@envbox\else%
1638       \ifvmode%
1639         \setbox\SB@box\lastbox%
1640         \nointerlineskip\noindent\hskip-\leftskip%
1641         \unhbox\SB@box\unskip%
1642       \fi%
1643       \unskip\nobreak\hfil\penalty50\hskip.8em\null\nobreak\hfil%
1644       \box\SB@envbox\kern-\SB@parindent%
1645       {\parfillskip\z@\finalhyphendemerits2000\par}%
1646     \fi%
1647   \else%
1648     \SB@errscrip\scitehere%
1649   \fi%
1650 }
```

`\Acolon` Typeset a line of poetry in a scripture quotation.
`\Bcolon`
```
1651 \newcommand\Acolon{\SB@colon2\Acolon}
1652 \newcommand\Bcolon{\SB@colon1\Bcolon}
```

`\SB@colon` Begin a group of temporary definitions that will end at the next ⟨*return*⟩. The ⟨*return*⟩ will end the paragraph and close the local scope.

```
1653 \newcommand\SB@colon[2]{%
1654   \ifSB@intersong\else%
1655     \SB@errscrip#2%
1656     \beginscripture{Unknown}%
1657   \fi%
1658   \ifhmode\par\fi%
1659   \begingroup%
1660     \rightskip\SB@parindent\@plus4em%
1661     \advance\leftskip2\SB@parindent%
1662     \advance\parindent-#1\SB@parindent%
1663     \def\par{\endgraf\endgroup}%
1664     \obeylines%
1665 }
```

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

```
1666 \newcommand\strophe{%
1667   \ifSB@intersong\else%
1668     \SB@errscrip\strophe\beginscripture{Unknown}%
1669   \fi%
1670   \vskip.9ex\@plus.45ex\@minus.68ex\relax%
1671 }
```

`\scripindent`	Create an indented sub-block within a scripture quotation.
`\scripoutdent`	1672 `\newcommand\SB@scripdent[2]{%`
`\SB@scripdent`	1673 `  \ifSB@intersong\else%`
	1674 `    \SB@errscrip#2\beginscripture{Unknown}%`
	1675 `  \fi%`
	1676 `  \ifhmode\par\fi%`
	1677 `  \advance\leftskip#1\SB@parindent\relax%`
	1678 `}`
	1679 `\newcommand\scripindent{\SB@scripdent1\scripindent}`
	1680 `\newcommand\scripoutdent{\SB@scripdent-\scripoutdent}`

`\shiftdblquotes`	The Zaph Chancery font used by default to typeset scripture quotations seems to
`\SB@ldqleft`	have some kerning problems with double-quote ligatures. The `\shiftdblquotes`
`\SB@ldqright`	macro allows one to modify the spacing around all double-quotes until the current
`\SB@rdqleft`	group ends.
`\SB@rdqright`	1681 `\newcommand\SB@quotesactive{%`
`\SB@scanlq`	1682 `  \catcode`'\active%`
`\SB@scanrq`	1683 ` \catcode``\active%`
`\SB@dolq`	1684 `}`
`\SB@dorq`	1685 `\newcommand\shiftdblquotes[4]{}`
	1686 `\newcommand\SB@ldqleft{}`
	1687 `\newcommand\SB@ldqright{}`
	1688 `\newcommand\SB@rdqleft{}`
	1689 `\newcommand\SB@rdqright{}`
	1690 `\newcommand\SB@scanlq{}`
	1691 `\newcommand\SB@scanrq{}`
	1692 `\newcommand\SB@dolq{}`
	1693 `\newcommand\SB@dorq{}`
	1694 `{`
	1695 `  \SB@quotesactive`
	1696 `  \gdef\shiftdblquotes#1#2#3#4{%`
	1697 `    \def\SB@ldqleft{\kern#1}%`
	1698 `    \def\SB@ldqright{\kern#2}%`
	1699 `    \def\SB@rdqleft{\kern#3}%`
	1700 `    \def\SB@rdqright{\kern#4}%`
	1701 `    \SB@quotesactive%`
	1702 `    \def`{\futurelet\SB@next\SB@scanlq}%`
	1703 `    \def'{\futurelet\SB@next\SB@scanrq}%`
	1704 `  }`
	1705 `  \gdef\SB@scanlq{%`
	1706 `    \ifx\SB@next`%`
	1707 `      \expandafter\SB@dolq%`
	1708 `    \else%`
	1709 `      \expandafter\lq%`
	1710 `    \fi%`
	1711 `  }`
	1712 `  \gdef\SB@scanrq{%`
	1713 `    \ifx\SB@next'%`
	1714 `      \expandafter\SB@dorq%`
	1715 `    \else%`
	1716 `      \expandafter\rq%`
	1717 `    \fi%`
	1718 `  }`
	1719 `  \gdef\SB@dolq`{%`

```
1720     \ifvmode\leavevmode\else\/\fi%
1721     \vadjust{}%
1722     \SB@ldqleft\lq\lq\SB@ldqright%
1723     \vadjust{}%
1724 }
1725 \gdef\SB@dorq'{%
1726     \ifvmode\leavevmode\else\/\fi%
1727     \vadjust{}%
1728     \SB@rdqleft\rq\rq\SB@rdqright%
1729     \vadjust{}%
1730 }
1731 }
```

15.8   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).

```
1732 \SB@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.

```
1733 \newif\ifSB@convertnotes
```

\notenameA   Reserve a control sequence for each note of the diatonic scale. These will be used
\notenameB   to identify which token sequences the input file uses to denote the seven scale
\notenameC   degrees. Their eventual definitions *must* consist entirely of uppercase letters, and
\notenameD   they must be assigned using \def, but that comes later.
\notenameE
```
1734 \newcommand\notenameA{}
```
\notenameF
```
1735 \newcommand\notenameB{}
```
\notenameG
```
1736 \newcommand\notenameC{}
1737 \newcommand\notenameD{}
1738 \newcommand\notenameE{}
1739 \newcommand\notenameF{}
1740 \newcommand\notenameG{}
```

\printnoteA   These control sequences are what the transposition logic actually outputs to denote
\printnoteB   each scale degree. They can include any LaTeX code that is legal in horizontal
\printnoteC   mode.
\printnoteD
```
1741 \newcommand\printnoteA{}
```
\printnoteE
```
1742 \newcommand\printnoteB{}
```
\printnoteF
```
1743 \newcommand\printnoteC{}
```
\printnoteG
```
1744 \newcommand\printnoteD{}
1745 \newcommand\printnoteE{}
1746 \newcommand\printnoteF{}
1747 \newcommand\printnoteG{}
```

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

```
1748 \newcommand\notenamesin[7]{%
1749   \def\notenameA{#1}%
1750   \def\notenameB{#2}%
1751   \def\notenameC{#3}%
1752   \def\notenameD{#4}%
1753   \def\notenameE{#5}%
1754   \def\notenameF{#6}%
1755   \def\notenameG{#7}%
1756   \SB@convertnotestrue%
1757 }
```

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

```
1758 \newcommand\notenamesout[7]{%
1759   \def\printnoteA{#1}%
1760   \def\printnoteB{#2}%
1761   \def\printnoteC{#3}%
1762   \def\printnoteD{#4}%
1763   \def\printnoteE{#5}%
1764   \def\printnoteF{#6}%
1765   \def\printnoteG{#7}%
1766   \SB@convertnotestrue%
1767 }
```

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

```
1768 \newcommand\notenames[7]{%
1769   \notenamesin{#1}{#2}{#3}{#4}{#5}{#6}{#7}%
1770   \notenamesout{#1}{#2}{#3}{#4}{#5}{#6}{#7}%
1771   \SB@convertnotesfalse%
1772 }
```

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

```
1773 \newcommand\alphascale{\notenames ABCDEFG}
1774 \newcommand\solfedge{\notenames{LA}{SI}{DO}{RE}{MI}{FA}{SOL}}
1775 \alphascale
```

\ifSB@prefshrps When a transposed chord falls on a black key, the code must choose which en-
harmonically equivalent name to give the new chord. (For example, should C
transposed by +1 be named C# or Db?) 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.

```
1776 \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 whether the current chord is the first one seen
in the song, and should therefore be used to guess the key of the song.

```
1777 \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.

```
1778 \newcommand\transpose[1]{%
1779   \advance\SB@transposefactor by#1\relax%
1780   \SB@cnt\SB@transposefactor%
1781   \divide\SB@cnt12 %
1782   \multiply\SB@cnt12 %
1783   \advance\SB@transposefactor-\SB@cnt%
1784   \SB@needkeytrue%
1785 }
```

\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.

```
1786 \newcommand\capo[1]{%
1787   \iftranscapos\transpose{#1}\else\musicnote{capo #1}\fi%
1788 }
```

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

```
1789 \newcommand\prefersharps{\SB@prefshrpstrue\SB@needkeyfalse}
1790 \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.

```
1791 \newcommand\transposehere[1]{%
1792   \ifnum\SB@transposefactor=\z@%
1793     \ifSB@convertnotes%
1794       \SB@dotranspose{#1}%
1795       \the\SB@toks%
1796     \else%
1797       #1%
1798     \fi%
1799   \else%
1800     \ifSB@convertnotes%
1801       {\SB@transposefactor\z@%
1802         \SB@dotranspose{#1}%
1803         \xdef\SB@tempv{\the\SB@toks}}%
1804     \else%
1805       \def\SB@tempv{#1}%
1806     \fi%
1807     \SB@dotranspose{#1}%
1808     \expandafter\trchordformat\expandafter{\SB@tempv}{\the\SB@toks}%
1809   \fi%
1810 }
```

\notrans Suppress chord transposition without suppressing note name conversion. When a \notrans{⟨text⟩} macro appears within text undergoing transposition, the \notrans macro and the group will be preserved verbatim by the transposition parser. When it is then expanded after parsing, we must therefore re-invoke the transposition logic on the argument, but in an environment where the transposition factor has been temporarily set to zero. This causes note name conversion to occur without actually transposing.

```
1811 \newcommand\notrans[1]{%
1812   \begingroup%
1813     \SB@transposefactor\z@%
1814     \transposehere{#1}%
1815   \endgroup%
1816 }
```

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

```
1817 \newcommand\SB@dotranspose[1]{%
1818   \SB@toks{}%
1819   \let\SB@dothis\SB@trmain%
1820   \SB@trscan#1\SB@trend%
1821 }
```

\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`.

```
1822 \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.

```
1823 \newcommand\SB@trscan{\futurelet\SB@next\SB@dothis}
```

\SB@trmain Test to see whether 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.

```
1824 \newcommand\SB@trmain{%
1825   \ifx\SB@next\bgroup%
1826     \let\SB@donext\SB@trgroup%
1827   \else\ifx\SB@next\egroup%
1828     \SB@toks\expandafter{\the\SB@toks\egroup}%
1829     \let\SB@donext\SB@trskip%
1830   \else\ifcat\noexpand\SB@next\@sptoken%
1831     \SB@appendsp\SB@toks%
1832     \let\SB@donext\SB@trskip%
1833   \else%
1834     \let\SB@donext\SB@trstep%
1835   \fi\fi\fi%
1836   \SB@donext%
1837 }
```

\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.

```
1838 \newcommand\SB@trgroup[1]{%
1839   \SB@toks\expandafter{\the\SB@toks{#1}}%
1840   \SB@trscan%
1841 }
```

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

```
1842 \newcommand\SB@trskip{%
1843   \afterassignment\SB@trscan%
1844   \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 whether 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.

```
1845 \newcommand\SB@trstep[1]{%
1846   \let\SB@donext\SB@trscan%
1847   \ifcat\noexpand\SB@next A%
1848     \ifnum\uccode`#1=`#1%
1849       \def\SB@temp{#1}%
1850       \let\SB@dothis\SB@trnote%
1851     \else%
1852       \SB@toks\expandafter{\the\SB@toks#1}%
1853     \fi%
1854   \else\ifx\SB@next\SB@trend
1855     \let\SB@donext\relax%
1856   \else%
1857     \SB@toks\expandafter{\the\SB@toks#1}%
1858   \fi\fi%
1859   \SB@donext%
1860 }
```

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

```
1861 \newcommand\SB@trnote{%
1862   \ifcat\noexpand\SB@next A%
1863     \let\SB@donext\SB@trnotestep%
1864   \else\ifnum\SB@transposefactor=\z@%
1865     \SB@cnt\z@%
1866     \let\SB@donext\SB@trtrans%
1867   \else\ifx\SB@next\flt%
1868     \SB@cnt\m@ne%
1869     \let\SB@donext\SB@tracc%
1870   \else\ifx\SB@next\shrp%
1871     \SB@cnt\@ne%
1872     \let\SB@donext\SB@tracc%
1873   \else%
1874     \SB@cnt\z@%
1875     \let\SB@donext\SB@trtrans%
1876   \fi\fi\fi\fi%
1877   \SB@donext%
1878 }
```

\SB@trnotestep The next token is a letter. Consume it and test to see whether 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.

```
1879 \newcommand\SB@trnotestep[1]{%
```

```
1880    \ifnum\uccode`#1=`#1%
1881      \SB@app\def\SB@temp{#1}%
1882      \expandafter\SB@trscan%
1883    \else%
1884      \SB@cnt\z@%
1885      \expandafter\SB@trtrans\expandafter#1%
1886    \fi%
1887 }
```

\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.

```
1888 \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.

```
1889 \newcommand\SB@trtrans{%
1890    \advance\SB@cnt%
1891      \ifx\SB@temp\notenameA\z@%
1892      \else\ifx\SB@temp\notenameB\tw@%
1893      \else\ifx\SB@temp\notenameC\thr@@%
1894      \else\ifx\SB@temp\notenameD5 %
1895      \else\ifx\SB@temp\notenameE7 %
1896      \else\ifx\SB@temp\notenameF8 %
1897      \else\ifx\SB@temp\notenameG10 %
1898      \else-99 \fi\fi\fi\fi\fi\fi\fi%
1899    \ifnum\SB@cnt<\m@ne%
1900      \SB@toks\expandafter\expandafter\expandafter{%
1901        \expandafter\the\expandafter\SB@toks\SB@temp}%
1902    \else%
1903      \advance\SB@cnt\SB@transposefactor%
1904      \ifnum\SB@cnt<\z@\advance\SB@cnt12 \fi%
1905      \ifnum\SB@cnt>11 \advance\SB@cnt-12 \fi%
1906      \ifSB@needkey\ifnum\SB@transposefactor=\z@\else\SB@setkeysig\fi\fi%
1907      \edef\SB@temp{%
1908        \the\SB@toks%
1909        \ifSB@prefshrps%
1910          \ifcase\SB@cnt\printnoteA\or\printnoteA\noexpand\shrp\or%
1911            \printnoteB\or\printnoteC\or\printnoteC\noexpand\shrp\or%
1912            \printnoteD\or\printnoteD\noexpand\shrp\or\printnoteE\or%
1913            \printnoteF\or\printnoteF\noexpand\shrp\or\printnoteG\or%
1914            \printnoteG\noexpand\shrp\fi%
1915        \else%
1916          \ifcase\SB@cnt\printnoteA\or\printnoteB\noexpand\flt\or%
1917            \printnoteB\or\printnoteC\or\printnoteD\noexpand\flt\or%
1918            \printnoteD\or\printnoteE\noexpand\flt\or\printnoteE\or%
1919            \printnoteF\or\printnoteG\noexpand\flt\or\printnoteG\or%
1920            \printnoteA\noexpand\flt\fi%
```

```
1921        \fi}%
1922      \SB@toks\expandafter{\SB@temp}%
1923    \fi%
1924    \let\SB@dothis\SB@trmain%
1925    \SB@trscan%
1926 }
```

\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).

```
1927 \newcommand\SB@setkeysig{%
1928    \global\SB@needkeyfalse%
1929    \ifcase\SB@cnt%
1930      \global\SB@prefshrpstrue\or% A
1931      \global\SB@prefshrpsfalse\or% Bb
1932      \global\SB@prefshrpstrue\or% B
1933      \ifx\SB@next m% C
1934        \global\SB@prefshrpsfalse%
1935      \else%
1936        \global\SB@prefshrpstrue%
1937      \fi\or%
1938      \global\SB@prefshrpstrue\or% C#
1939      \ifx\SB@next m% D
1940        \global\SB@prefshrpsfalse%
1941      \else%
1942        \global\SB@prefshrpstrue%
1943      \fi\or%
1944      \global\SB@prefshrpsfalse\or% Eb
1945      \global\SB@prefshrpstrue\or% E
1946      \global\SB@prefshrpsfalse\or% F
1947      \global\SB@prefshrpstrue\or% F#
1948      \ifx\SB@next m% G
1949        \global\SB@prefshrpsfalse%
1950      \else%
1951        \global\SB@prefshrpstrue%
1952      \fi\or%
1953      \global\SB@prefshrpsfalse\else% Ab
1954      \global\SB@needkeytrue% non-chord
1955    \fi%
1956 }
```

\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.

```
1957 \newcommand\SB@trend{%
1958    \SB@Error{Internal Error: Transposition failed}%
1959           {This error should not occur.}%
1960 }
```

103

15.9 Measure Bars

The following code handles the typesetting of measure bars.

\SB@metertop These macros remember the current numerator and denominator of the meter.
\SB@meterbot
```
1961 \newcommand\SB@metertop{}
1962 \newcommand\SB@meterbot{}
```

\meter Set the current meter without producing an actual measure bar yet.
```
1963 \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 whether this measure bar immediately follows another measure bar by using \vsplit on \lastbox.
```
1964 \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.
```
1965 \newcommand\SB@makembar[2]{%
1966   \ifSB@inverse\else%
1967     \ifSB@inchorus\else\SB@errmbar\fi%
1968   \fi%
1969   \ifhmode%
1970     \SB@skip\lastskip\unskip%
1971     \setbox\SB@box\lastbox%
1972     \copy\SB@box%
1973     \ifvbox\SB@box%
1974       \begingroup%
1975         \setbox\SB@boxii\copy\SB@box%
1976         \vbadness\@M\vfuzz\maxdimen%
1977         \setbox\SB@boxii%
1978           \vsplit\SB@boxii to\maxdimen%
1979       \endgroup%
1980       \long\edef\SB@temp{\splitfirstmark}%
1981       \ifx\SB@temp\SB@measuremark%
1982         \penalty100\hskip1em%
1983       \else%
1984         \penalty100\hskip\SB@skip%
1985       \fi%
1986     \else%
1987       \penalty100\hskip\SB@skip%
1988     \fi%
1989   \fi%
1990   \ifvmode\leavevmode\fi%
1991   \setbox\SB@box\hbox{{\meterfont\relax#1}}%
1992   \setbox\SB@boxii\hbox{{\meterfont\relax#2}}%
1993   \SB@dimen\wd\ifdim\wd\SB@box>\wd\SB@boxii\SB@box\else\SB@boxii\fi%
```

```
1994    \SB@dimenii\baselineskip%
1995    \advance\SB@dimenii-2\p@%
1996    \advance\SB@dimenii-\ht\SB@box%
1997    \advance\SB@dimenii-\dp\SB@box%
1998    \advance\SB@dimenii-\ht\SB@boxii%
1999    \advance\SB@dimenii-\dp\SB@boxii%
2000    \let\SB@temp\relax%
2001    \ifdim\SB@dimen>\z@%
2002      \advance\SB@dimenii-.75\p@%
2003      \def\SB@temp{\kern.75\p@}%
2004    \fi%
2005    \SB@maxmin\SB@dimen<{.5\p@}%
2006    \SB@maxmin\SB@dimenii<\z@%
2007    \vbox{%
2008      \mark{\SB@measuremark}%
2009      \hbox to\SB@dimen{%
2010        \hfil%
2011        \box\SB@box%
2012        \hfil%
2013      }%
2014      \nointerlineskip%
2015      \hbox to\SB@dimen{%
2016        \hfil%
2017        \box\SB@boxii%
2018        \hfil%
2019      }%
2020      \SB@temp%
2021      \nointerlineskip%
2022      \hbox to\SB@dimen{%
2023        \hfil%
2024        \vrule\@width.5\p@\@height\SB@dimenii%
2025        \hfil%
2026      }%
2027    }%
2028    \meter{}{}%
2029 }
```

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

```
2030 \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.

```
2031 \newcommand\measurebar{%
2032   \mbar\SB@metertop\SB@meterbot%
2033 }
```

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

```
2034 \newcommand\SB@repcolon{{%
2035   \usefont{OT1}{cmss}{m}{n}\selectfont%
2036   \ifchorded%
2037     \baselineskip.5\SB@dimen%
2038     \vbox{\hbox{:}\hbox{:}\kern.5\p@}%
```

105

```
2039    \else%
2040        \raise.5\p@\hbox{:}%
2041    \fi%
2042 }}
```

\lrep Create a begin-repeat sign.

```
2043 \newcommand\lrep{%
2044    \SB@dimen\baselineskip%
2045    \advance\SB@dimen-2\p@%
2046    \vrule\@width1.5\p@\@height\SB@dimen\@depth\p@%
2047    \kern1.5\p@%
2048    \vrule\@width.5\p@\@height\SB@dimen\@depth\p@%
2049    \SB@repcolon%
2050 }
```

\rrep Create an end-repeat sign.

```
2051 \newcommand\rrep{%
2052    \SB@dimen\baselineskip%
2053    \advance\SB@dimen-2\p@%
2054    \SB@repcolon%
2055    \vrule\@width.5\p@\@height\SB@dimen\@depth\p@%
2056    \kern1.5\p@%
2057    \vrule\@width1.5\p@\@height\SB@dimen\@depth\p@%
2058 }
```

15.10   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 Chord macros must look ahead in the input stream to see whether this chord is
\ifSB@brokenword 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.

```
2059 \newif\ifSB@wordends
2060 \newif\ifSB@brokenword
```

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

```
2061 \SB@newtoks\SB@lyric
```

<code>\SB@numhyps</code> 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.

```
2062 \SB@newcount\SB@numhyps
```

<code>\SB@lyricnohyp</code> 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.

```
2063 \SB@newtoks\SB@lyricnohyp
```

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

```
2064 \SB@newbox\SB@lyricbox
2065 \SB@newbox\SB@chordbox
```

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

```
2066 \newcommand\SB@chbstok{}
```

<code>\SB@setchord</code> The following macro typesets its argument as a chord and stores the result in box `\SB@chordbox` for later placement into the document. The hat token (^) is redefined so that outside of math mode it suppresses chord memorization, but inside of math mode it retains its usual superscript meaning. If memorization is active, the chord's token sequence is stored in the current replay register. If `\SB@chordbox` is non-empty, the new chord is appended to it rather than replacing it. This allows consecutive chords not separated by whitespace to be typeset as a single chord sequence atop a common lyric.

```
2067 \newcommand\SB@setchord{}
2068 {
2069   \catcode`^\active
2070   \gdef\SB@setchord#1{%
2071     \SB@gettabindtrue\SB@nohattrue%
2072     \setbox\SB@chordbox\hbox{%
2073       \unhbox\SB@chordbox%
2074       \begingroup%
2075         \ifSB@trackch%
2076           \let\SB@activehat\SB@hat@tr%
2077         \else%
2078           \let\SB@activehat\SB@hat@notr%
2079         \fi%
2080         \let^\SB@activehat%
2081         \printchord{%
2082           \ifSB@firstchord\else\kern.15em\fi%
2083           \vphantom/%
2084           \transposehere{#1}%
2085           \kern.2em%
2086         }%
2087       \endgroup%
2088     }%
2089     \SB@gettabindfalse%
2090     \ifSB@trackch\ifSB@nohat%
2091       \global\SB@creg\expandafter{\the\SB@creg#1\\}%
2092     \fi\fi%
```

```
2093      \let\SB@noreplay\@firstofone%
2094    }
2095 }
```

\SB@outertest The lyric-scanning code must preemptively determine whether the next token
\SB@otesta is a macro declared \outer before it tries to accept that token as an argument.
\SB@otestb Otherwise TeX will abort with a parsing error. Macros declared \outer are not
allowed in arguments, so determining whether a token is \outer is a delicate
process. The following does so by consulting \meaning. A macro can be identified
as \outer if its meaning has the word "\outer" before the first colon.

```
2096 \newcommand\SB@outertest{%
2097   \expandafter\SB@otesta\meaning\SB@next:\SB@otesta%
2098 }
2099 \newcommand\SB@otesta{}
2100 \edef\SB@otesta#1:#2\SB@otesta{%
2101   \noexpand\SB@otestb%
2102   #1\string\outer%
2103   \noexpand\SB@otestb%
2104 }
2105 \newcommand\SB@otestb{}
2106 \expandafter\def\expandafter\SB@otestb%
2107 \expandafter#\expandafter1\string\outer#2\SB@otestb{%
2108   \def\SB@temp{#2}%
2109   \ifx\SB@temp\@empty\SB@testfalse\else\SB@testtrue\fi%
2110 }
```

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

```
2111 \newcommand\SB@UTFtest{}
2112 \edef\SB@UTFtest#1{%
2113   \noexpand\expandafter%
2114   \noexpand\SB@@UTFtest%
2115   \noexpand\meaning#1%
2116   \string\UTFviii@zero@octets%
2117   \noexpand\SB@@UTFtest%
2118 }
2119 \newcommand\SB@U@two{\global\SB@cnt\tw@}
2120 \newcommand\SB@U@three{\global\SB@cnt\thr@@}
2121 \newcommand\SB@U@four{\global\SB@cnt4\relax}
2122 \newcommand\SB@@UTFtest{}
2123 {\escapechar\m@ne
2124   \xdef\SB@temp{\string\@octets}}
2125 \edef\SB@temp{##1\string\UTFviii@##2\SB@temp}
2126 \expandafter\def\expandafter\SB@@UTFtest\SB@temp#3\SB@@UTFtest{%
2127   \SB@cnt\z@%
2128   {\csname SB@U@#2\endcsname}%
2129 }
```

\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.

```
2130 \newcommand\SB@declare[3]{%
2131   \afterassignment\iffalse\let\SB@next= #3\relax\fi%
2132   \SB@UTFtest\SB@next%
2133   \ifcase\SB@cnt%
2134     \ifcat\noexpand#3\relax%
2135       \SB@addNtest\SB@macrotests#1#2#3%
2136     \else\ifcat\noexpand#3.%
2137       \SB@addDtest\SB@othertests#1#2%
2138     \else\ifcat\noexpand#3A%
2139       \SB@addDtest\SB@lettertests#1#2%
2140     \else%
2141       \SB@addDtest\relax0#2%
2142     \fi\fi\fi%
2143   \or%
2144     \SB@addNtest\SB@macrotests#1#2#3%
2145   \else%
2146     \SB@addMtest\SB@multitests#1#2#3\relax\relax\relax%
2147   \fi%
2148 }
2149 \newcommand\DeclareLyricChar{\SB@declare\SB@testtrue0}
2150 \newcommand\DeclareNonLyric{\SB@declare\SB@testfalse\SB@testfalse}
2151 \newcommand\DeclareNoHyphen{\SB@declare\SB@testfalse\SB@testtrue}
```

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

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

```
2152 \newcommand\SB@lettertests{}
2153 \newcommand\SB@macrotests{}
2154 \newcommand\SB@multitests{}
2155 \newcommand\SB@othertests{}
```

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 whether it matches.

\SB@addDtest A definition-test: The test succeeds if the next lyric token has the same meaning (at test-time) of the non-macro, non-active character token that was given to the **\Declare** macro.

```
2156 \newcommand\SB@addDtest[3]{%
2157   \ifx0#2\else%
2158     \def#1{{\csname SB@!\meaning\SB@next\endcsname}}%
2159     \expandafter\def\csname SB@!\meaning\SB@next\endcsname{\global#2}%
2160   \fi%
2161   \ifx0#3\else%
2162     \expandafter\def\csname SB@HT@\meaning\SB@next\endcsname{\global#3}%
2163   \fi%
2164 }
```

\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.

```
2165 \newcommand\SB@addNtest[4]{%
2166   \ifx0#2\else%
2167     \def#1{{\csname SB@!\SB@nextname\endcsname}}%
2168     \expandafter\def\csname SB@!\string#4\endcsname{\global#2}%
2169   \fi%
2170   \ifx0#3\else%
2171     \expandafter\def\csname SB@HT@\string#4\endcsname{\global#3}%
2172   \fi%
2173 }
```

\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.

```
2174 \newcommand\SB@addMtest[7]{%
2175   \edef\SB@temp{%
2176     \string#4%
2177     \ifx\relax#5\else\string#5\fi%
2178     \ifx\relax#6\else\string#6\fi%
2179     \ifx\relax#7\else\string#7\fi%
2180   }%
2181   \ifx0#2\else%
2182     \def#1{{\csname SB@!\SB@nextname\endcsname}}%
2183     \expandafter\def\csname SB@!\SB@temp\endcsname{\global#2}%
2184   \fi%
2185   \ifx0#3\else%
2186     \expandafter\def\csname SB@HT@\SB@temp\endcsname{\global#3}%
2187   \fi%
2188 }
```

The following code declares the common intra-word macros provided by TeX (as listed on p. 52 of The TeXbook) to be lyric-continuing.

```
2189 \DeclareLyricChar\`
2190 \DeclareLyricChar\'
2191 \DeclareLyricChar\^
2192 \DeclareLyricChar\"
2193 \DeclareLyricChar\~
2194 \DeclareLyricChar\=
2195 \DeclareLyricChar\.
2196 \DeclareLyricChar\u
2197 \DeclareLyricChar\v
2198 \DeclareLyricChar\H
2199 \DeclareLyricChar\t
2200 \DeclareLyricChar\c
2201 \DeclareLyricChar\d
2202 \DeclareLyricChar\b
2203 \DeclareLyricChar\oe
2204 \DeclareLyricChar\OE
2205 \DeclareLyricChar\ae
2206 \DeclareLyricChar\AE
2207 \DeclareLyricChar\aa
```

```
2208 \DeclareLyricChar\AA
2209 \DeclareLyricChar\o
2210 \DeclareLyricChar\O
2211 \DeclareLyricChar\l
2212 \DeclareLyricChar\L
2213 \DeclareLyricChar\ss
2214 \DeclareLyricChar\i
2215 \DeclareLyricChar\j
2216 \DeclareLyricChar\/
2217 \DeclareLyricChar\-
2218 \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.

```
2219 \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.

```
2220 \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.

```
2221 \newcommand\SB@chord{\SB@begincname\SB@@chord}
```

`\SB@begincname` While parsing a chord name, certain characters such as `#` and `&` are temporarily set
`\SB@endcname` 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 ⟨*return*⟩ 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.

```
2222 \newcommand\SB@begincname{}
2223 {\catcode'\^^M\active
2224  \gdef\SB@begincname{%
2225    \begingroup%
2226      \catcode'##\active\catcode'&\active%
2227      \catcode':12\relax%
2228      \catcode'\^^M\active\SB@outer\def^^M{}%
2229      \SB@outer\def\[{}%
2230      \chordlocals\relax%
2231  }
2232 }
2233 \newcommand\SB@endcname{}
2234 \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, which are typically not lyric-ending. To distinguish the real `~` from redefined `~`, we need

111

to create a macro whose definition is the non-breaking space definition normally assigned to ~.

2235 \newcommand*\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.

2236 \newif\ifSB@firstchord\SB@firstchordtrue

\SB@@chord Finish processing the chord name and then begin scanning the implicit lyric argument that follows it. This is the main entrypoint to the lyric-scanner code.

```
2237 \newcommand*\SB@@chord{}
2238 \def\SB@@chord#1]{%
2239   \SB@endcname%
2240   \ifSB@firstchord%
2241     \setbox\SB@lyricbox\hbox{\kern\SB@tabindent}%
2242     \global\SB@tabindent\z@%
2243     \SB@lyric{}%
2244     \SB@numhyps\z@%
2245     \SB@spcinit%
2246     \setbox\SB@chordbox\box\voidb@x%
2247   \fi%
2248   \SB@setchord{#1}%
2249   \SB@firstchordfalse%
2250   \let\SB@dothis\SB@chstart%
2251   \SB@chscan%
2252 }
```

\MultiwordChords The \SB@spcinit macro is invoked at the beginning of the lyric scanning process.
\SB@spcinit By default it does nothing, but if \MultiwordChords is invoked, it initializes the lyric-scanner state to process spaces as part of lyrics.

```
2253 \newcommand\SB@spcinit{}
2254 \newcommand\MultiwordChords{%
2255   \def\SB@spcinit{%
2256     \let\SB@chdone\SB@chlyrdone%
2257     \let\SB@chimpspace\SB@chnxtdone%
2258     \let\SB@chexpspace\SB@chnxtdone%
2259     \let\SB@chespace\SB@chendspace%
2260   }%
2261 }
```

\SB@chscan This is the main loop of the lyric-scanner. Peek ahead at the next token without
\SB@chmain consuming it, then execute a loop body based on the current state (\SB@dothis), and finally go to the next iteration (\SB@donext).

```
2262 \newcommand\SB@chscan{%
2263   \let\SB@nextname\relax%
2264   \futurelet\SB@next\SB@chmain%
2265 }
2266 \newcommand\SB@chmain{\SB@dothis\SB@donext}
```

\SB@chnxtrelax To shorten the lyric parser macros that follow and thereby improve their speed,
\SB@chnxtstep we here define some abbreviations for common logic in untaken branches.
\SB@chnxtdone
2267 \newcommand\SB@chnxtrelax{\let\SB@donext\relax}
2268 \newcommand\SB@chnxtstep{\let\SB@donext\SB@chstep}
2269 \newcommand\SB@chnxtdone{\let\SB@donext\SB@chdone}

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.

```
2270 \newcommand\SB@chstart{%
2271   \ifx\SB@next\[\SB@chnxtrelax%
2272   \else\ifx\SB@next\SB@activehat\SB@chnxtrelax%
2273   \else\ifx\SB@next\ch\SB@chnxtrelax%
2274   \else\ifx\SB@next\mch\SB@chnxtrelax%
2275   \else\ifx\SB@next'\SB@chnxtstep%
2276   \else\ifx\SB@next'\SB@chnxtstep%
2277   \else\ifx\SB@next"\SB@chnxtstep%
2278   \else%
2279     \the\SB@lyric%
2280     \SB@lyric{}%
2281     \SB@firstchordtrue%
2282     \let\SB@dothis\SB@chnorm%
2283     \SB@chnorm%
2284   \fi\fi\fi\fi\fi\fi\fi%
2285 }
```

`\SB@chnorm`   First, check to see whether the lyric token is a letter. Since that's the most common case, we do this check first for speed.

```
2286 \newcommand\SB@chnorm{%
2287   \ifcat\noexpand\SB@next A%
2288     \SB@testtrue\SB@lettertests%
2289     \ifSB@test%
2290       \SB@chespace\SB@chnxtstep%
2291     \else%
2292       \SB@chnxtdone%
2293     \fi%
2294   \else%
2295     \SB@chtrymacro%
2296   \fi%
2297 }
```

`\SB@chtrymacro`   Next, check to see whether 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.

```
2298 \newcommand\SB@chtrymacro{%
2299   \ifcat\noexpand\SB@next\relax%
2300     \SB@chmacro%
2301   \else%
```

113

```
2302        \SB@chother%
2303    \fi%
2304 }
```

`\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.

```
2305 \newcommand\SB@chother{%
2306    \ifcat\noexpand\SB@next\@sptoken%
2307        \SB@chexpspace%
2308    \else\ifcat\noexpand\SB@next\bgroup%
2309        \SB@chespace\let\SB@donext\SB@chbgroup%
2310    \else\ifcat\noexpand\SB@next\egroup%
2311        \SB@chespace\let\SB@donext\SB@chegroup%
2312    \else\ifx\SB@next-%
2313        \SB@numhyps\@ne\relax%
2314        \SB@lyricnohyp\expandafter{\the\SB@lyric}%
2315        \let\SB@dothis\SB@chhyph%
2316        \SB@chespace\SB@chnxtstep%
2317    \else\ifcat\noexpand\SB@next.%
2318        \SB@testtrue\SB@othertests%
2319        \ifSB@test%
2320            \SB@chespace\SB@chnxtstep%
2321        \else%
2322            \SB@chnxtdone%
2323        \fi%
2324    \else%
2325        \SB@chespace\SB@chnxtstep%
2326    \fi\fi\fi\fi\fi%
2327 }
```

`\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.

```
2328 \newcommand\SB@chmacro{%
2329    \SB@outertest%
2330    \ifSB@test%
2331        \SB@chnxtdone%
2332    \else%
2333        \let\SB@donext\SB@chgetname%
2334    \fi%
2335 }
```

`\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.

```
2336 \newcommand\SB@chgetname[1]{%
2337    \edef\SB@nextname{\string#1}%
2338    \SB@@chmacro\SB@donext#1%
2339 }
```

114

\SB@@chmacro The lyric-scanner has encountered a non-\outer macro or active character. Its
\stringified name has been stored in \SB@nextname. Test to see whether 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.

```
2340 \newcommand\SB@@chmacro{%
2341   \ifx\SB@next\SB@activehat%
2342     \SB@chnxtdone%
2343   \else\ifx\SB@next\SB@par%
2344     \SB@chnxtdone%
2345   \else\ifx\SB@next\measurebar%
2346     \SB@chmbar%
2347   \else\ifx\SB@next\mbar%
2348     \SB@chmbar%
2349   \else\ifx\SB@next\ch%
2350     \SB@chespace\let\SB@donext\SB@chlig%
2351   \else\ifx\SB@next\mch%
2352     \SB@chespace\let\SB@donext\SB@mchlig%
2353   \else\ifx\SB@next\ %
2354     \SB@chimpspace%
2355   \else\ifx\SB@next\SB@nbsp%
2356     \SB@chimpspace%
2357   \else%
2358     \SB@UTFtest\SB@next%
2359     \ifcase\SB@cnt\SB@chothermac%
2360     \or\or\SB@chespace\let\SB@donext\SB@chsteptwo%
2361     \or\SB@chespace\let\SB@donext\SB@chstepthree%
2362     \or\SB@chespace\let\SB@donext\SB@chstepfour\fi%
2363   \fi\fi\fi\fi\fi\fi\fi\fi%
2364 }
```

\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 interna-
tional character. First, check the macro's name (stored in \SB@nextname) to see
whether 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.

```
2365 \newcommand\SB@chothermac{%
2366   \SB@testfalse%
2367   \afterassignment\iffalse%
2368   \SB@cnt\expandafter`\SB@nextname x\fi%
2369   \ifnum\the\catcode\SB@cnt=\z@\else\SB@testtrue\fi%
2370   \SB@macrotests%
2371   \ifSB@test%
2372     \SB@chespace\SB@chnxtstep%
2373   \else%
2374     \SB@chnxtdone%
2375   \fi%
2376 }
```

\SB@chstep We've encountered one or more tokens that should be included in the lyric text.
\SB@chsteptwo
\SB@chstepthree
\SB@chstepfour
\SB@chmulti
\SB@chmstop

115

(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.

```
2377 \newcommand\SB@chstep[1]{%
2378   \SB@lyric\expandafter{\the\SB@lyric#1}%
2379   \SB@chscan%
2380 }
2381 \newcommand\SB@chsteptwo[2]{\SB@chmulti{#1#2}{\string#1\string#2}}
2382 \newcommand\SB@chstepthree[3]{%
2383   \SB@chmulti{#1#2#3}{\string#1\string#2\string#3}%
2384 }
2385 \newcommand\SB@chstepfour[4]{%
2386   \SB@chmulti{#1#2#3#4}{\string#1\string#2\string#3\string#4}%
2387 }
2388 \newcommand\SB@chmulti[2]{%
2389   \def\SB@next{#1}%
2390   \edef\SB@nextname{#2}%
2391   \SB@testtrue\SB@multitests%
2392   \ifSB@test%
2393     \SB@lyric\expandafter{\the\SB@lyric#1}%
2394     \expandafter\SB@chscan%
2395   \else%
2396     \expandafter\SB@chmstop%
2397   \fi%
2398 }
2399 \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.

```
2400 \newcommand\SB@chhyph{%
2401   \ifx\SB@next-%
2402     \advance\SB@numhyps\@ne\relax%
2403     \SB@chnxtstep%
2404   \else%
2405     \SB@chnxtdone%
2406   \fi%
2407 }
```

\SB@chimpspace   We've encountered an implicit or explicit space. Normally this just ends the lyric,
\SB@chexpspace   but if \MultiwordChords is active, these macros both get redefined to process the space.

```
2408 \newcommand\SB@chimpspace{}
2409 \let\SB@chimpspace\SB@chnxtdone
2410 \newcommand\SB@chexpspace{}
2411 \let\SB@chexpspace\SB@chnxtdone
```

\SB@chespace   The \SB@chespace macro gets invoked by the lyric-scanner just before a non-space
\SB@chendspace   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,

116

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.

```
2412 \newcommand\SB@chespace{}
2413 \newcommand\SB@chendspace{%
2414   \let\SB@chdone\SB@chlyrdone%
2415   \def\SB@chexpspace{\SB@chbspace\SB@chexpspace}%
2416   \def\SB@chimpspace{\SB@chbspace\SB@chimpspace}%
2417   \def\SB@chespace{}%
2418 }
```

\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.

```
2419 \newcommand\SB@chbspace{%
2420   \setbox\SB@lyricbox\hbox{%
2421     \unhbox\SB@lyricbox%
2422     \the\SB@lyric%
2423   }%
2424   \SB@lyric{}%
2425   \ifdim\wd\SB@lyricbox<\wd\SB@chordbox%
2426     \let\SB@chbstok= \SB@next%
2427     \def\SB@chexpspace{\let\SB@donext\SB@chgetspace}%
2428     \let\SB@chimpspace\SB@chnxtstep%
2429     \let\SB@chespace\SB@chendspace%
2430     \let\SB@chdone\SB@chspcdone%
2431   \else%
2432     \let\SB@chimpspace\SB@chnxtdone%
2433     \let\SB@chexpspace\SB@chnxtdone%
2434   \fi%
2435 }
2436 \newcommand\SB@chgetspace{%
2437   \SB@appendsp\SB@lyric%
2438   \let\SB@nextname\relax%
2439   \afterassignment\SB@chscan%
2440   \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.

```
2441 \newcommand\SB@chmbar{%
2442   \ifmeasures%
```

```
2443        \SB@chnxtdone%
2444     \else%
2445        \SB@chespace\SB@chnxtstep%
2446     \fi%
2447 }
```

\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.

```
2448 \newcommand\SB@chbgroup[1]{%
2449     \SB@lyric\expandafter{\the\SB@lyric{#1}}%
2450     \SB@chscan%
2451 }
```

\SB@chegroup   We've encountered an end-group brace whose matching begin-group brace must
\SB@chegrpscan have come before the chord macro itself. This forcibly ends the lyric text. Before
\SB@chegrpmacro stopping, we must set \SB@next to the token following the brace and \SB@nextname
\SB@chegrpouter to its \stringified name so that \SB@emitchord will know whether to add hyphen-
\SB@chegrpname ation. Therefore, we temporarily consume the end-group brace, then scan the next
\SB@chegrpdone token without consuming it, and finally reinsert the end-group brace and stop.

```
2452 \newcommand\SB@chegroup{%
2453     \let\SB@nextname\relax%
2454     \afterassignment\SB@chegrpscan%
2455     \let\SB@next= }
2456 \newcommand\SB@chegrpscan{%
2457     \futurelet\SB@next\SB@chegrpmacro%
2458 }
2459 \newcommand\SB@chegrpmacro{%
2460     \ifcat\noexpand\SB@next\relax%
2461        \expandafter\SB@chegrpouter%
2462     \else%
2463        \expandafter\SB@chegrpdone%
2464     \fi%
2465 }
2466 \newcommand\SB@chegrpouter{%
2467     \SB@outertest%
2468     \ifSB@test%
2469        \expandafter\SB@chegrpdone%
2470     \else%
2471        \expandafter\SB@chegrpname%
2472     \fi%
2473 }
2474 \newcommand\SB@chegrpname[1]{%
2475     \edef\SB@nextname{\string#1}%
2476     \SB@chegrpdone#1%
2477 }
2478 \newcommand\SB@chegrpdone{\SB@chdone\egroup}
```

\SB@chlig   We've encountered a \ch chord-over-ligature macro, or an \mch measurebar-and-
\SB@mchlig 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 whether we decide to add hyphenation.)
            Then end the lyric text here.

```
2479 \newcommand\SB@chlig[5]{%
```

```
2480    \gdef\SB@ligpre{{#3}}%
2481    \gdef\SB@ligpost{\[#2]{#4}}%
2482    \gdef\SB@ligfull{%
2483      \[\SB@noreplay{\hphantom{{\lyricfont\relax#3}}}#2]{#5}%
2484    }%
2485    \SB@chdone%
2486 }
2487 \newcommand\SB@mchlig[5]{%
2488    \SB@lyric\expandafter{\the\SB@lyric#3}%
2489    \let\SB@next\measurebar%
2490    \edef\SB@nextname{\string\measurebar}%
2491    \gdef\SB@ligpost{\measurebar\[#2]{#4}}%
2492    \gdef\SB@ligfull{\measurebar\[#2]{#4}}%
2493    \SB@chdone%
2494 }
```

\SB@chdone The \SB@chdone macro is invoked when we've decided to end the lyric text (usu-
\SB@chlyrdone ally because we've encountered a non-lyric token). Normally this expands to
\SB@chspcdone \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.

```
2495 \newcommand\SB@chlyrdone{%
2496    \setbox\SB@lyricbox\hbox{%
2497      \unhbox\SB@lyricbox%
2498      \ifnum\SB@numhyps=\@ne%
2499        \the\SB@lyricnohyp%
2500      \else%
2501        \the\SB@lyric%
2502      \fi%
2503    }%
2504    \SB@emitchord%
2505 }
2506 \newcommand\SB@chspcdone{%
2507    \let\SB@nextname\relax%
2508    \let\SB@next= \SB@chbstok%
2509    \expandafter\SB@emitchord\the\SB@lyric%
2510 }
2511 \newcommand\SB@chdone{}
2512 \let\SB@chdone\SB@chlyrdone
```

\SB@ligpre The following three macros record arguments passed to a \ch macro that concludes
\SB@ligpost the lyric text of the \[] macro currently being processed.
\SB@ligfull
```
2513 \newcommand\SB@ligpre{}
2514 \newcommand\SB@ligpost{}
2515 \newcommand\SB@ligfull{}
```

\SB@clearlig Clear all ligature-chord registers.
```
2516 \newcommand\SB@clearlig{%
2517    \gdef\SB@ligpre{}%
2518    \gdef\SB@ligpost{}%
2519    \gdef\SB@ligfull{}%
2520 }
```

119

15.11 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 §**??** to the token that immediately follows the lyric under this chord, to determine whether the lyric text ends on a word boundary.

```
2521 \newcommand\SB@emitchord{%
2522   \ifSB@inverse\else\ifSB@inchorus\else\SB@errchord\fi\fi%
2523   \SB@testfalse%
2524   \ifcat\noexpand\SB@next\@sptoken\SB@testtrue\fi%
2525   \ifcat\noexpand\SB@next.\SB@testtrue\fi%
2526   \ifx\SB@next\SB@par\SB@testtrue\fi%
2527   \ifx\SB@next\egroup\SB@testtrue\fi%
2528   \ifx\SB@next\endgroup\SB@testtrue\fi%
2529   {\csname%
2530     SB@HT@\ifx\SB@nextname\relax\meaning\SB@next\else\SB@nextname\fi%
2531    \endcsname}%
2532   \ifSB@test\SB@wordendstrue\else\SB@wordendsfalse\fi%
```

Next, compare the width of the lyric to the width of the chord to determine whether 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.

```
2533   \SB@dimen\wd\SB@chordbox%
2534   \ifvmode\leavevmode\fi%
2535   \SB@brokenwordfalse%
2536   \ifdim\wd\SB@lyricbox>\z@%
2537     \ifdim\SB@dimen>\wd\SB@lyricbox%
2538       \ifSB@wordends\else\SB@brokenwordtrue\fi%
2539     \fi%
2540   \else%
2541     \SB@skip\lastskip%
2542     \unskip\penalty200\hskip\SB@skip%
2543   \fi%
2544   \ifnum\SB@numhyps>\z@%
2545     \ifnum\SB@numhyps>\@ne%
2546       \SB@brokenwordfalse%
2547     \else%
2548       \SB@brokenwordtrue%
2549     \fi%
2550   \fi%
```

If lyrics are suppressed on this line (e.g., by using `\nolyrics`), then just typeset the chord text on the natural baseline.

```
2551    \SB@testfalse%
2552    \ifnolyrics\ifdim\wd\SB@lyricbox=\z@\SB@testtrue\fi\fi%
2553    \ifSB@test%
2554      \unhbox\SB@chordbox%
2555      \gdef\SB@temp{\expandafter\SB@clearlig\SB@ligfull}%
2556    \else%
```

Otherwise, typeset the chord above the lyric on a double-height line.

```
2557      \vbox{\clineparams\relax%
2558        \ifSB@brokenword%
2559          \global\setbox\SB@lyricbox\hbox{%
2560            \unhbox\SB@lyricbox%
2561            \SB@ligpre%
2562          }%
2563          \SB@maxmin\SB@dimen<{\wd\SB@lyricbox}%
2564          \advance\SB@dimen.5em%
2565          \hbox to\SB@dimen{\unhbox\SB@chordbox\hfil}%
2566          \hbox to\SB@dimen{%
2567            \unhcopy\SB@lyricbox\hfil
2568            \ifnum\hyphenchar\font>\m@ne\char\hyphenchar\font\hfil\fi%
2569          }%
2570          \global\SB@cnt\@m%
2571          \gdef\SB@temp{\expandafter\SB@clearlig\SB@ligpost}%
2572        \else%
2573          \box\SB@chordbox%
2574          \hbox{%
2575            \unhcopy\SB@lyricbox%
2576            \global\SB@cnt\spacefactor%
2577            \hfil%
2578          }%
2579          \gdef\SB@temp{\expandafter\SB@clearlig\SB@ligfull}%
2580        \fi%
2581      }%
```

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.

```
2582      \ifSB@wordends%
2583        \ifdim\wd\SB@lyricbox>\z@\else\nobreak\fi%
2584      \else%
2585        \penalty%
2586          \ifnum\SB@numhyps>\z@\exhyphenpenalty%
2587          \else\ifSB@brokenword\hyphenpenalty%
2588          \else\@M\fi\fi%
2589        \spacefactor\SB@cnt%
2590      \fi%
2591    \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.

```
2592    \SB@temp%
2593 }
```

\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.

```
2594 \newcommand\SB@accidental[1]{{%
2595    \/%
2596    \m@th#1%
2597    \SB@dimen-\fontdimen\@ne\font%
2598    \advance\SB@dimen.088142\p@%
2599    \ifdim\SB@dimen<\z@%
2600       \kern\f@size\SB@dimen%
2601    \fi%
2602 }}
```

\sharpsymbol   When changing the sharp or flat symbol, change these macros rather than changing
\flatsymbol   \shrp or \flt. This will ensure that other shortcuts like # and & will reflect your change.

```
2603 \newcommand\sharpsymbol{\ensuremath{^\#}}
2604 \newcommand\flatsymbol{\raise.5ex\hbox{{\SB@flatsize$\flat$}}}
```

\shrp   These macros typeset sharp and flat symbols.
\flt
```
2605 \newcommand\shrp{\SB@accidental\sharpsymbol}
2606 \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.

```
2607 \newcommand\DeclareFlatSize[2]{%
2608    \expandafter\xdef\csname SB@flatsize@#1\endcsname{#2}%
2609 }
2610 \DeclareFlatSize\@vpt\@vpt
2611 \DeclareFlatSize\@vipt\@vipt
2612 \DeclareFlatSize\@viipt\@vipt
2613 \DeclareFlatSize\@viiipt\@viipt
2614 \DeclareFlatSize\@ixpt\@viiipt
2615 \DeclareFlatSize\@xpt\@ixpt
2616 \DeclareFlatSize\@xipt\@xpt
2617 \DeclareFlatSize\@xiipt\@xipt
2618 \DeclareFlatSize\@xivpt\@xiipt
2619 \DeclareFlatSize\@xviipt\@xivpt
```

```
2620 \DeclareFlatSize\@xxpt\@xviipt
2621 \DeclareFlatSize\@xxvpt\@xxpt
```

\SB@flatsize   Select the correct flat symbol size based on the current font size.

```
2622 \newcommand\SB@flatsize{%
2623   \@ifundefined{SB@flatsize@\f@size}{}{%
2624     \expandafter\fontsize%
2625       \csname SB@flatsize@\f@size\endcsname\f@baselineskip%
2626     \selectfont%
2627   }%
2628 }
```

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   The `\ch` macro puts a chord atop a ligature without breaking the ligature. Normally
\SB@ch   this just means placing the chord midway over the unbroken ligature (ignoring the
\SB@ch@on   third argument completely). However, when a previous chord macro encounters it
\SB@@ch   while scanning ahead in the input stream to parse its lyric, the `\ch` macro itself
\SB@@@ch   is not actually expanded at all. Instead, the chord macro scans ahead, spots the
\SB@ch@off `\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.

```
2629 \newcommand\ch{\SB@ch}
2630 \newcommand\SB@ch{}
2631 \newcommand\SB@ch@on{\SB@begincname\SB@@ch}
2632 \newcommand*\SB@@ch[1]{\SB@endcname\SB@@@ch{#1}}
2633 \newcommand*\SB@@@ch[4]{\[\SB@noreplay{\hphantom{#2}}#1]#4}
2634 \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
```
2635 \newcommand\mch{\SB@mch}
2636 \newcommand\SB@mch{}
2637 \newcommand*\SB@mch@m[4]{#2\measurebar#3}
2638 \newcommand\SB@mch@on{\SB@begincname\SB@@mch}
2639 \newcommand*\SB@@mch[1]{\SB@endcname\SB@@@mch{#1}}
2640 \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.

```
2641 \newcommand\SB@activehat{%
2642   \ifmmode^\else\expandafter\SB@rechord\fi%
2643 }
```

123

\SB@hat@tr In verses/choruses where chords are being memorized, `\SB@activehat` gets set to this definition, which marks the current chord as immune to memorization.

```
2644 \newcommand\SB@hat@tr{%
2645   \ifmmode^\else\global\SB@nohatfalse\fi%
2646 }
```

\SB@hat@notr In verses/choruses where chords are being replayed, `\SB@activehat` get set to the following, which replays the next memorized chord and subjects it to any required transposition and/or note conversion.

```
2647 \newcommand\SB@hat@notr{%
2648   \ifmmode^\else%
2649     \SB@lop\SB@ctail\SB@toks%
2650     \expandafter\transposehere\expandafter{\the\SB@toks}%
2651   \fi%
2652 }
```

\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.

```
2653 \newcommand\SB@loadactives{}
2654 {
2655   \catcode`\&\active
2656   \catcode`\#\active
2657   \catcode`\|\active
2658   \catcode`\^\active
2659   \global\let&\flt
2660   \global\let#\shrp
2661   \global\let|\measurebar
2662   \global\let^\SB@activehat
2663   \gdef\SB@loadactives{%
2664     \catcode`\^\ifchorded\active\else9 \fi%
2665     \catcode`\|\ifmeasures\active\else9 \fi%
2666     \def\[{\SB@bracket}%
2667   }
2668 }
```

15.12 Chord Replaying

\SB@trackch While inside a verse where the chord history is being remembered for future verses, `\SB@trackch` is true.

```
2669 \newif\ifSB@trackch
```

\SB@cr@ Reserve token registers to record a history of the chords seen in a verse.

```
2670 \SB@newtoks\SB@cr@
2671 \SB@newtoks\SB@ctail
```

\SB@creg The following control sequence equals the token register being memorized into or replayed from.

```
2672 \newcommand\SB@creg{}
```

\newchords Allocate a new chord-replay register to hold memorized chords.

```
2673 \newcommand\newchords[1]{%
2674   \@ifundefined{SB@cr@#1}{%
2675     \expandafter\SB@newtoks\csname SB@cr@#1\endcsname%
2676     \global\csname SB@cr@#1\endcsname{\\}%
2677   }{\SB@errdup{#1}}%
2678 }
```

\memorize Saying \memorize throws out any previously memorized list of chords and starts
\SB@memorize memorizing chords until the end of the current verse or chorus.

```
2679 \newcommand\memorize{%
2680   \@ifnextchar[\SB@memorize{\SB@memorize[]}%
2681 }
2682 \newcommand\SB@memorize{}
2683 \def\SB@memorize[#1]{%
2684   \@ifundefined{SB@cr@#1}{\SB@errreg{#1}}{%
2685     \SB@trackchtrue%
2686     \global\expandafter\let\expandafter\SB@creg%
2687       \csname SB@cr@#1\endcsname%
2688     \global\SB@creg{\\}%
2689   }%
2690 }
```

\replay Saying \replay stops any memorization and begins replaying memorized chords.
\SB@replay
\SB@@replay

```
2691 \newcommand\replay{\@ifnextchar[\SB@replay\SB@@replay}
2692 \newcommand\SB@replay{}
2693 \def\SB@replay[#1]{%
2694   \@ifundefined{SB@cr@#1}{\SB@errreg{#1}}{%
2695     \SB@trackchfalse%
2696     \global\expandafter\let\expandafter\SB@creg%
2697       \csname SB@cr@#1\endcsname%
2698     \global\SB@ctail\SB@creg%
2699   }%
2700 }
2701 \newcommand\SB@@replay{%
2702   \SB@trackchfalse%
2703   \global\SB@ctail\SB@creg%
2704 }
```

\SB@rechord Replay the same chord that was in a previous verse.
\SB@@rechord

```
2705 \newcommand\SB@rechord{}
2706 \newcommand\SB@@rechord{%
2707   \SB@ifempty\SB@ctail{%
2708     \SB@errreplay%
2709     \SB@toks{}%
2710     \let\SB@donext\@gobble%
2711   }{%
2712     \SB@lop\SB@ctail\SB@toks%
2713     \let\SB@donext\SB@chord%
2714     \let\SB@noreplay\@gobble%
2715   }%
2716   \expandafter\SB@donext\the\SB@toks]%
2717 }
```

\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.

```
2718 \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.

```
2719 \newcommand\SB@noreplay{}
2720 \let\SB@noreplay\@firstofone
```

15.13 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.

```
2721 \newlength\SB@fretwidth
2722 \setlength\SB@fretwidth{6\p@}
```

\SB@fretnum Typeset a fret number to appear to the left of the diagram.

```
2723 \newcommand\SB@fretnum[1]{{%
2724   \sffamily\fontsize\@xpt\@xpt\selectfont#1%
2725 }}
```

\SB@onfret Typeset one string of one fret with ⟨arg1⟩ typeset overtop of it (usually a dot or nothing at all).

```
2726 \newcommand\SB@onfret[1]{%
2727   \kern.5\SB@fretwidth\kern-.2\p@%
2728   \vrule\@height6\p@%
2729   \kern-.2\p@\kern-.5\SB@fretwidth%
2730   \hbox to\SB@fretwidth{\hfil#1\hfil}%
2731 }
```

\SB@atopfret Typeset material (given by ⟨arg1⟩) to be placed above a string in the tablature diagram.

```
2732 \newcommand\SB@atopfret[1]{%
2733   \hbox to\SB@fretwidth{\hfil#1\hfil}%
2734 }
```

\SB@fretbar Typeset a horizontal fret bar of width \SB@dimen.

```
2735 \newcommand\SB@fretbar{%
2736   \nointerlineskip%
2737   \hbox to\SB@dimen{%
2738     \advance\SB@dimen-\SB@fretwidth%
2739     \advance\SB@dimen.4\p@%
2740     \hfil%
2741     \vrule\@width\SB@dimen\@height.4\p@\@depth\z@%
2742     \hfil%
2743   }%
2744   \nointerlineskip%
2745 }
```

`\SB@topempty` Above a string in a tablature diagram there can be nothing, an ×, or an ○.

```
\SB@topX 2746 \newcommand\SB@topempty{\SB@atopfret\relax}
\SB@topO 2747 \newcommand\SB@topX{\SB@atopfret{%
         2748   \hbox{%
         2749     \kern-.2\p@%
         2750     \fontencoding{OMS}\fontfamily{cmsy}%
         2751     \fontseries{m}\fontshape{n}%
         2752     \fontsize\@viipt\@viipt\selectfont\char\tw@%
         2753     \kern-.2\p@%
         2754   }%
         2755 }}
         2756 \newcommand\SB@topO{\SB@atopfret{%
         2757   \vrule\@width\z@\@height4.3333\p@\@depth.8333\p@%
         2758   \lower.74\p@\hbox{%
         2759     \fontencoding{OMS}\fontfamily{cmsy}%
         2760     \fontseries{m}\fontshape{n}%
         2761     \fontsize\@xpt\@xpt\selectfont\char14%
         2762   }%
         2763 }}
```

`\SB@doify` Define the macro given in the first argument to equal the fully expanded content
`\SB@@doify` of the second argument, but with `\SB@do` inserted before each token or group.

```
\SB@do 2764 \newcommand\SB@do[1]{}
       2765 \newcommand\SB@doify[2]{%
       2766   \SB@toks{}%
       2767   \edef#1{#2}%
       2768   \expandafter\SB@@doify#1\SB@@doify%
       2769   \edef#1{\the\SB@toks}%
       2770 }
       2771 \newcommand\SB@@doify[1]{%
       2772   \ifx#1\SB@@doify\else%
       2773     \SB@toks\expandafter{\the\SB@toks\SB@do{#1}}%
       2774     \expandafter\SB@@doify%
       2775   \fi%
       2776 }
```

`\SB@allbarres` Reserve a control sequence to remember all the stacks, start control sequences, and
`\SB@dobarre` end control sequences associated with barre delimiter pairs; and a control sequence
to perform an arbitrary action on them.

```
2777 \newcommand\SB@allbarres{}
2778 \newcommand\SB@dobarre{}
```

`\SB@barreI` As we process strings in order, barres in progress can be in one of three states: initial
`\SB@barreN` (`\SB@barreI`), deactivated (`\SB@barreN`), or tentatively activated (`\SB@barreY`).
`\SB@barreY` 2779 \newcommand\SB@barreI{\noexpand\SB@barreI}
```
2780 \newcommand\SB@barreN{\noexpand\SB@barreN}
2781 \newcommand\SB@barreY{\noexpand\SB@barreY}
```

`\SB@lowfret` If we see a lower numbered fret than the current fret within a barre, deactivate
`\SB@@lowfret` the barre. (It has already been shown on an earlier fret.)

```
2782 \newcommand\SB@lowfret{%
2783   \let\SB@dobarre\SB@@lowfret\SB@allbarres%
2784   \SB@fretempty%
```

```
2785 }
2786 \newcommand\SB@@lowfret[3]{{%
2787    \let\SB@barreI\SB@barreN%
2788    \let\SB@barreY\SB@barreN%
2789    \xdef#1{#1}%
2790 }}
```

\SB@bactivate   If we see the current fret within a barre, tentatively activate the barre (unless it is
                already deactivated).

```
2791 \newcommand\SB@bactivate[3]{{%
2792    \let\SB@barreI\SB@barreY%
2793    \xdef#1{#1}%
2794 }}
```

\SB@bbarre   Starting a barre group pushes it onto its stack in the initial state.

```
2795 \newcommand\SB@bbarre[1]{%
2796    \xdef#1{\SB@barreI{\the\SB@cntii}#1}%
2797 }
```

\SB@ebarre   Ending a barre group pops it and draws it if it's active.
\SB@@ebarre
\SB@@@ebarre
```
2798 \newcommand\SB@ebarre[3]{%
2799    \ifx#1\@empty%
2800       \ifnum\SB@cnt=\@ne\SB@errebar#2#3\fi%
2801    \else%
2802       \expandafter\SB@@ebarre#1\SB@@ebarre#1%
2803    \fi%
2804 }
2805 \newcommand\SB@@ebarre{}
2806 \def\SB@@ebarre#1#2#3\SB@@ebarre#4{{%
2807    \gdef#4{#3}%
2808    \let\SB@barreI\@gobble%
2809    \let\SB@barreN\@gobble%
2810    \let\SB@barreY\SB@barre%
2811    #1{#2}%
2812 }}
```

\SB@barreson   Turn barre delimiters on or off, depending on whether we're typesetting the interior
\SB@barresoff or upper part of the tablature diagram.

```
2813 \newcommand\SB@barreson[3]{%
2814    \def#2{\SB@bbarre#1}%
2815    \def#3{\SB@ebarre#1#2#3}%
2816 }
2817 \newcommand\SB@barresoff[3]{\let#2\relax\let#3\relax}
```

\SB@fretempty On a string in a fret diagram there can be nothing or a filled circle.
\SB@fretdot
\SB@@fretdot
```
2818 \newcommand\SB@fretempty{%
2819    \advance\SB@cntii\@ne%
2820    \SB@onfret\relax%
2821 }
2822 \newcommand\SB@fretdot{%
2823    \advance\SB@cntii\@ne%
2824    \let\SB@dobarre\SB@bactivate\SB@allbarres%
2825    \SB@@fretdot%
```

```
2826 }
2827 \newcommand\SB@@fretdot{%
2828   \SB@onfret{%
2829     \fontencoding{OMS}\fontfamily{cmsy}%
2830     \fontseries{m}\fontshape{n}%
2831     \fontsize\@xiipt\@xiipt\selectfont\char15%
2832   }%
2833 }
```

\SB@barre Draw a barre.

```
2834 \newcommand\SB@barre[1]{{%
2835   \SB@dimen\SB@fretwidth%
2836   \multiply\SB@dimen\SB@cntii%
2837   \advance\SB@dimen-#1\SB@fretwidth%
2838   \kern-\SB@dimen%
2839   \SB@@fretdot%
2840   \kern-.5\SB@fretwidth%
2841   \advance\SB@dimen-\SB@fretwidth%
2842   \raise.7pt\hbox{\vrule\@height4.6\p@\@width\SB@dimen}%
2843   \kern-.5\SB@fretwidth%
2844   \SB@@fretdot%
2845 }}
```

\SB@fretend At the end of a barred row in a tablature diagram, we auto-finish any activated barres that weren't explicitly closed by the user.

```
2846 \newcommand\SB@fretend{{%
2847   \let\SB@barreI\@gobble%
2848   \let\SB@barreN\@gobble%
2849   \let\SB@barreY\SB@barre%
2850   \def\SB@dobarre##1##2##3{##1\gdef##1{}}\SB@allbarres%
2851 }}
```

\SB@finger If we're including fingering info in the tablature diagram, then below each string
\SB@X there might be a number.
\SB@Z
\SB@O

```
2852 \newcommand*\SB@X{X}
2853 \newcommand*\SB@Z{0}
2854 \newcommand*\SB@O{O}
2855 \newcommand\SB@finger[1]{%
2856   \def\SB@temp{#1}%
2857   \ifx\SB@temp\SB@X\SB@topempty\else%
2858   \ifx\SB@temp\SB@Z\SB@topempty\else%
2859   \ifx\SB@temp\SB@O\SB@topempty\else%
2860     \SB@atopfret{\sffamily\fontsize\@vipt\@vipt\selectfont#1}%
2861   \fi\fi\fi%
2862 }
```

\ifSB@gettabind Lyrics under tablature diagrams look odd if they aren't aligned with the leftmost
\SB@tabindent 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.

```
2863 \newif\ifSB@gettabind\SB@gettabindfalse
2864 \SB@newdimen\SB@tabindent
```

\SB@targfret Reserve some macro names in which to store the three pieces of the second argument
\SB@targstr to the \gtab macro. The first is for the fret number, the second is for the ⟨strings⟩
\SB@targfing info, and the last is for the ⟨fingering⟩ info.

```
2865 \newcommand\SB@targfret{}
2866 \newcommand\SB@targstr{}
2867 \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@gtinit Different meanings are assigned to digits, X's, and O's as we typeset each row of the
\SB@gtinc interior of the diagram. These meanings are set by \SB@gtinit and \SB@gtinc.

```
2868 \newcommand\SB@gtinit{%
2869   \def\SB@do##1{\csname##1\endcsname}%
2870   \let\O\0%
2871   \let\3\2\let\4\2\let\5\2\let\6\2%
2872   \let\7\2\let\8\2\let\9\2%
2873 }
2874 \newcommand\SB@gtinc{%
2875   \advance\SB@cnt\@ne%
2876   \let\9\8\let\8\7\let\7\6\let\6\5\let\5\4%
2877   \let\4\3\let\3\2\let\2\1\let\1\SB@lowfret%
2878 }
```

\BarreDelims Each pair of barre delimiters reserves a stack and augments the initialization state
\SB@bdelims to recognize those delimiters.

```
2879 \newcommand\BarreDelims[2]{%
2880   \expandafter\SB@bdelims\csname SB@bs@#1#2\expandafter\endcsname%
2881     \csname#1\expandafter\endcsname\csname#2\endcsname%
2882 }
2883 \newcommand\SB@bdelims[3]{%
2884   \newcommand*#1{}%
2885   \SB@app\def\SB@allbarres{\SB@dobarre#1#2#3}%
2886 }
2887 \BarreDelims()
2888 \BarreDelims[]
```

\gtab A \gtab macro begins by setting catcodes suitable for parsing a chord name as
\SB@gtab its first argument. This allows tokens like # and & to be used for sharp and flat
even when \gtab is used outside a chord macro. Colon is reset to a non-active
character while processing the second argument to avoid a potential conflict with
Babel French.

```
2889 \newcommand\gtab{\SB@begincname\SB@gtab}
2890 \newcommand*\SB@gtab[1]{%
2891   \SB@endcname%
2892   \begingroup%
```

```
2893        \catcode`:12\relax%
2894        \SB@@gtab{#1}%
2895 }
```

\SB@@gtab If transposition is currently taking place, allow the user to customize the behavior by redefining \gtabtrans. Using \gtab within \gtabtrans should go directly to \SB@@@gtab (otherwise an infinite loop would result!).

```
2896 \newcommand*\SB@@gtab[2]{%
2897    \endgroup%
2898    \ifnum\SB@transposefactor=\z@%
2899      \SB@@@gtab{#1}{#2}%
2900    \else%
2901      \begingroup%
2902        \let\gtab\SB@@@gtab%
2903        \gtabtrans{#1}{#2}%
2904      \endgroup%
2905    \fi%
2906 }
```

\gtabtrans By default, transposed guitar tablatures just display the transposed chord name and omit the diagram. Transposing a tablature diagram requires manual judgment calls for most stringed instruments, so we can't do it automatically.

```
2907 \newcommand\gtabtrans[2]{\transposehere{#1}}
```

\SB@@@gtab Typeset a full tablature diagram. Text ⟨arg1⟩ is a chord name placed above the diagram. Text ⟨arg2⟩ consists of a colon-separated list 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 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).

```
2908 \newcommand\SB@@@gtab[2]{%
2909    \let\SB@targfret\@empty%
2910    \let\SB@targstr\@empty%
2911    \let\SB@targfing\@empty%
2912    \SB@tabargs#2:::\SB@tabargs%
2913    \ifx\SB@targstr\@empty%
2914      \def\SB@targstr{\0\0\0\0\0\0}%
2915    \fi%
2916    \ifvmode\leavevmode\fi%
2917    \vbox{%
2918      \normalfont\normalsize%
2919      \setbox\SB@box\hbox{%
2920        \thinspace{\printchord{\transposehere{#1}\strut}}\thinspace%
2921      }%
2922      \setbox\SB@boxii\hbox{\SB@fretnum{\SB@targfret}}%
2923      \setbox\SB@boxiii\hbox{{%
2924        \let\X\SB@topX\let\0\SB@topO%
2925        \let\1\SB@topempty\let\2\1%
2926        \SB@gtinit%
2927        \let\SB@dobarre\SB@barresoff\SB@allbarres%
2928        \SB@targstr%
```

```
2929    }}%
2930    \hsize\wd\SB@box%
2931    \ifSB@gettabind%
2932      \global\SB@tabindent\wd\SB@boxii%
2933      \global\advance\SB@tabindent.5\SB@fretwidth%
2934      \global\advance\SB@tabindent-.5\p@%
2935    \fi%
2936    \SB@dimen\wd\SB@boxii%
2937    \advance\SB@dimen\wd\SB@boxiii%
2938    \ifdim\hsize<\SB@dimen%
2939      \hsize\SB@dimen%
2940    \else\ifSB@gettabind%
2941      \SB@dimenii\hsize%
2942      \advance\SB@dimenii-\SB@dimen%
2943      \divide\SB@dimenii\tw@%
2944      \global\advance\SB@tabindent\SB@dimenii%
2945    \fi\fi%
2946    \hbox to\hsize{\hfil\unhbox\SB@box\hfil}%
2947    \kern-\p@\nointerlineskip%
2948    \hbox to\hsize{%
2949      \hfil%
2950      \vtop{\kern\p@\kern2\p@\box\SB@boxii}%
2951      \vtop{%
2952        \SB@dimen\wd\SB@boxiii%
2953        \box\SB@boxiii%
2954        \let\X\SB@fretempty\let\0\X%
2955        \let\1\SB@fretdot\def\2{\SB@fretempty\global\SB@testtrue}%
2956        \SB@gtinit%
2957        \let\SB@dobarre\SB@barreson\SB@allbarres%
2958        \SB@cnt\@ne%
2959        \loop%
2960          \SB@testfalse%
2961          \SB@fretbar\hbox{\SB@cntii\z@\SB@targstr\SB@fretend}%
2962          \ifnum\SB@cnt<\minfrets\SB@testtrue\fi%
2963        \ifSB@test\SB@gtinc\repeat%
2964        \SB@fretbar%
2965        \ifx\SB@targsfing\@empty\else%
2966          \kern1.5\p@%
2967          \hbox{\let\SB@do\SB@finger\SB@targfing}%
2968        \fi%
2969      }%
2970      \hfil%
2971    }%
2972    \kern3\p@%
2973  }%
2974  \SB@gettabindfalse%
2975 }
```

\SB@tabargs   Break the second argument to a \gtab macro into three sub-arguments. The
\SB@@tabargs possible forms are: (a) ⟨strings⟩, (b) ⟨fret⟩:⟨strings⟩, (c) ⟨strings⟩:⟨fingering⟩, or
\SB@ctoken   (d) ⟨fret⟩:⟨strings⟩:⟨fingering⟩. To distinguish forms (b) and (c), we count the
number of tokens before the first colon. If there is only one token or group, we
assume it must be form (b), since frets larger than 9 and 1-stringed instruments

132

are both rare. Otherwise we assume form (c).

```
2976 \newcommand\SB@ctoken{} \def\SB@ctoken{:}
2977 \newcommand\SB@tabargs{}
2978 \def\SB@tabargs#1:#2:#3:#4\SB@tabargs{%
2979   \def\SB@temp{#4}%
2980   \ifx\SB@temp\@empty%
2981     \SB@doify\SB@targstr{#1}%
2982   \else\ifx\SB@temp\SB@ctoken%
2983     \SB@@tabargs#1\SB@@tabargs%
2984     \ifx\SB@temp\@empty%
2985       \def\SB@targfret{#1}%
2986       \SB@doify\SB@targstr{#2}%
2987     \else%
2988       \SB@doify\SB@targfing{#2}%
2989       \SB@doify\SB@targstr{#1}%
2990     \fi%
2991   \else%
2992     \def\SB@targfret{#1}%
2993     \SB@doify\SB@targfing{#3}%
2994     \SB@doify\SB@targstr{#2}%
2995   \fi\fi%
2996 }
2997 \newcommand\SB@@tabargs{}
2998 \def\SB@@tabargs#1#2\SB@@tabargs{\def\SB@temp{#2}}
```

15.14   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.

```
2999 \newcommand\songchapter{%
3000   \let\SB@temp\@seccntformat%
3001   \def\@seccntformat##1{}%
3002   \@startsection{chapter}{0}{\z@}%
3003     {3.5ex\@plus1ex\@minus.2ex}%
3004     {.4ex\let\@seccntformat\SB@temp}%
3005     {\sffamily\bfseries\LARGE\centering}%
3006 }
```

\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.

```
3007 \newcommand\songsection{%
3008   \let\SB@temp\@seccntformat%
3009   \def\@seccntformat##1{}%
3010   \@startsection{section}{1}{\z@}%
3011     {3.5ex\@plus1ex\@minus.2ex}%
3012     {.4ex\let\@seccntformat\SB@temp}%
3013     {\sffamily\bfseries\LARGE\centering}%
3014 }
```

133

songs Begin and end a book section. The argument is a list of indexes with which to associate songs in this section.

```
3015 \newenvironment{songs}[1]{%
3016   \ifSB@songsenv\SB@errnse\fi%
3017   \gdef\SB@indexlist{#1}%
3018   \SB@chkidxlst%
3019   \stepcounter{SB@songsnum}%
3020   \setcounter{songnum}{1}%
3021   \let\SB@sgroup\@empty%
3022   \ifinner\else\ifdim\pagetotal>\z@%
3023     \null\nointerlineskip%
3024   \fi\fi%
3025   \songcolumns\SB@numcols%
3026   \SB@songsenvtrue%
3027 }{%
3028   \commitsongs%
3029   \global\let\SB@indexlist\@empty%
3030   \ifinner\else\clearpage\fi%
3031   \SB@songsenvfalse%
3032 }
```

Each **songs** section needs a unique number to aid in hyperlinking.

```
3033 \newcounter{SB@songsnum}
```

15.15   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.

Internally, this package code uses a *four* step process to move the index data from the source `.tex` file to the `.sxd` data files.

1. While the current song box is in the midst of construction, the data is stored in a box of non-immediate write whatsit nodes.

2. The whatsits are migrated out to the top of the song box when it is finalized at `\endsong`.

3. When the song box is shipped out to the output file, TeX expands the whatsits, causing the data to be written to the `.sxc` auxiliary file.

4. At the `\end{document}` line, the `.sxc` is processed multiple times—once for each index—to split the data into the respective `.sxd` files.

The first and second steps allow index references to point to the beginning of the song no matter where the indexing commands appear within the song. The third step allows TeX to drop index entries that refer to songs that do not actually appear in the output (e.g., because of `\includeonlysongs`). It also allows index entries to refer to information that is only decided at shipout time, such as page numbers. The fourth step allows all indexing to be accomplished with at most one write register. LaTeX provides extremely few write registers, so using as few as possible is essential for supporting books with many indexes.

`\songtarget` This macro is invoked by each `\beginsong` environment with two arguments: (1) a suggested pdf bookmark index level, and (2) a target name to which hyperlinks for this song in the index will refer. The macro is expected to produce a suitable pdf bookmark entry and/or link target. The default definition tries to use `\pdfbookmark` if generating a PDF, and resorts to `\hypertarget` (if it exists) otherwise. The user can redefine the macro to customize how and whether bookmarks and/or links are created.

```
3034 \newcommand\songtarget[2]{%
3035   \ifnum\@ne=0\ifSB@pdf\ifx\pdfbookmark\undefined\else%
3036                      \ifx\pdfbookmark\relax\else1\fi\fi\fi\relax%
3037     \pdfbookmark[#1]{\thesongnum. \songtitle}{#2}%
3038   \else\ifx\hypertarget\undefined%
3039   \else\ifx\hypertarget\relax\else%
3040     \hypertarget{#2}{\relax}%
3041   \fi\fi\fi%
3042 }
```

`\songlink` This macro is invoked by the index code to produce a link to a song target created by `\songtarget`. Its two arguments are: (1) the target name (same as the second argument to `\songtarget`, and (2) the text that is to be linked. The default implementation uses `\hyperlink` if it exists; otherwise it just leaves the text unlinked.

```
3043 \newcommand\songlink{%
3044   \ifnum\@ne=0\ifx\hyperlink\undefined\else%
3045              \ifx\hyperlink\relax\else1\fi\fi\relax%
3046     \expandafter\hyperlink%
3047   \else%
3048     \expandafter\@gobble%
3049   \fi%
3050 }
```

`\SB@indexlist` This macro records the comma-separated list of the identifiers of indexes associated with the current book section.

```
3051 \newcommand\SB@indexlist{}
```

135

`\SB@allindexes` This macro records a comma-separated list of all the index identifiers for the entire document.

```
3052 \newcommand\SB@allindexes{}
3053 \let\SB@allindexes\@empty
```

`\SB@out` The `\SB@out` control sequence is reserved for the write register allocated by the package code, if one is needed. (It is allocated at the first index declaration.)

```
3054 \newcommand\SB@out{}
3055 \let\SB@out\relax
```

`\SB@newindex` Initialize a new title, author, or scripture index.

```
3056 \newcommand\SB@newindex[4]{%
3057   \expandafter\newcommand\csname SB@idxfilename@#3\endcsname{#4}%
3058   \expandafter\newcommand\csname SB@idxsel@#3\endcsname[3]{###1}%
3059   \expandafter\newcommand\csname SB@idxref@#3\endcsname{\thesongnum}%
3060   \xdef\SB@allindexes{%
3061     \ifx\SB@allindexes\@empty\else\SB@allindexes,\fi#3%
3062   }%
3063   \if@filesw%
3064     \ifx\SB@out\relax%
3065       \SB@newwrite\SB@out%
3066       \immediate\openout\SB@out=\jobname.sxc\relax%
3067     \fi%
3068     \immediate\write\SB@out{\noexpand\SB@iwrite{#3}{#2}}%
3069   \fi%
3070 }
```

`\newindex` Define a new title index. The first argument is an identifier for the index (used in constructing index-specific control sequence names). The second argument is a filename root; auxiliary file ⟨*arg2*⟩`.sxd` is where the index data is stored at the end of processing.

```
3071 \newcommand\newindex{\SB@newindex1{TITLE INDEX DATA FILE}}
3072 \@onlypreamble\newindex
```

`\newscripindex` Define a new scripture index. This is exactly like `\newindex` except that scripture references are added to the auxiliary file instead of titles.

```
3073 \newcommand\newscripindex{\SB@newindex2{SCRIPTURE INDEX DATA FILE}}
3074 \@onlypreamble\newscripindex
```

`\newauthorindex` Define a new author index. This is exactly like `\newindex` except that author info is added to the auxiliary file instead of titles.

```
3075 \newcommand\newauthorindex{\SB@newindex3{AUTHOR INDEX DATA FILE}}
3076 \@onlypreamble\newauthorindex
```

`\SB@cwrite` Write index data to a Song indeX Combined (`.sxc`) auxiliary file. The first argument is the identifier for the index to which the data ultimately belongs. The second argument is the data itself. The write is non-immediate so that it is only output if its enclosing song is ultimately shipped to the output file.

```
3077 \newcommand\SB@cwrite[2]{%
3078   \ifx\SB@out\relax\else%
3079     \protected@write\SB@out\SB@keepactive{\protect\SB@iwrite{#1}{#2}}%
3080   \fi%
3081 }
```

136

\SB@keepactive By default, the `inputenc` package expands Unicode characters into macro names when writing them to files. This behavior must be inhibited when writing to the `.sxc` file, since `songidx` needs the original Unicode characters for sorting. To achieve this, we temporarily redefine most active characters so that they expand to an unexpandable string version of themselves.

```
3082 \newcommand\SB@keepactive{}
3083 {\catcode`\~\active
3084 \catcode`\.12
3085 \def\\#1#2{%
3086   \endgroup
3087   \SB@app\gdef\SB@keepactive{\def#1{#2}}%
3088 }
3089 \def\SB@temp#1#2{%
3090   \SB@cnt#1\relax
3091   \loop
3092     \begingroup
3093       \uccode`\~\SB@cnt
3094       \uccode`\.\SB@cnt
3095     \uppercase{\\~.}
3096   \ifnum\SB@cnt<#2\relax
3097     \advance\SB@cnt\@ne
3098   \repeat
3099 }
3100 \SB@temp{1}{8}
3101 \SB@temp{11}{11}
3102 \SB@temp{14}{91}
3103 \SB@temp{93}{255}
3104 }
```

\SB@iwrite The line contributed by `\SB@cwrite` to the `.sxc` file is an `\SB@iwrite` macro that re-outputs the data to an appropriate `.sxd` file.

```
3105 \newcommand\SB@iwrite[2]{%
3106   \def\SB@tempii{#1}%
3107   \ifx\SB@temp\SB@tempii%
3108     \SB@toks{#2}%
3109     \immediate\write\SB@out{\the\SB@toks}%
3110   \fi%
3111 }
```

\SB@uncombine At the end of the document, the `.sxc` file can be processed multiple times to produce all the `.sxd` files without resorting to multiple write registers. Each pass activates the subset of the `\SB@iwrite` commands that apply to one index.

```
3112 \newcommand\SB@uncombine{%
3113   \ifx\SB@out\relax\else%
3114     \immediate\closeout\SB@out%
3115     \ifsongindexes%
3116       \@for\SB@temp:=\SB@allindexes\do{%
3117         \immediate\openout\SB@out=%
3118           \csname SB@idxfilename@\SB@temp\endcsname.sxd\relax%
3119         \begingroup\makeatletter\catcode`\%12\relax%
3120                    \input{\jobname.sxc}\endgroup%
3121         \immediate\closeout\SB@out%
3122       }%
```

```
3123        \fi%
3124      \fi%
3125 }
3126 \AtEndDocument{\SB@uncombine}
```

\SB@songwrites   The following box register stores index data until it can be migrated to the top of
the song box currently under construction.

```
3127 \SB@newbox\SB@songwrites
```

\SB@addtoindex   Queue data ⟨*arg2*⟩ associated with the current song for eventual writing to the
index whose identifier is given by ⟨*arg1*⟩.

```
3128 \newcommand\SB@addtoindex[2]{%
3129   \protected@edef\SB@tempii{#2}%
3130   \ifx\SB@tempii\@empty\else%
3131     \global\setbox\SB@songwrites\vbox{%
3132       \unvbox\SB@songwrites%
3133       \SB@cwrite{#1}{#2}%
3134       \SB@cwrite{#1}{\csname SB@idxref@#1\endcsname}%
3135       \SB@cwrite{#1}{song\theSB@songsnum-\thesongnum.%
3136                      \ifnum\c@section=\z@1\else2\fi}%
3137     }%
3138   \fi%
3139 }
```

\SB@addtoindexes   Add ⟨*arg1*⟩ to all title indexes, ⟨*arg2*⟩ to all scripture indexes, and ⟨*arg3*⟩ to all
author indexes.

```
3140 \newcommand\SB@addtoindexes[3]{%
3141   \@for\SB@temp:=\SB@indexlist\do{%
3142     \SB@addtoindex\SB@temp%
3143       {\csname SB@idxsel@\SB@temp\endcsname{#1}{#2}{#3}}%
3144   }%
3145 }
```

\SB@addtotitles   Add ⟨*arg1*⟩ to all title indexes, but leave other indexes unaffected.

```
3146 \newcommand\SB@addtotitles[1]{%
3147   \@for\SB@temp:=\SB@indexlist\do{%
3148     \csname SB@idxsel@\SB@temp\endcsname%
3149       {\SB@addtoindex\SB@temp{#1}}{}{}%
3150   }%
3151 }
```

\SB@chkidxlst   Check the current list of indexes and flag an error if any are undefined.

```
3152 \newcommand\SB@chkidxlst{%
3153   \let\SB@temp\SB@indexlist%
3154   \let\SB@indexlist\@empty%
3155   \@for\SB@tempii:=\SB@temp\do{%
3156     \@ifundefined{SB@idxsel@\SB@tempii}{\SB@errnoidx\SB@tempii}{%
3157       \ifx\SB@indexlist\@empty%
3158         \SB@toks\expandafter{\SB@tempii}%
3159       \else%
3160         \SB@toks\expandafter\expandafter\expandafter{%
3161           \expandafter\SB@indexlist\expandafter,\SB@tempii}%
3162       \fi%
```

```
3163        \edef\SB@indexlist{\the\SB@toks}%
3164      }%
3165    }%
3166 }
```

\indexentry \SB@addtoindexes will be called automatically for each song in a section. However,
\SB@idxentry \indexentry may be called by the user in order to add an alternative index entry
\SB@@idxentry 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.

```
3167 \newcommand\indexentry{\@ifnextchar[{\SB@idxentry*}{\SB@@idxentry*}}
3168 \newcommand\SB@idxentry{}
3169 \def\SB@idxentry#1[#2]#3{{%
3170    \def\SB@indexlist{#2}%
3171    \SB@chkidxlst%
3172    \SB@addtoindexes{#1#3}{#3}{#3}%
3173 }}
3174 \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.)

```
3175 \newcommand\indextitleentry{%
3176    \@ifnextchar[{\SB@idxentry{}}{\SB@@idxentry{}}%
3177 }
```

\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.

```
3178 \newcommand\indexsongsas[1]{%
3179    \@ifundefined{SB@idxref@#1}%
3180      {\SB@errnoidx{#1}\@gobble}%
3181      {\expandafter\renewcommand\csname SB@idxref@#1\endcsname}%
3182 }
```

\SB@idxcmd The songidx index-generation script understands several different directives that
\SB@@idxcmd each dictate various aspects of how index entries are parsed, sorted, and displayed.
\authsepword Such directives should typically appear at the start of the .sxd file just after the
\authbyword header line that identifies the type of index.
\authignoreword
\titleprefixword
```
3183 \newcommand\SB@idxcmd[3]{%
3184    \ifx\SB@allindexes\@empty%
3185      \SB@warnnoidx%
3186    \else\ifx\SB@out\relax\else%
3187      \@for\SB@temp:=\SB@allindexes\do{%
3188        \csname SB@idxsel@\SB@temp\endcsname%
3189          {\SB@@idxcmd{#1}}{\SB@@idxcmd{#2}}{\SB@@idxcmd{#3}}%
3190      }%
3191    \fi\fi%
3192 }
3193 \newcommand\SB@@idxcmd[1]{%
3194    \def\SB@tempii{#1}%
3195    \ifx\SB@tempii\@empty\else%
3196      \immediate\write\SB@out{%
3197        \noexpand\SB@iwrite{\SB@temp}{#1}%
3198      }%
```

```
3199    \fi%
3200 }
3201 \newcommand\authsepword[1]{}
3202 \newcommand\authbyword[1]{}
3203 \newcommand\authignoreword[1]{}
3204 \newcommand\titleprefixword[1]{}
3205 {\catcode`\%=12
3206  \gdef\authsepword#1{\SB@idxcmd{}{}{%sep #1}}
3207  \gdef\authbyword#1{\SB@idxcmd{}{}{%after #1}}
3208  \gdef\authignoreword#1{\SB@idxcmd{}{}{%ignore #1}}
3209  \gdef\titleprefixword#1{\SB@idxcmd{%prefix #1}{}{}}}
3210 \@onlypreamble\authsepword
3211 \@onlypreamble\authbyword
3212 \@onlypreamble\authignoreword
3213 \@onlypreamble\titleprefixword
```

\SB@idxlineskip Set the spacing between lines in an index.

```
3214 \newcommand\SB@idxlineskip[1]{%
3215    \vskip#1\p@\@plus#1\p@\@minus#1\p@%
3216 }
```

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, its 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.

```
3217 \newcommand\SB@ellipspread[2]{%
3218    \begingroup%
3219      \SB@dimen\z@%
3220      \def\SB@temp{#1}%
3221      \SB@toks{#2}%
3222      \setbox\SB@box\hbox{{%
3223        \SB@temp%
3224        \leaders\hbox to.5em{\hss.\hss}\hskip2em\@plus1fil%
3225        {\the\SB@toks}%
3226      }}%
3227      \ifdim\wd\SB@box>\hsize%
3228        \SB@balancerows%
3229      \else%
3230        \hbox to\hsize{\unhbox\SB@box}\par%
3231      \fi%
```

```
3232    \endgroup%
3233 }
```

\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.

```
3234 \newcommand\SB@balancerows{%
3235   \edef\SB@tempii{\the\SB@toks}%
3236   \setbox\SB@box\vbox{%
3237     \SB@toks\expandafter{\expandafter\\\the\SB@toks\\}%
3238     \SB@lop\SB@toks\SB@toks%
3239     \settowidth\SB@dimen{\the\SB@toks}%
3240     \advance\SB@dimen-.5em%
3241     \leftskip.5cm%
3242     {\hbadness\@M\hfuzz\maxdimen%
3243      \hskip-.5cm\relax\SB@temp\unskip\nobreak%
3244      \hskip\SB@dimen\nobreak%
3245      \rightskip2em\@plus1fil\par}%
3246     \setbox\SB@box\lastbox%
3247     \setbox\SB@box\hbox{%
3248       \unhbox\SB@box%
3249       \unskip\unskip\unpenalty%
3250       \unpenalty\unskip\unpenalty%
3251     }%
3252     \expandafter%
3253   }%
3254   \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.

```
3255   \SB@dimen\hsize%
3256   \advance\SB@dimen-.5cm%
3257   \setbox\SB@box\vbox{%
3258     \SB@multiline{\hbadness\@M\hfuzz\maxdimen}%
3259   }%
3260   \SB@dimeniii.5\SB@dimen%
3261   \SB@dimeniv\SB@dimeniii%
3262   \loop%
3263     \SB@dimeniv.5\SB@dimeniv%
3264     \setbox\SB@boxii\vbox{%
3265       \SB@dimen\SB@dimeniii%
```

```
3266        \SB@multiline{\hbadness\@M\hfuzz\maxdimen}%
3267      }%
3268      \ifnum\SB@cnt<\@M%
3269        \ifdim\ht\SB@boxii>\ht\SB@box%
3270          \advance\SB@dimeniii\SB@dimeniv%
3271        \else%
3272          \SB@dimen\SB@dimeniii%
3273          \advance\SB@dimeniii-\SB@dimeniv%
3274        \fi%
3275      \else%
3276        \advance\SB@dimeniii\SB@dimeniv%
3277      \fi%
3278    \ifdim\SB@dimeniv>2\p@\repeat%
3279    \setbox\SB@box\box\voidb@x%
3280    \setbox\SB@boxii\box\voidb@x%
```

Finally, typeset the results based on the quantities computed above.

```
3281    \SB@multiline\relax%
3282 }
```

\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!

```
3283 \newcommand\SB@multiline[1]{%
3284   \begingroup%
3285     \SB@dimen-\SB@dimen%
3286     \advance\SB@dimen\hsize%
3287     \SB@dimenii-\SB@dimenii%
3288     \advance\SB@dimenii\SB@dimen%
3289     {#1\hskip-.5cm\relax\SB@temp\unskip\nobreak%
3290      \SB@maxmin\SB@dimenii<{1.5em}%
3291      \leftskip.5cm\rightskip2em\@plus1fil%
3292      \interlinepenalty\@M%
3293      \leaders\hbox to.5em{\hss.\hss}\hskip\SB@dimenii\@plus1fill%
3294      \nobreak{\SB@tempii\kern-2em}%
3295      \par\global\SB@cnt\badness}%
3296   \endgroup%
3297 }%
```

\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.

```
3298 \newcommand\SB@idxitemsep{%
3299   ,\kern-2em\penalty-8\hskip2.33em\@minus.11em%
3300   \hskip-\SB@dimen\@plus-1fill%
```

```
3301    \vadjust{}\nobreak%
3302    \hskip\SB@dimen\@plus1fill\relax%
3303 }
```

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).

```
3304 \newenvironment{idxblock}[1]{}{}
```

\idxentry   Within each `idxblock` environment there should be a series of `\idxentry` and/or
\idxaltentry   `\idxaltentry` macros, one for each line of the index. Again, the exact definitions
of these macros will vary between index types.

```
3305 \newcommand\idxentry[2]{}
3306 \newcommand\idxaltentry[2]{}
```

SB@lgidx   Some indexes actually have two definitions for each `idxblock` environment—one
SB@smidx   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.

```
3307 \newenvironment{SB@lgidx}[1]{}{}
3308 \newenvironment{SB@smidx}[1]{}{}
```

\SB@idxsetup   Set various parameters for a column of an index environment.

```
3309 \newcommand\SB@idxsetup{%
3310    \hsize\SB@colwidth%
3311    \parskip\z@skip\parfillskip\z@skip\parindent\z@%
3312    \baselineskip\f@size\p@\@plus\p@\@minus\p@%
3313    \lineskiplimit\z@\lineskip\p@\@plus\p@\@minus\p@%
3314    \hyphenpenalty\@M\exhyphenpenalty\@M%
3315 }
```

\SB@makeidxcolumn   Break off enough material from `\SB@box` to create one column of the index.

```
3316 \newcommand\SB@makeidxcolumn{%
3317    \ifdim\ht\SB@box=\z@%
3318       \hskip\hsize\relax%
3319    \else%
3320       \splittopskip\z@skip\splitmaxdepth\maxdepth%
3321       \vsplit\SB@box to\SB@dimen%
3322       \global\setbox\SB@box\vbox{%
3323          \SB@idxsetup%
3324          \splitbotmark%
3325          \unvbox\SB@box%
3326       }%
3327    \fi%
3328 }
```

143

\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.

```
3329 \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.

```
3330 \newcommand\SB@displayindex[3]{%
3331   \ifsongindexes\begingroup%
3332     \SB@colwidth\hsize%
3333     \advance\SB@colwidth-#1\columnsep%
3334     \advance\SB@colwidth\columnsep%
3335     \divide\SB@colwidth#1%
3336     \setbox\SB@envbox\vbox{%
3337       \let\SB@temp\songsection%
3338       \ifx\chapter\undefined\else%
3339         \ifx\chapter\relax\else%
3340           \let\SB@temp\songchapter%
3341         \fi%
3342       \fi%
3343       \SB@temp{#2}%
3344     }%
```

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).

```
3345     \IfFileExists{\csname SB@idxfilename@#3\endcsname.sbx}{%
3346       \ifsepindexes%
3347         \global\setbox\SB@box\vbox{%
3348           \null%
3349           \vfil%
3350           \unvcopy\SB@envbox%
3351           \vskip.5in\@minus.3in\relax%
3352           \hbox to\hsize{%
3353             \hfil%
3354             \vbox{%
3355               \SB@idxsetup%
3356               \renewenvironment{idxblock}[1]%
3357                 {\begin{SB@smidx}{####1}}{\end{SB@smidx}}%
3358               \let\\\SB@idxitemsep%
3359               \input{\csname SB@idxfilename@#3\endcsname.sbx}%
3360             }%
3361             \hfil%
3362           }%
3363           \vskip\z@\@plus2fil\relax%
3364         }%
```

Test whether the resulting small index fits within one page. If not, re-typeset it as a large index.

```
3365         {\vbadness\@M\vfuzz\maxdimen%
3366          \splitmaxdepth\maxdepth\splittopskip\z@skip%
3367          \global\setbox\SB@boxii\vsplit\SB@box to\textheight}%
3368         \ifvoid\SB@box%
3369           \box\SB@boxii%
```

144

```
3370        \else%
3371          \SB@lgindex{#1}{#3}%
3372        \fi%
3373      \else%
3374        \SB@lgindex{#1}{#3}%
3375      \fi%
3376    }%
```

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.

```
3377      {%
3378        \ifsepindexes%
3379          \vbox to\textheight{%
3380            \vfil%
3381            \unvbox\SB@envbox%
3382            \vskip1em\relax%
3383            \hbox to\hsize{\hfil[Index not yet generated.]\hfil}%
3384            \vskip\z@\@plus2fil\relax%
3385          }%
3386        \else%
3387          \unvbox\SB@envbox%
3388          \hbox to\hsize{\hfil[Index not yet generated.]\hfil}%
3389        \fi%
3390      }%
3391      \ifsepindexes\clearpage\fi%
3392    \endgroup\fi%
3393 }
```

\SB@lgindex Typeset a large-style index. We begin by typesetting the entire index into a box.

```
3394 \newcommand\SB@lgindex[2]{%
3395   \global\setbox\SB@box\vbox{%
3396     \renewenvironment{idxblock}[1]%
3397       {\begin{SB@lgidx}{##1}}{\end{SB@lgidx}}%
3398     \let\\\SB@idxitemsep%
3399     \SB@idxsetup%
3400     \input{\csname SB@idxfilename@#2\endcsname.sbx}%
3401     \unskip%
3402   }%
```

Next, we split the box into columns and pages until the last page is reached.

```
3403   \SB@toks{\SB@makeidxcolumn}%
3404   \SB@cnt#1\relax%
3405   \loop\ifnum\SB@cnt>\@ne%
3406     \SB@toks\expandafter{\the\SB@toks%
3407       \kern\columnsep\SB@makeidxcolumn}%
3408     \advance\SB@cnt\m@ne%
3409   \repeat%
3410   \edef\SB@oneidxpage{\the\SB@toks}%
3411   \unvbox\SB@envbox%
3412   \vskip.2in\relax%
3413   \nointerlineskip%
3414   \null%
3415   \nointerlineskip%
3416   \SB@cnt\vbadness\vbadness\@M%
```

```
3417    \SB@dimenii\vfuzz\vfuzz\maxdimen%
3418    \loop%
3419      \SB@dimen\textheight%
3420      \ifinner\else\kern\z@\advance\SB@dimen-\pagetotal\fi%
3421      \global\setbox\SB@boxii\copy\SB@box%
3422      \global\setbox\SB@boxiii\hbox{\SB@oneidxpage}%
3423      \ifdim\ht\SB@box>\z@%
3424        \box\SB@boxiii%
3425        \vfil\break%
3426    \repeat%
```

The final page of the index should have all equal-height columns instead of a few full columns followed by some short or empty columns at the end. To achieve this, we re-typeset the final page, trying different column heights until we find one that causes the material to span an equal percentage of all the columns on the page.

```
3427    \SB@dimenii\ht\SB@boxii%
3428    \divide\SB@dimenii#1\relax%
3429    \SB@maxmin\SB@dimen>\SB@dimenii%
3430    \loop%
3431      \global\setbox\SB@box\copy\SB@boxii%
3432      \global\setbox\SB@boxiii\hbox{\SB@oneidxpage}%
3433      \ifdim\ht\SB@box>\z@%
3434        \advance\SB@dimen\p@%
3435    \repeat%
3436    \box\SB@boxiii%
3437    \global\setbox\SB@boxii\box\voidb@x%
3438    \vbadness\SB@cnt\vfuzz\SB@dimenii%
3439 }
```

\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.

```
3440 \newcommand\showindex[3][0]{%
3441    \@ifundefined{SB@idxsel@#3}{\SB@errnoidx{#3}}{%
3442      \expandafter\let\expandafter\SB@temp\csname SB@idxsel@#3\endcsname%
3443      \SB@cnt#1\relax%
3444      \ifnum\SB@cnt<\@ne\SB@cnt\SB@temp232\relax\fi%
3445      \expandafter\SB@temp%
3446      \expandafter\SB@maketitleindex%
3447      \expandafter\SB@makescripindex%
3448      \expandafter\SB@makeauthorindex%
3449      \expandafter{\the\SB@cnt}%
3450      {#2}{#3}%
3451    }%
3452 }
```

\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).

```
3453 \newcommand\SB@maketitleindex{%
3454    \ifnum\idxheadwidth>\z@%
3455      \renewenvironment{SB@lgidx}[1]{
3456        \hbox{\SB@colorbox\idxbgcolor{\vbox{%
3457          \hbox to\idxheadwidth{{\idxheadfont\relax##1}\hfil}%
```

146

```
3458        }}}%
3459        \nobreak\vskip3\p@\@plus2\p@\@minus2\p@\nointerlineskip%
3460      }{\penalty-50\vskip5\p@\@plus5\p@\@minus4\p@}%
3461    \else%
3462      \renewenvironment{SB@lgidx}[1]{}{}%
3463    \fi%
3464    \renewenvironment{SB@smidx}[1]{}{}%
3465    \renewcommand\idxentry[2]{%
3466      \SB@ellipspread{\idxtitlefont\relax\ignorespaces##1\unskip}%
3467                    {{\idxrefsfont\relax##2}}%
3468    }%
3469    \renewcommand\idxaltentry[2]{%
3470      \SB@ellipspread{\idxlyricfont\relax\ignorespaces##1\unskip}%
3471                    {{\idxrefsfont\relax##2}}%
3472    }%
3473    \SB@displayindex%
3474 }
```

\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)".

```
3475 \newcommand\SB@idxcolhead{}
```

\SB@idxheadsep Add vertical space following the header line that begins (or continues) a section of
a scripture index.

```
3476 \newcommand\SB@idxheadsep{{%
3477    \SB@dimen4\p@%
3478    \advance\SB@dimen-\prevdepth%
3479    \SB@maxmin\SB@dimen<\z@%
3480    \SB@dimenii\SB@dimen%
3481    \SB@maxmin\SB@dimenii>\p@%
3482    \vskip\SB@dimen\@plus\p@\@minus\SB@dimenii%
3483 }}
```

\SB@idxcont Typeset the "Bookname (continued)" line that continues a scripture index section
when it spans a column break.

```
3484 \newcommand\SB@idxcont[1]{%
3485    \hbox to\hsize{{\idxcont{#1}}\hfil}%
3486    \nobreak%
3487    \SB@idxheadsep\nointerlineskip%
3488 }
```

\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).

```
3489 \newcommand\SB@makescripindex{%
3490    \renewenvironment{SB@lgidx}[1]{%
3491      \gdef\SB@idxcolhead{##1}%
3492      \hbox to\hsize{{\idxbook{##1}}\hfil}%
3493      \nobreak%
3494      \SB@idxheadsep\nointerlineskip%
3495    }{%
3496      \mark{\noexpand\relax}%
3497      \penalty-20\vskip3\p@\@plus3\p@\relax%
3498    }%
```

147

```
3499    \renewenvironment{SB@smidx}[1]
3500      {\begin{SB@lgidx}{##1}}{\end{SB@lgidx}}%
3501    \renewcommand\idxentry[2]{%
3502      \SB@ellipspread{\hskip.25cm\idxscripfont\relax##1}%
3503                     {{\idxrefsfont\relax##2}}%
3504      \SB@toks\expandafter{\SB@idxcolhead}%
3505      \mark{\noexpand\SB@idxcont{\the\SB@toks}}%
3506    }%
3507    \renewcommand\idxaltentry[2]{\SB@erridx{a scripture}}%
3508    \SB@displayindex%
3509 }
```

\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).

```
3510 \newcommand\SB@makeauthorindex{%
3511    \renewenvironment{SB@lgidx}[1]{}{}%
3512    \renewenvironment{SB@smidx}[1]{}{}%
3513    \renewcommand\idxentry[2]{%
3514      \SB@ellipspread{{\idxauthfont\relax\sfcode`.\@m##1}}%
3515                     {{\idxrefsfont##2}}%
3516    }%
3517    \renewcommand\idxaltentry[2]{\SB@erridx{an author}}%
3518    \SB@displayindex%
3519 }
```

15.16 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   All errors and warnings will be reported as coming from package "songs".
\SB@Warn
```
3520 \newcommand\SB@Error{\PackageError{songs}}
3521 \newcommand\SB@Warn{\PackageWarning{songs}}
```

\SB@errspos
```
3522 \newcommand\SB@errspos{%
3523    \SB@Error{Illegal \protect\songpos\space argument}{The argume%
3524    nt to \protect\songpos\space must be a number from 0 to 3.}%
3525 }
```

\SB@errnse
```
3526 \newcommand\SB@errnse{%
3527    \SB@Error{Nested songs environments are not supported}{End th%
3528    e previous songs environment before beginning the next one.}%
3529 }
```

\SB@errpl
```
3530 \newcommand\SB@errpl{%
3531    \SB@Error{\protect\includeonlysongs\space not permitted with%
3532    in a songs environment}{\protect\includeonlysongs\space can o%
3533    nly be used in the document preamble or between songs environ%
3534    ments in the document body.}%
3535 }
```

\SB@errrtopt

```
3536 \newcommand\SB@errrtopt{%
3537   \SB@Error{Cannot display chords in a rawtext dump}{You have u%
3538   sed the rawtext option in the \protect\usepackage\space lin%
3539   e and have either used the chorded option as well or have use%
3540   d the \protect\chordson\space macro subsequently.}%
3541 }
```

\SB@warnrc

```
3542 \newcommand\SB@warnrc{%
3543   \SB@Warn{The \protect\repchoruses\space feature will not wor%
3544   k when the number of columns is set to zero}%
3545 }
```

\SB@warnnoidx

```
3546 \newcommand\SB@warnnoidx{%
3547   \SB@Warn{Index command has no effect since no indexes are ye%
3548   t declared}%
3549 }
```

\SB@errboo

```
3550 \newcommand\SB@errboo{%
3551   \SB@Error{Encountered \protect\beginsong\space without seein%
3552   g an \protect\endsong\space for the previous song}%
3553   {Song \thesongnum\space might be missing a%
3554   n \protect\endsong\space line.}%
3555 }
```

\SB@errbor

```
3556 \newcommand\SB@errbor{%
3557   \SB@Error{Encountered \protect\beginsong\space without seein%
3558   g an \protect\endscripture\space for the preceding scriptur%
3559   e quotation}{A scripture quotation appearing after son%
3560   g \thesongnum\space might be missing a%
3561   n \protect\endscripture\space line.}%
3562 }
```

\SB@erreov

```
3563 \newcommand\SB@erreov{%
3564   \SB@Error{Encountered \protect\endsong\space without seein%
3565   g an \protect\endverse\space for the preceding verse}{Son%
3566   g \thesongnum\space has a \protect\beginverse\space%
3567   line with no matching \protect\endverse\space line.}%
3568 }
```

\SB@erreoc

```
3569 \newcommand\SB@erreoc{%
3570   \SB@Error{Encountered \protect\endsong\space without seein%
3571   g an \protect\endchorus\space for the preceding chorus}{Son%
3572   g \thesongnum\space has a \protect\beginchorus\space%
3573   line with no matching \protect\endchorus\space line.}%
3574 }
```

\SB@erreor

```
3575 \newcommand\SB@erreor{%
3576   \SB@Error{Encountered \protect\endsong\space without seein%
3577   g an \protect\endscripture for the preceding scripture quot%
3578   e}{A scripture quote appearing before song \thesongnum\space%
3579   ended with \protect\endsong\space instead of wit%
3580   h \protect\endscripture.}%
3581 }
```

\SB@erreot

```
3582 \newcommand\SB@erreot{%
3583   \SB@Error{Encountered \protect\endsong\space with no matchin%
3584   g \protect\beginsong}{Before song \thesongnum\space there wa%
3585   s an \protect\endsong\space with no matchin%
3586   g \protect\beginsong.}%
3587 }
```

\SB@errbvv

```
3588 \newcommand\SB@errbvv{%
3589   \SB@Error{Encountered \protect\beginverse\space without seein%
3590   g an \protect\endverse\space for the preceding verse}{Son%
3591   g \thesongnum\space might have a verse that has n%
3592   o \protect\endendverse\space line.}%
3593 }
```

\SB@errbvc

```
3594 \newcommand\SB@errbvc{%
3595   \SB@Error{Encountered \protect\beginverse\space without seein%
3596   g an \protect\endchorus\space for the preceding chorus}{Son%
3597   g \thesongnum\space might have a chorus that has n%
3598   o \protect\endchorus\space line.}%
3599 }
```

\SB@errbvt

```
3600 \newcommand\SB@errbvt{%
3601   \SB@Error{Encountered \protect\beginverse\space without firs%
3602   t seeing a \protect\beginsong\space line}{Before son%
3603   g \thesongnum, there is a \protect\beginverse\space line no%
3604   t contained in any song.}%
3605 }
```

\SB@errevc

```
3606 \newcommand\SB@errevc{%
3607   \SB@Error{Encountered \protect\endverse\space while process%
3608   ing a chorus}{Song \thesongnum\space might hav%
3609   e a \protect\beginchorus\space concluded by a%
3610   n \protect\endverse\space instead of an \protect\endchorus.}%
3611 }
```

```
3612 \newcommand\SB@errevo{%
3613   \SB@Error{Encountered \protect\endverse\space without firs%
3614   t seeing a \protect\beginverse}{Song \thesongnum\space m%
3615   ight have an \protect\endverse\space with no matchin%
3616   g \protect\beginverse.}%
3617 }
```

```
3618 \newcommand\SB@errevt{%
3619   \SB@Error{Encountered an \protect\endverse\space outside o%
3620   f any song}{Before song \thesongnum, there is a%
3621   n \protect\endverse\space line not preceded b%
3622   y a \protect\beginsong\space line.}%
3623 }
```

```
3624 \newcommand\SB@erretex{%
3625   \SB@Error{The \protect\repchoruses\space feature requires e-%
3626   TeX compatibility}{Your version of LaTeX2e does not appear t%
3627   o be e-TeX compatible. Find a distribution that includes e-T%
3628   eX support in order to use this feature.}%
3629 }
```

```
3630 \newcommand\SB@errbcv{%
3631   \SB@Error{Encountered \protect\beginchorus\space without see%
3632   ing an \protect\endverse\space for the preceding verse}{Son%
3633   g \thesongnum\space might hav%
3634   e a \protect\beginverse\space with no match%
3635   ing \protect\endverse.}%
3636 }
```

```
3637 \newcommand\SB@errbcc{%
3638   \SB@Error{Encountered \protect\beginchorus\space without see%
3639   ing an \protect\endchorus\space for the preceding chorus}%
3640   {Song \thesongnum\space might have a \protect\beginchorus%
3641   \space with no matching \protect\endchorus.}%
3642 }
```

```
3643 \newcommand\SB@errbct{%
3644   \SB@Error{Encountered \protect\beginchorus\space without see%
3645   ing a \protect\beginsong\space line first}{After son%
3646   g \thesongnum\space there is a \protect\beginchorus\space%
3647   line outside of any song.}%
3648 }
```

```
3649 \newcommand\SB@errecv{%
3650   \SB@Error{Encountered an \protect\endchorus\space while proc%
3651   essing a verse}{Song \thesongnum\space might hav%
3652   e a \protect\beginverse\space concluded by \protect\endchorus%
3653   \space instead of \protect\endverse.}%
3654 }
```

```
3655 \newcommand\SB@erreco{%
3656   \SB@Error{Encountered \protect\endchorus\space without firs%
3657   t seeing a \protect\beginchorus}{Song \thesongnum\space m%
3658   ight have an \protect\endchorus\space with no match%
3659   ing \protect\beginchorus.}%
3660 }
```

```
3661 \newcommand\SB@errect{%
3662   \SB@Error{Encountered an \protect\endchorus\space outside o%
3663   f any song}{Before song \thesongnum, there is a%
3664   n \protect\endchorus\space line not preceded b%
3665   y a \protect\beginsong\space line.}%
3666 }
```

```
3667 \newcommand\SB@errbro{%
3668   \SB@Error{Missing \protect\endsong}%
3669   {Nested song and intersong environments are not supported%
3670   . Song \thesongnum\space might be missing a%
3671   n \protect\endsong\space line.}%
3672 }
```

```
3673 \newcommand\SB@errbrr{%
3674   \SB@Error{Nested intersong environments are not supported}%
3675   {A scripture quote or other intersong environment before s%
3676   ong \thesongnum\space is missing its ending line.}%
3677 }
```

```
3678 \newcommand\SB@errero{%
3679   \SB@Error{Encountered an \protect\endscripture\space whil%
3680   e processing a song}{Song \thesongnum\space ends wit%
3681   h \protect\endscripture\space when it should end wit%
3682   h \protect\endsong.}%
3683 }
```

```
3684 \newcommand\SB@errert{%
3685   \SB@Error{Encountered an \protect\endscripture\space with%
3686   out first seeing a \protect\beginscripture}{Before son%
3687   g \thesongnum, there is an \protect\endscripture\space w%
3688   ith no matching \protect\beginscripture.}%
3689 }
```

```
3690 \newcommand\SB@errscrip[1]{%
3691   \SB@Error{Encountered a \protect#1\space outside a scriptu%
3692   re quote}{\protect#1\space can only appear betwee%
3693   n \protect\beginscripture\space an%
3694   d \protect\endscripture\space lines.}%
3695 }
```

```
3696 \newcommand\SB@errchord{%
3697   \SB@Error{Song \thesongnum\space seems to have chord%
3698   s that appear outside of any verse or chorus}{All chords a%
3699   nd lyrics should appear between \protect\beginverse\space%
3700   and \protect\endverse, or between \protect\beginchorus\space%
3701   and \protect\endchorus.}%
3702 }
```

```
3703 \newcommand\SB@errreplay{%
3704   \SB@Error{Replayed chord has no matching chord}{Son%
3705   g \thesongnum\space uses \protect^ more times than the%
3706   re are chords in the previously memorized verse.}%
3707 }
```

```
3708 \newcommand\SB@errreg[1]{%
3709   \SB@Error{Unknown chord-replay register name: #1}{Chord-re%
3710   play registers must be declared with \protect\newchords.}%
3711 }
```

```
3712 \newcommand\SB@errdup[1]{%
3713   \SB@Error{Duplicate definition of chord-replay register%
3714   : #1}{\protect\newchords\space was used to declare the sa%
3715   me chord-replay register twice.}%
3716 }
```

```
3717 \newcommand\SB@errmbar{%
3718   \SB@Error{Song \thesongnum\space seems to have measur%
3719   e bars that appear outside of any verse or chorus}{All mea%
3720   sure bars (produced with \protect\mbar\space or |) must ap%
3721   pear between \protect\beginverse\space an%
3722   d \protect\endverse, or between \protect\beginchorus\space%
3723   and \protect\endchorus.}%
3724 }
```

```
3725 \newcommand\SB@errebar[2]{%
3726   \SB@Error{Ignoring unbalanced \expandafter\@gobble\string#2 i%
3727   n \protect\gtab}{Found no \expandafter\@gobble\string#1 to ma%
3728   tch the \expandafter\@gobble\string#2.}%
3729 }
```

```
3730 \newcommand\SB@errnoidx[1]{%
3731   \SB@Error{Unknown index identifier: #1}{This index identifie%
3732   r was never declared using \protect\newindex.}%
3733 }
```

```
3734 \newcommand\SB@erridx[1]{%
3735   \SB@Error{\protect\idxaltentry\space not allowed in #1 index}%
3736   {This error should not occur. The index generation routines ha%
3737   ve malfunctioned. Try deleting all temporary files and then re%
3738   compiling.}%
3739 }
```

15.17 Option Processing

\ifchorded
\iflyric
\ifslides
\ifmeasures
\ifpartiallist
\ifrepchorus
\iftranscapos
\ifnolyrics
\ifrawtext
\ifsongindexes
\ifsepindexes
\ifpagepreludes
\ifSB@colorboxes
\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.

```
3740 \newif\ifchorded
3741 \newif\iflyric\lyrictrue
3742 \newif\ifslides
3743 \newif\ifmeasures
3744 \newif\ifpartiallist
3745 \newif\ifrepchorus
3746 \newif\iftranscapos
3747 \newif\ifnolyrics
3748 \newif\ifrawtext
3749 \newif\ifsongindexes\songindexestrue
3750 \newif\ifsepindexes\sepindexestrue
3751 \newif\ifpagepreludes
3752 \newif\ifSB@colorboxes
3753 \IfFileExists{color.sty}\SB@colorboxestrue\SB@colorboxesfalse
3754 \newif\ifSB@omitscrip
```

\nolyrics
\pagepreludes
The \nolyrics and \pagepreludes macros are just shorthand for \nolyricstrue and \pagepreludestrue, respectively.

```
3755 \newcommand\nolyrics{}
3756 \let\nolyrics\nolyricstrue
3757 \newcommand\pagepreludes{\pagepreludestrue\songpos0}
```

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.

```
3758 \SB@chordson
3759 \ProcessOptions\relax
```

\SB@colorbox Include the colors package and define colors, if requested.

```
3760 \ifSB@colorboxes
3761   \RequirePackage{color}
3762   \definecolor{SongbookShade}{gray}{.80}
3763   \newcommand\SB@colorbox[2]{%
```

154

```
3764    \ifx\@empty#1%
3765      \vbox{%
3766        \kern3\p@%
3767        \hbox{\kern3\p@{#2}\kern3\p@}%
3768        \kern3\p@%
3769      }%
3770    \else%
3771      \colorbox{#1}{#2}%
3772    \fi%
3773  }
3774 \else
3775  \newcommand\SB@colorbox[2]{\vbox{%
3776    \kern3\p@%
3777    \hbox{\kern3\p@{#2}\kern3\p@}%
3778    \kern3\p@%
3779  }}
3780 \fi
```

15.18   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.

```
3781 \ifrawtext
3782  \SB@newwrite\SB@txtout
3783  \immediate\openout\SB@txtout=\jobname.txt
3784  \newif\ifSB@doEOL
3785  {\catcode`\^^M12 %
3786   \catcode`\^^J12 %
3787   \gdef\SB@printEOL{\ifSB@doEOL^^M^^J\fi}}
3788  {\catcode`#12\gdef\SB@hash{#}}
3789  {\catcode`&12\gdef\SB@amp{&}}
3790  \renewcommand\SB@@@@beginsong{%
3791    \begingroup%
3792      \def\'{}\def\`{}\def\v{}\def\u{}\def\={}\def\^{}%
3793      \def\.{}\def\H{}\def\~{}\def\"{}\def\t{}%
3794      \def\copyright{(c)}%
3795      \let~\space%
3796      \let\par\SB@printEOL%
3797      \let\#\SB@hash%
3798      \let\&\SB@amp%
3799      \catcode`|9 %
3800      \catcode`*9 %
3801      \catcode`^9 %
3802      \def\[##1]{}%
3803      \resettitles%
3804      \immediate\write\SB@txtout{\thesongnum. \songtitle}%
3805      \nexttitle%
3806      \foreachtitle{\immediate\write\SB@txtout{(\songtitle)}}%
3807      \ifx\songauthors\@empty\else%
3808        \immediate\write\SB@txtout{\songauthors}%
3809      \fi%
3810      \ifx\SB@rawrefs\@empty\else%
3811        \immediate\write\SB@txtout{\SB@rawrefs}%
3812      \fi%
```

155

```
3813        \immediate\write\SB@txtout{}%
3814        \SB@doEOLfalse%
3815        \obeylines%
3816  }
3817  \renewcommand\SB@endsong{%
3818        \SB@doEOLtrue%
3819        \immediate\write\SB@txtout{\songcopyright\space%
3820          \songlicense\SB@printEOL}%
3821      \endgroup%
3822      \SB@insongfalse%
3823      \stepcounter{songnum}%
3824  }
3825  \def\SB@parsesrefs#1{\def\songrefs{#1}}
3826  \long\def\beginverse#1#2\endverse{%
3827    \SB@doEOLtrue\begingroup%
3828        \def\textnote##1{##1}%
3829        \def\SB@temp{#1}%
3830        \def\SB@star{*}%
3831        \ifx\SB@temp\SB@star%
3832          \immediate\write\SB@txtout{\@gobble#2}%
3833        \else%
3834          \immediate\write\SB@txtout{#2}%
3835        \fi%
3836      \endgroup\SB@doEOLfalse}
3837  \long\def\beginchorus#1\endchorus{%
3838    \SB@doEOLtrue\begingroup%
3839        \def\textnote##1{##1}%
3840        \immediate\write\SB@txtout{Chorus:#1}%
3841      \endgroup\SB@doEOLfalse}
3842  \long\def\beginscripture#1\endscripture{}
3843  \def\musicnote#1{}
3844  \def\textnote#1{%
3845    \SB@doEOLtrue%
3846    \immediate\write\SB@txtout{#1\SB@printEOL}%
3847    \SB@doEOLfalse}
3848  \def\brk{}
3849  \def\rep#1{(x#1)}
3850  \def\echo#1{(#1)}
3851  \def\mbar#1#2{}
3852  \def\lrep{}
3853  \def\rrep{}
3854  \def\nolyrics{}
3855  \renewcommand\memorize[1][]{}
3856  \renewcommand\replay[1][]{}
3857 \fi
```

156