The songproj package*

Tanguy Ortolo
`tanguy+latex@ortolo.eu`

March 29, 2023

1 Introduction

This package, together with the `beamer` class, is used to generate slideshows with song lyrics. This is typically used in religious services in churches equipped with a projector, for which this package has been written, but it can be useful for any type of singing assembly[1]. It provides environments to describe a song in a natural way, and formatting it into slides with overlays.

2 Usage

2.1 The `song` environment

The main feature of this package is the `song`, that allows the user to describe an entire song that will be formatted into slides.

`song` The `song{`⟨*stanzas per slide*⟩`}[`⟨*couplet list*⟩`]` environment is used around an entire song. It takes a mandatory argument, ⟨*stanzas per slide*⟩, to specify whether the user wants to show one or two stanzas[2] on the slide. An optional argument, ⟨*couplet list*⟩ is a comma-separated list of couplet (or verse) indexes, that allows the user to indicate that they want to include only these couplets of a large song: without this, all couplets will be included.

Inside of the `song` environment, the user will use the `\longest` command and the `intro`, `refrain`, `couplet`[3] and `final` environments.

Warning Inside a `song` environment, it is an error to write anything that is not an `intro`, `refrain`, `couplet`, `final` environment or a `\longest` command. Direct text would be typeset in a way that would disrupt the song formatting.

---

[*] This document corresponds to songproj v1.2.0, dated 2023/03/29.

[1] Indeed, the song used here as an example is not really a religious one! It was chosen because it is in the public domain and the author likes it.

[2] including the refrain

[3] We chose to use the French words *refrain* and *couplet* for several reason: the author is French, these words are understandable in English and their English equivalents, *chorus* and *verse*, have multiple meanings that would make them very ambiguous in both usage and implementation of this package.

`\longest` `\longest{⟨song line⟩}`

Inside a `song` environment, the `\longest{⟨song line⟩}` command is used to declare the longest line of a song, that will be used to properly center the song stanzas, as allowed by the verse package. That line is only used to compute and record its length, and will not be typeset.

`\numbercouplets` Inside a `song` environment, the `\numbercouplets` command can be used to enable couplet numbering. This can be useful when specific couplets have been selected but the lead singer has a score a lyrics sheet that includes all of them: indicating the couplet numbers in the projection will allow them to check which couplet to sing.

`intro` Inside a `song` environment, the optional `intro` environment declares a number of lines meant to be sung once, at the beginning of the song. In a psalm, this may be an antiphon.

`refrain` Inside a `song` environment, the optional `refrain` environment declares the song refrain (or chorus). A song may start with its refrain, or with a first couplet, followed by the refrain. It is not useful to declare the refrain several time, as the `song` environment will take care of repeating between the couplets.

`couplet` Inside a `song` environment, the `couplet` environment declares each couplet (or verse) of the song.

`final` Inside a `song` environment, the optional `final` environment declares a number of lines meant to be sung once, at the end of the song. In an hymn, that may be a doxology.

Example The following song is defined with three couplets and a refrain. Since its begins with a couplet, it will be formatted with the first couplet, the refrain, the second couplet, the refrain, and so on.

The `song` environment is given two arguments, `{2}[1,2]`. The first one tells it to show two stanzas, that is, both a couplet and the refrain, on the generated slide. The second argument tells it to include only the first two couplets in the output.

```
\begin{frame}
  \begin{song}{2}[1,2]
    \longest{Light she was, and like a fairy,}
    \begin{couplet}
      In a cavern, in a canyon, \\
      Excavating for a mine. \\
      Dwelt a miner, forty-niner, \\
      And his daughter, Clementine. \\
    \end{couplet}
    \begin{refrain}
      Oh my darling, oh my darling, \\
      Oh my darling Clementine, \\
      You are lost and gone forever, \\
      Dreadful sorry, Clementine. \\
    \end{refrain}
    \begin{couplet}
      Light she was, and like a fairy, \\
      And her shoes were number nine, \\
      Herring boxes, without topses, \\
      Sandals were for Clementine. \\
    \end{couplet}
    \begin{couplet}
      […]
    \end{couplet}
  \end{song}
\end{frame}
```

2.2 The `\inputsong` command

`\inputsong`
`\inputsong*`

`\inputsong{⟨file⟩}{⟨stanzas per slide⟩}[⟨couplet list⟩]`
`\inputsong*{⟨file⟩}{⟨stanzas per slide⟩}[⟨couplet list⟩]`

The `\inputsong` command environment is used as a shortcut for typesetting a song written in an external file. That file should contain the song content, including intro, refrain, couplet or final as needed, but *without* being wrapped in a `song` environment.

For instance, one could write a file named `clementine.tex` containing the *content* of the `song` environment shown in example page 2, and use it in a slideshow:

```
\frame{\inputsong{clementine.tex}{2}[1,2]}
```

The starred version `\inputsong*` enables couplet numbering, as described in 2.1.

2.3 The `refrain`, `couplet`, `intro` and `final` environments

These commands are also usable outside of a `song` environment. In that case, they simply format a refrain or couplet, which can be useful when you need more manual control.

`refrain` Outside of a `song` environment, this environment simply wraps its content inside a `structure` and a `verse` environment. It takes an optional ⟨*verse width*⟩ argument, that is used to properly center the refrain, as allowed by the verse package.

`couplet` Outside of a `song` environment, this environment simply wraps its content inside a `verse` environment. It takes an optional ⟨*verse width*⟩ argument, that is used to properly center the refrain, as allowed by the verse package.

`intro`
`final` Outside of a `song` environment, these environments simply wrap their content inside a `em` and a `verse` environment. They takes an optional ⟨*verse width*⟩ argument, that is used to properly center the refrain, as allowed by the verse package.

2.4 Usage tips

For regular offices, there are several suggestions that can ease the creation and usage of lyric slideshows.

2.4.1 Using dedicated song files

It is suggested to write song lyrics in dedicated files, each containing a single the *content* of a `song` environment, without the environment wrapping itself. They can then be used with the `\inputsong` command.

For instance, one could write a file named `clementine.tex` containing the *content* of the `song` environment shown in example page 2. It would then be used in a slideshow such as:

```
\documentclass{beamer}
\usepackage{songproj}

\begin{document}
  \begin{frame}
    \inputsong{clementine.tex}{2}[1,2,3]
  \end{frame}
\end{document}
```

2.4.2 Importing text lyrics

Song lyrics are often found in text format with basic markup:

```
1. In a cavern, in a canon,
Excavating for a mine.
Dwelt a miner, forty-niner,
And his daughter, Clementine.

C. Oh my darling, oh my darling,
Oh my darling Clementine
You are lost and gone forever,
Dreadful sorry Clementine.

2. Light she was, and like a fairy,
And her shoes were number nine,
Herring boxes, without topses,
Sandals were for Clementine.

[…]
```

To avoid the tedious task of manually removing text and adding LaTeX markup, we provide the `song2tex.py` helper. Please refer to its embedded help for detailed instructions about its usage:

```
$ ./song2tex.py --help
$ ./song2tex.py clementine.txt clementine.tex
```

2.4.3 Projection layout advice

During a religious service, a song lyrics projection is only a support, and should not draw their attention away from the main feature, which is the common prayer.

I therefore suggest using a very simple Beamer theme, such as its default one with the owl color theme, and removing the navigation symbols. I also suggest not showing song titles (or anything else that is not actually sung by the assembly) unless there is a good reason to do so, such as getting used to a song or set of songs you intend to reuse often.

```
\documentclass{beamer}
\usecolortheme{owl}
\setbeamertemplate{navigation symbols}{}
\usepackage{songproj}
\begin{document}
  […]
\end{document}
```

2.4.4 Projection advice

For projecting song lyrics, you can take advantage of using a PDF presentation software able to show a presenter console on your laptop screen, and the current slide on the projector. Software like as pdfpc or Pympress can also understand and adapt their display to the concept of Beamer overlay.

3 Implementation

3.1 Dependencies

This module is written using LaTeX3 programming interfaces and command definitions:

```
\RequirePackage{expl3}
\RequirePackage{xparse}
```

The implementation of the `song` environment and its friends is mainly based on the verse package:

```
\RequirePackage{verse}
```

3.2 Internal definitions

Almost all definitions use the expl3 syntax:

```
\ExplSyntaxOn
```

3.2.1 Internal variables

We define a number of internal variables, that are used when reading and formatting a song. All of these variables are meant to be set globally: since there is no notion of a song within a song, we are certain that we will always be either outside of a song or inside a single song.

```
\bool_new:N \g__sp_song_bool            % are we in a song?
\bool_new:N \g__sp_song_start_bool      % are we at the start of a song?
\bool_new:N \g__sp_refrain_first_bool   % does current song start with the
                                        % refrain?
\bool_new:N \g__sp_show_numbers_bool    % should we show the couplet numbers?
\int_new:N  \g__sp_stanzas_per_slide_int % number of stanzas to show on each
                                        % slide (1 or 2)
\dim_new:N  \g__sp_linewidth_dim        % length of the longest line in current
```

```
                                          % song
\tl_new:N   \g__sp_intro_tl              % current song intro
\tl_new:N   \g__sp_refrain_tl            % current song refrain
\seq_new:N  \g__sp_couplets_seq          % current song couplets
\tl_new:N   \g__sp_final_tl              % current song final
\seq_new:N  \g__sp_couplet_indexes_seq   % indexes of couplets to include
```

3.2.2 Internal environments

These are high-level internal environments, that are used in the implementation of user interface environments.

`__sp_refrain` This environment simply formats a song refrain. It is used in the user interface `refrain` environment.

```
\NewDocumentEnvironment {__sp_refrain} {}
  % The environment opening may be followed by a [length], in fact part of its
  % body, and will appear just after the opening of the verse environment and
  % constitute its optional argument.
  {
    \begin{structureenv}
    \begin{verse}
  }
  {
    \end{verse}
    \end{structureenv}
  }
```

`__sp_couplet` This environment simply formats a song couplet. It is used in the user interface `couplet` environment.

```
\NewDocumentEnvironment {__sp_couplet} {}
  % The environment opening may be followed by a [length], in fact part of its
  % body, and will appear just after the opening of the verse environment and
  % constitute its optional argument.
  { \begin{verse} }
  { \end{verse} }
```

`__sp_special` This environments simply formats a song intro of final. It is used in the user interface `intro` and `final` environments.

```
\NewDocumentEnvironment {__sp_special} {}
  % The environment opening may be followed by a [length], in fact part of its
  % body, and will appear just after the opening of the verse environment and
  % constitute its optional argument.
  {
    \begin{em}
    \begin{verse}
  }
  {
    \end{verse}
    \end{em}
  }
```

3.2.3 Internal macros

These are macros that are used in the implementation of the `song` environment.

`\__sp_song_refrain` This macro uses the `__sp_refrain` environment to format the current song refrain.

```
\tl_gset:Nn \__sp_song_refrain
  {
    % Do we know the width of the longest song line?
    \dim_compare:nNnTF \g__sp_linewidth_dim {=} {0pt}
      { \begin{__sp_refrain} }
      { \begin{__sp_refrain} [\g__sp_linewidth_dim] }
    \tl_use:N \g__sp_refrain_tl
    \end{__sp_refrain}
  }
```

(*End definition for `\__sp_song_refrain`.*)

`\__sp_song_couplet:n` This macro uses the `__sp_couplet` environment to format a specified couplet of the current song. It takes a single argument:
`#1` : index of the couplet to format.

```
\cs_gset:Npn \__sp_song_couplet:n #1
  {
    % Do we know the width of the longest song line?
    \dim_compare:nNnTF \g__sp_linewidth_dim {=} {0pt}
      { \begin{__sp_couplet} }
      { \begin{__sp_couplet} [\g__sp_linewidth_dim] }
    \bool_if:NTF \g__sp_show_numbers_bool
      { \flagverse{#1.} }
      {}
    \seq_item:Nn \g__sp_couplets_seq {#1}
    \end{__sp_couplet}
  }
```

(*End definition for `\__sp_song_couplet:n`.*)

`\__sp_song_couplets:n` This macro inserts an containing all couplets of the current song in an `overprint` environment, in groups separated with `\onslide` commands. It takes a single argument:
`#1` : number of couplets to show together on each slide.

```
\cs_gset:Npn \__sp_song_couplets:n #1
  {
    \begin{overprint}
    % Loop on all specified couplets
    \int_step_inline:nn
      { \seq_count:N \g__sp_couplet_indexes_seq }
      {
        % Before every #1 lines, i.e. when (##1 - 1) mod #1 == 0),
        % insert an \onslide
        \int_compare:nNnTF
          { \int_mod:nn { \int_eval:n{##1 - 1} } {#1} } { = } { 0 }
          { \onslide<+> }
          { \vskip \stanzaskip }
        \__sp_song_couplet:n { \seq_item:Nn \g__sp_couplet_indexes_seq {##1} }
      }
    \end{overprint}
  }
```

(*End definition for* `\__sp_song_couplets:n`.)

`\__sp_song_intro` This macro uses the `__sp_special` environment to format the current song intro.

```
\tl_gset:Nn \__sp_song_intro
  {
    % Do we know the width of the longest song line?
    \dim_compare:nNnTF \g__sp_linewidth_dim {=} {0pt}
      { \begin{__sp_special} }
      { \begin{__sp_special} [\g__sp_linewidth_dim] }
    \tl_use:N \g__sp_intro_tl
    \end{__sp_special}
  }
```

(*End definition for* `\__sp_song_intro`.)

`\__sp_song_final` This macro uses the `__sp_refrain` environment to format the current song final.

```
\tl_gset:Nn \__sp_song_final
  {
    % Do we know the width of the longest song line?
    \dim_compare:nNnTF \g__sp_linewidth_dim {=} {0pt}
      { \begin{__sp_special} }
      { \begin{__sp_special} [\g__sp_linewidth_dim] }
    \tl_use:N \g__sp_final_tl
    \end{__sp_special}
  }
```

(*End definition for* `\__sp_song_final`.)

`\__sp_song` This macro inserts the entire song, alternating refrain and couplets in an `overprint` environment.

```
\tl_gset:Nn \__sp_song
  {
    % Is there a song intro?
    \tl_if_empty:NTF \g__sp_intro_tl
    {}
    {
      \visible<1> {\__sp_song_intro}
      % The combination of overprint with verse that comes next somehow adds
      % extra vertical space that needs to be removed.
      \vskip -\stanzaskip
    }

    \begin{overprint}

    % Does the song begin with the refrain?
    \bool_if:NTF \g__sp_refrain_first_bool
      {
        % If so, print an initial refrain
        \onslide<+>
        \__sp_song_refrain
      }
      {}

    % Is there a refrain?
    \tl_if_empty:NTF \g__sp_refrain_tl
      {
        % No refrain, loop on all specified couplets and insert them
        \seq_map_inline:Nn
          \g__sp_couplet_indexes_seq
          {
            \onslide<+>
            \__sp_song_couplet:n {#1}
          }
      }
      {
        % There is a refrain, loop on all specified couplets and, each time,
        % insert both a couplet and the refrain
        \seq_map_inline:Nn
          \g__sp_couplet_indexes_seq
          {
            \onslide<+>
            \__sp_song_couplet:n {#1}
            \onslide<+>
            \__sp_song_refrain
          }
      }
    \end{overprint}

    % Is there a song final?
    \tl_if_empty:NTF \g__sp_final_tl
    {}
    {
      % Add extra spacing
      \vskip \stanzaskip
      \visible<.> {\__sp_song_final}
    }
  }
```

(*End definition for* `\__sp_song`.)

3.3 User interface

These environments constitute our user interface. They allow the user to define songs, refrains and couplets.

`refrain` This environment handles a refrain :

- outside of a song, it uses the `__sp_refrain` environment to directly format it ;
- inside a song, it stores it into `\g__sp_retrain_tl`, so it can be formatted by the end of the `song` environment.

```
\NewDocumentEnvironment {refrain} { +b }
  % The environment opening may be followed by a [length], in fact part of its
  % body, and will appear just after the opening of the __sp_refrain
  % environment and constitute its optional argument.
  {
    % Are we in a song?
    \bool_if:NTF \g__sp_song_bool
```

```
      {
        % We are in a song, are we at its start?
        \bool_if:NTF \g__sp_song_start_bool
          {
            % Indicate that we are no longer at the start of the song
            \bool_gset_false:N\g__sp_song_start_bool
            % and that the refrain comes first
            \bool_gset_true:N\g__sp_refrain_first_bool
          }
          {}
        % Anyway, store the refrain
        \tl_gset:Nn \g__sp_refrain_tl {#1}
      }
      {
        % We are not in a song, use __sp_refrain to format the refrain
          \begin{__sp_refrain}
            #1
          \end{__sp_refrain}
      }
  }
  {}
```

`couplet` This environment handles a couplet, in a similar way:

- outside of a song, it uses the `__sp_couplet` environment to directly format it ;
- inside a song, it stores it by appending it to to `\g__sp_couplets_seq`, so it can be formatted by the end of the `song` environment.

```
\NewDocumentEnvironment {couplet} { +b }
  % The environment opening may be followed by a [length], in fact part of its
  % body, and will appear just after the opening of the __sp_couplet
  % environment and constitute its optional argument.
  {
    % Are we in a song?
    \bool_if:NTF \g__sp_song_bool
      {
        % Are we at in a song, are we at its start?
        \bool_if:NTF \g__sp_song_start_bool
          {
            % Indicate that we are no longer at the start of the song
            \bool_gset_false:N \g__sp_song_start_bool
            % and that the refrain does not come first
            \bool_gset_false:N \g__sp_refrain_first_bool
          }
          {}
        % Anyway, store this couplet
        \seq_gput_right:Nn \g__sp_couplets_seq { {#1} }
      }
      {
        % We are not in a song, use __sp_couplet to format this couplet
          \begin{__sp_couplet}
            #1
          \end{__sp_couplet}
      }
  }
  {}
```

`intro` This environment handles a song intro, in a similar way:

- outside of a song, it uses the `__sp_special` environment to directly format it;
- inside a song, it stores it into `\g__sp_intro_tl` so it can be formatted by the end of the `song` environment.

```
\NewDocumentEnvironment {intro} { +b }
  % The environment opening may be followed by a [length], in fact part of its
  % body, and will appear just after the opening of the __sp_special
  % environment and constitute its optional argument.
  {
    % Are we in a song?
    \bool_if:NTF \g__sp_song_bool
      {
        % We are in a song, store its intro
        \tl_gset:Nn \g__sp_intro_tl {#1}
      }
      {
        % We are not in a song, use __sp_special to format the intro
        \begin{__sp_special}
          #1
        \end{__sp_special}
      }
  }
  {}
```

`final` This environment handles a song final, in a similar way:

- outside of a song, it uses the `__sp_special` environment to directly format it;
- inside a song, it stores it into `\g__sp_final_tl` so it can be formatted by the end of the `song` environment.

```
\NewDocumentEnvironment {final} { +b }
  % The environment opening may be followed by a [length], in fact part of its
  % body, and will appear just after the opening of the __sp_special
  % environment and constitute its optional argument.
  {
    % Are we in a song?
    \bool_if:NTF \g__sp_song_bool
      {
        % We are in a song, store its intro
        \tl_gset:Nn \g__sp_final_tl {#1}
      }
      {
        % We are not in a song, use __sp_special to format the intro
        \begin{__sp_special}
          #1
        \end{__sp_special}
      }
  }
  {}
```

`\longest` This macro measures the length of a song line and stores it, so it can be used by the `song` environment to properly center refrain and couplets. It takes a single argument:

`#1` : a song line to measure.

```
\NewDocumentCommand {\longest} { m } { \settowidth {\g__sp_linewidth_dim} {#1} }
```

(*End definition for* `\longest`. *This function is documented on page 2.*)

`\numbercouplets` This macro can be used within a song to indicate that its couplets should be numbered.

```
    \NewDocumentCommand {\numbercouplets}{}
      { \bool_gset_true:N \g__sp_show_numbers_bool }
```

(*End definition for* `\numbercouplets`. *This function is documented on page 2.*)

`song` This environment is used as a container for entire songs. On opening, it does several things:

1. its stores its arguments into variables with a descriptive name;
2. it clears out any previously stored refrain, couplet, intro, final and longest song line;
3. it sets the `\g__sp_song_bool` variable to indicate that we are inside a song, which will alter the behaviour of the `refrain` and `couplet` environments so they record their content rather than directly formatting it into the document;
4. it sets the `\g__sp_song_start_bool` variable to indicate that we are at the start of the song, which will allow the next `refrain` or `couplet` to tell if the song starts with the refrain or with a couplet;
5. it sets the `\g__sp_show_numbers_bool` variable to false, to indicate that the couplets should not be numbered by default (the user will be able to override this with the `\numbercouplets` command).

This environment takes two arguments:

`#1` : number of stanzas (counting couplets and refrain, when there is one) per slide;

`#2` : list of couplets to include (defaults to all), for instance `1,3,4`.

```
\NewDocumentEnvironment {song} { m o }
  % {number of stanzas per slide (1 or 2)}
  % [list of couplets to include (defaults to all)]
  {
    % Put arguments into variables with understandable names
    \int_gset_eq:NN {\g__sp_stanzas_per_slide_int} {#1}
    \IfNoValueTF {#2}
      { \seq_gclear:N \g__sp_couplet_indexes_seq }
      { \seq_gset_from_clist:Nn \g__sp_couplet_indexes_seq {#2} }

    % Clear out intro, refrain, couplet, final and longest song line
    \tl_gclear:N \g__sp_intro_tl
    \tl_gclear:N \g__sp_refrain_tl
    \seq_gclear:N \g__sp_couplets_seq
    \tl_gclear:N \g__sp_final_tl
    \dim_zero:N {\g__sp_linewidth_dim}

    % Indicate that we are in a song, and at its start
  \bool_gset_true:N \g__sp_song_bool
  \bool_gset_true:N \g__sp_song_start_bool

  % Couplets should not be numbered by default
  \bool_gset_false:N \g__sp_show_numbers_bool
}
```

And on closing:

- if no list of couplet indexes to use have been given, it generates one covering all couplets in order;
- it uses internal functions to insert the intro, refrain, couplets and final into the document, in the right order according to the song structure (refrain or couplet first) and to the formatting instructions (one or two stanzas per slide).

```
{
  % Have we been given indexes of specific couplets to use?
  \seq_if_empty:NTF \g__sp_couplet_indexes_seq
    {
      % If not, generate it from the list of couplets
      \int_step_inline:nn
        { \seq_count:N \g__sp_couplets_seq }
        { \seq_gput_right:Nn \g__sp_couplet_indexes_seq {##1} }
    }
    {}

  % Now we actually start inserting things into the document.
  % How many stanzas per side did the user request?
  \int_compare:nNnTF \g__sp_stanzas_per_slide_int {>} {1}
    {
      % More than one stanza per slide
      %
      % Is there an intro?
      \tl_if_empty:NTF \g__sp_intro_tl
        {}
        {
          \visible<1> {\__sp_song_intro}
          % Adjust vertical spacing depending on whether the refrain or the
          % couplets follow.
          \bool_if:NTF\g__sp_refrain_first_bool
            {
              % Refrain comes next, add extra space
              \vskip \parsep
            }
            {
              % Couplets come next, the combination of their overprint and
              % verse environment somehow adds extra vertical space that
              % needs to be removed.
              \vskip -\stanzaskip
            }
        }

      % Is there a refrain?
      \tl_if_empty:NTF \g__sp_refrain_tl
```

```
          {
            % If there is no refrain, we use \__sp_song_couplets:n to write the
            % couplets, \g__sp_stanzas_per_slide_int at a time.
            \__sp_song_couplets:n { \int_use:N \g__sp_stanzas_per_slide_int }
          }
          {
            % If there is a refrain, we use \__sp_song_refrain to write the
            % refrain and \__sp_song_couplets:n to write overprint with all
            % couplets.

            % Does the song begin with the refrain?
            \bool_if:NTF\g__sp_refrain_first_bool
              {
                \__sp_song_refrain
                \vskip -\stanzaskip
                \__sp_song_couplets:n 1
              }
              {
                \__sp_song_couplets:n 1
                \vskip \stanzaskip
                \__sp_song_refrain
              }
          }

        % Is there a final?
        \tl_if_empty:NTF \g__sp_final_tl
          {}
          {
            % Adjust vertical spacing depending on whether we follow the
            % refrain or the couplets.
            \tl_if_empty:NTF \g__sp_refrain_tl
              {
                % No refrain, we follow the couplets, add extra space
                \vskip \stanzaskip
              }
              {
                % There was a refrain, did it come first?
                \bool_if:NTF \g__sp_refrain_first_bool
                {
                  % Refrain came first, we follow the couplets, add extra space
                  \vskip \stanzaskip
                }
                {
                  % Refrain came last, we follow it, add extra space
                  \vskip \parsep
                }
              }
            \visible<.> {\__sp_song_final}
          }
      }
      {
        % If the user requested one stanza per slide, we use \__sp_song to
        % write the entire song in a single overprint environment.
        \__sp_song
```

```
    }
  % Indicate that we are no longer in a song
  \bool_gset_false:N\g__sp_song_bool
}
```

`\inputsong` This macro starts a `song` environment and `\input`s the song content from an external file.

```
\NewDocumentCommand {\inputsong} { s m m o }
  {
    \IfNoValueTF {#4}
      { \begin{song} {#3} }
      { \begin{song} {#3} [#4] }
    \IfBooleanT {#1}
      { \numbercouplets }
    \input{#2}
    \end{song}
  }
```

(*End definition for* `\inputsong`*. This function is documented on page 3.*)

3.4 Wrapping up

Now that we have defined everything we need, we can leave the expl3 syntax and return to normal TEX syntax:

```
\ExplSyntaxOff
```

Change History

v0.1.0
  General: Initial version 1
v0.2.0
  General: Use `sp` prefix for functions and variables 5
v0.3.0
  General: Define aux functions at top-level 7
    Use expl3 naming conventions 1
v0.4.0
  `__sp_special`: Add an environment to format intro and final 6
  General: Document `intro` and `final` environments 2, 3
  `final`: Add a `final` environment . . . 11
  `intro`: Add an `intro` environment . . 11
  `song`: Include intro and final in formatted song 13
  `\__sp_song`: Include intro and final in complete song 8
  `\__sp_song_final`: Add an environment to format current song final 8
  `\__sp_song_intro`: Add an environment to format current song intro 8
v1.0.0
  General: Add projection advice 5
    Add projection layout advice 4
    Document the `\inputsong` command 3
    First public release 1
  `\inputsong`: Add an `\inputsong` command 15
v1.0.1
  General: Add the generated extension file to Git 1
v1.1.0
  General: Add the `\g__sp_show_numbers_bool` variable to number or not couplets 5
    Allow couplet numbering with

`\numbercouplets` 1
Derivate output file name from input file name if not specified . . . 4
Document the `\numbercouplets` command 2
`\numbercouplets`: New command to enable couplet numbering 12
`song`: Do not number couplets by default 12

`\__sp_song_couplet:n`: Print the couplet number when requested . . 7

v1.2.0
General: Allow couplet numbering with `\inputsong*` 1
Document the `\inputsong*` command 3
`\inputsong`: Add an `\inputsong*` command 15

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.