Package 'subtitles'

July 29, 2026

Version 0.1.1

Title Generate and Render Timed Subtitles Using 'FFmpeg'

Description Tools for generating timed subtitle files and rendering them into video with the 'FFmpeg' command-line interface (<https://ffmpeg.org/>). Converts structured transcription output (such as that produced by 'Whisper' speech-to-text models) into SubRip ('SRT') and Advanced SubStation Alpha ('ASS') subtitle files, with word-level caption timing, styling presets, and line-break control, and burns subtitles into video files using the 'FFmpeg' subtitles filter.

License MIT + file LICENSE

URL https://github.com/cornball-ai/subtitles

BugReports https://github.com/cornball-ai/subtitles/issues

Encoding UTF-8

SystemRequirements ffmpeg (>= 5.0), libass

Imports jsonlite, processx

Suggests tinytest

RoxygenNote 7.3.3

NeedsCompilation no

Author Troy Hernandez [aut, cre] (ORCID: <https://orcid.org/0009-0005-4248-604X>), cornball.ai [cph]

Maintainer Troy Hernandez <troy@cornball.ai>

Repository CRAN

Date/Publication 2026-07-29 16:20:16 UTC

Contents

`burn_subtitles` *Burn Subtitles into Video Using FFmpeg*

Description

Invokes FFmpeg to render subtitles into a video file. Supports SRT and ASS formats.

Usage

```
burn_subtitles(video, subtitles, output, style = NULL, overwrite = TRUE,
               dry_run = FALSE)
```

Arguments

`video`	Path to source video file.
`subtitles`	Path to subtitle file (.srt or .ass).
`output`	Path for output video file.
`style`	Optional ASS style override string passed to FFmpeg's subtitles filter. Example: ‘"FontSize=36,Outline=2,Shadow=1"‘.
`overwrite`	Logical. If ‘TRUE‘ (default), overwrite output without prompting. If ‘FALSE‘, refuse to overwrite existing files.
`dry_run`	Logical. If ‘TRUE‘, return the FFmpeg command without executing it. Useful for debugging. Default is ‘FALSE‘.

Details

This function requires FFmpeg (>= 5.0) with libass support on your system PATH. Verify installation: ‘system("ffmpeg -version")‘.

FFmpeg invocation is transparent: all arguments are passed directly to the ‘ffmpeg‘ binary. Errors surface via processx with stderr visible for diagnostics.

The function uses FFmpeg's ‘subtitles‘ filter with audio passthrough (‘-c:a copy‘).

Value

Invisibly returns the ‘output‘ path. If ‘dry_run = TRUE‘, returns the FFmpeg command as a character string. On a real run a JSON call record is also written at ‘<output>.json‘ alongside the video, holding the resolved arguments and facts about the produced file (fps, dimensions, duration). Disable the sidecar with ‘options(subtitles.sidecar = FALSE)‘.

Examples

```
## Not run:
  # Basic usage
  burn_subtitles("input.mp4", "captions.srt", "output.mp4")

  # With custom styling
  burn_subtitles(
    "input.mp4",
    "captions.srt",
    "output.mp4",
    style = "FontSize=36,Outline=2,Shadow=1"
  )

## End(Not run)
```

`validate_ass` *Validate ASS Subtitle File*

Description

Checks an ASS subtitle file for common issues that can cause rendering failures. Useful for debugging subtitle problems before burning them into video.

Usage

```
validate_ass(file)
```

Arguments

`file`	Path to ASS subtitle file to validate.

Details

This function checks for common ASS file issues:

Fatal errors (will stop): - File does not exist or cannot be read

Warnings (will continue): - Missing PlayResX or PlayResY (causes off-screen rendering) - No Dialogue lines found (nothing to render) - Zero-length karaoke tags (backslash-k0) present (causes silent failures) - Missing or zero margins in Dialogue lines (may be covered by UI) - Missing or invalid Style definition

Use this function to debug ASS files that aren't rendering as expected. Per-check progress is reported via 'message()' and can be silenced with 'suppressMessages()'.

Value

Invisibly returns 'TRUE' if validation passes, 'FALSE' if warnings were issued. Stops execution only for fatal errors.

Examples

```
x <- structure(
  list(data = data.frame(
    from = "00:00:01.000", to = "00:00:03.000", text = "Hello world")),
  class = "whisper_transcription")
f <- tempfile(fileext = ".ass")
whisper_to_ass(x, f, karaoke = FALSE)

validate_ass(f)
unlink(c(f, paste0(f, ".json")))
```

`whisper_to_ass` *Convert Whisper Transcription to Advanced SubStation Alpha Format*

Description

Converts transcription to ASS format with optional word-level karaoke timing.

Usage

```
whisper_to_ass(x, file, karaoke = TRUE, playres = c(1080, 1080),
               max_chars_per_line = 22, preset = "shorts", max_lines = 2)
```

Arguments

`x`	An object of class ‘"whisper_transcription"‘.
`file`	Path where the ASS file will be written.
`karaoke`	Logical. If ‘TRUE‘ (default), include word-level timing using the ‘tokens‘ field. If ‘FALSE‘, use segment-level timing only.
`playres`	Numeric vector of length 2 specifying PlayResX and PlayResY (video resolution). Defaults to ‘c(1080, 1080)‘ for square (1:1) video.
`max_chars_per_line`	Maximum characters per line before wrapping when ‘karaoke = FALSE‘. Default is 22. Set to ‘NULL‘ to disable line breaking. Karaoke mode computes its own pixel-width line breaks instead.
`preset`	Style preset for subtitle rendering: ‘"shorts"‘ (default, large fonts with thick outlines for vertical video), ‘"square"‘ (Instagram-style opaque grey box), or ‘"horizontal"‘ (smaller fonts for 16:9 landscape).
`max_lines`	Maximum number of subtitle lines on screen at once (default 2). Segments that would produce more lines are split into multiple timed events. Set to ‘NULL‘ to disable splitting.

Details

When ‘karaoke = TRUE‘, the function uses the ‘tokens‘ field (built automatically from a ‘words‘ field when ‘tokens‘ is missing) to emit word-by-word caption events: each word is appended to the on-screen text as it is spoken, with per-word font sizing (shorter words render larger). Line breaks are pre-computed from pixel widths so earlier words stay in place as new words appear. If ‘karaoke = FALSE‘, only segment-level timing from the ‘data‘ field is used.

Karaoke guardrails: When karaoke mode is enabled, the function: - Validates that the ‘tokens‘ field exists and has required columns - Filters out punctuation-only tokens for cleaner rendering - Provides clear error messages if karaoke requirements aren’t met

The ASS format includes styling information in the header (font size, colors, positioning, etc.).

The ‘playres‘ parameter sets the video resolution for subtitle rendering. Common values: ‘c(1920, 1080)‘ for 16:9, ‘c(1080, 1920)‘ for 9:16 (vertical), ‘c(1080, 1080)‘ for 1:1 (square).

The ‘preset‘ parameter provides optimized styling configurations: - ‘"shorts"‘: Large fonts (56pt), thick outline (3px), shadow, optimized for vertical/square videos - ‘"square"‘: Instagram-style opaque grey box (Roboto 48pt bold, BorderStyle 3) - ‘"horizontal"‘: Medium fonts (42pt), thinner outline (2px), optimized for 16:9 landscape videos

Value

Invisibly returns the ‘file‘ path for piping. A JSON call record is also written at ‘<file>.json‘ alongside the output, holding the resolved arguments and facts about the produced file (event count, time span). Disable the sidecar with ‘options(subtitles.sidecar = FALSE)‘.

See Also

[whisper_to_srt()] for simpler SRT format.

Examples

```
# Toy transcription object; real ones come from speech-to-text tooling
x <- structure(
  list(
    data = data.frame(
      from = c("00:00:01.000", "00:00:05.000"),
      to = c("00:00:05.000", "00:00:10.000"),
      text = c("Hello world", "This is a test")),
    tokens = data.frame(
      segment = c(1, 1, 2, 2, 2, 2),
      token = c("Hello", "world", "This", "is", "a", "test"),
      token_from = c("00:00:01.000", "00:00:02.000", "00:00:05.000",
                     "00:00:06.000", "00:00:07.000", "00:00:08.000"),
      token_to = c("00:00:02.000", "00:00:05.000", "00:00:06.000",
                   "00:00:07.000", "00:00:08.000", "00:00:10.000"))),
  class = "whisper_transcription")

# With karaoke (word-by-word captions)
f <- tempfile(fileext = ".ass")
whisper_to_ass(x, f, karaoke = TRUE)
```

```
# Segment-level only, 16:9 canvas with horizontal preset
f2 <- tempfile(fileext = ".ass")
whisper_to_ass(x, f2, karaoke = FALSE,
               playres = c(1920, 1080), preset = "horizontal")

unlink(c(f, f2, paste0(c(f, f2), ".json")))
```

`whisper_to_srt` *Convert Whisper Transcription to SubRip SRT Format*

Description

Converts a structured transcription object (e.g., from audio.whisper) into a SubRip (.srt) subtitle file with numbered segments and timecodes.

Usage

```
whisper_to_srt(x, file)
```

Arguments

`x`	An object of class ‘"whisper_transcription"‘ with a ‘data‘ field containing ‘from‘, ‘to‘, and ‘text‘ columns.
`file`	Path where the SRT file will be written.

Details

The input must inherit from class ‘"whisper_transcription"‘. The ‘data‘ field must contain: - ‘from‘: Start time as "HH:MM:SS.mmm" - ‘to‘: End time as "HH:MM:SS.mmm" - ‘text‘: Segment text

Value

Invisibly returns the ‘file‘ path for piping. A JSON call record is also written at ‘<file>.json‘ alongside the output, holding the resolved arguments and facts about the produced file (event count, time span). Disable the sidecar with ‘options(subtitles.sidecar = FALSE)‘.

See Also

[whisper_to_ass()] for ASS format with styling support.

Examples

```
# Toy transcription object; real ones come from speech-to-text tooling
# such as the audio.whisper package
x <- structure(
  list(data = data.frame(
    from = c("00:00:00.000", "00:00:02.500"),
    to = c("00:00:02.500", "00:00:05.000"),
```

```
    text = c("Hello world", "Goodbye"))),
  class = "whisper_transcription")

f <- tempfile(fileext = ".srt")
whisper_to_srt(x, f)
readLines(f)
unlink(c(f, paste0(f, ".json")))
```

`words_to_tokens` *Convert word-level timestamps to tokens format*

Description

Converts a data frame of word timestamps (as returned by whisper with ‘word_timestamps = TRUE‘) into the ‘tokens‘ format expected by [whisper_to_ass()] for karaoke rendering.

Usage

```
words_to_tokens(words, segments)
```

Arguments

`words`	Data frame with columns ‘word‘ (character), ‘start‘ (numeric seconds), ‘end‘ (numeric seconds).
`segments`	Data frame with columns ‘start‘ (numeric seconds), ‘end‘ (numeric seconds), ‘text‘ (character). Used to assign words to segments.

Value

Data frame with columns ‘segment‘, ‘token‘, ‘token_from‘, ‘token_to‘ suitable for use as ‘x$tokens‘ in a whisper_transcription object.

Examples

```
words <- data.frame(
  word = c("Hello", "world"),
  start = c(0.0, 0.6),
  end = c(0.5, 1.2))
segments <- data.frame(start = 0, end = 1.2, text = "Hello world")
words_to_tokens(words, segments)
```

Index