Package 'audubon'

April 22, 2026

Title Japanese Text Processing Tools

Version 0.6.3

Description A collection of Japanese text processing tools for filling Japanese iteration marks, Japanese character type conversions, segmentation by phrase, and text normalization which is based on rules for the 'Sudachi' morphological analyzer and the 'NEologd' (Neologism dictionary for 'MeCab'). These features are specific to Japanese and are not implemented in 'ICU' (International Components for Unicode).

License Apache License (>= 2)

URL https://github.com/paithiov909/audubon, https://paithiov909.github.io/audubon/

BugReports https://github.com/paithiov909/audubon/issues

Depends R (>= 4.1)

Imports dplyr (>= 1.1.0), magrittr, purrr, readr, rlang, stringi (>= 1.8.3), V8

Suggests roxygen2, scales, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.3

NeedsCompilation no

Author Akiru Kato [cre, aut], Koki Takahashi [cph] (Author of japanese.js), Shuhei Iitsuka [cph] (Author of budoux), Taku Kudo [cph] (Author of TinySegmenter)

Maintainer Akiru Kato <paithiov909@gmail.com>

Repository CRAN

Date/Publication 2026-04-22 14:30:02 UTC

Contents

`default_format` *Default Japanese date format*

Description

Returns the default date format string used for Japanese calendar date parsing and formatting.

This helper function exists to provide a UTF-8 encoded format string without embedding non-ASCII characters directly in function defaults.

Usage

```
default_format()
```

Value

A character string representing a Japanese calendar date format.

Examples

```
default_format()
```

hiroba *Whole tokens of 'Porano no Hiroba' written by Miyazawa Kenji from Aozora Bunko*

Description

A tidy text data of `audubon::polano` that tokenized with 'MeCab'.

Usage

```
hiroba
```

Format

An object of class `data.frame` with 26849 rows and 5 columns.

Examples

```
head(hiroba)
```

label_date_jp *Japanese date labeller for ggplot2*

Description

Formats date labels using the Japanese calendar system and returns labels suitable for use with ggplot2 scales.

Usage

```
label_date_jp(labels, format = default_format(), tz = NULL)

label_date_jp_gen(format = default_format(), tz = NULL)
```

Arguments

`labels`	A vector of values coercible to Date objects.
`format`	A date-time format string following ICU conventions.
`tz`	A time zone used when coercing values to Date objects.

Details

This labeller formats dates according to a locale-aware Japanese calendar, allowing era-based representations such as Reiwa or Heisei. The output is intended for discrete or continuous date scales in ggplot2.

Value

- `label_date_jp()` returns a character vector of formatted date labels.
- `label_date_jp_gen()` returns a labeller function for use in ggplot2 scales.

Examples

```
## Not run:
date_range <- function(start, days) {
  start <- as.POSIXct(start)
  c(start, start + days * 24 * 60 * 60)
}
two_months <- date_range("2025-12-31", 60)
label_date_jp(two_months)

if (requireNamespace("scales", quietly = TRUE)) {
  scales::demo_datetime(two_months, labels = label_date_jp_gen())
}

## End(Not run)
```

`label_wrap_jp` *Japanese word-wrapping labeller for ggplot2*

Description

Wraps character strings using Japanese phrase boundaries and returns labels suitable for use with ggplot2 scales.

Usage

```
label_wrap_jp(labels, wrap = 16, width = 50, collapse = "\n")

label_wrap_jp_gen(wrap = 16, width = 50, collapse = "\n")
```

Arguments

`labels`	A character vector of labels to wrap.
`wrap`	An integer giving the target number of characters per line.
`width`	An integer giving the maximum total width of the wrapped label.
`collapse`	A character string used to join wrapped lines.

Details

This labeller uses ICU-based Japanese phrase boundary detection to insert line breaks at natural word boundaries. Long labels can be truncated to a fixed display width with an ellipsis.

Value

- `label_wrap_jp()` returns a character vector of wrapped labels.
- `label_wrap_jp_gen()` returns a labeller function for use in ggplot2 scales.

Examples

```
## Not run:
label_wrap_jp(polano[4:6], width = 32)

if (requireNamespace("scales", quietly = TRUE)) {
  scales::demo_discrete(polano[4:6], labels = label_wrap_jp_gen())
}

## End(Not run)
```

---

polano *Whole text of 'Porano no Hiroba' written by Miyazawa Kenji from Aozora Bunko*

Description

Whole text of 'Porano no Hiroba' written by Miyazawa Kenji from Aozora Bunko

Usage

```
polano
```

Format

An object of class `character` of length 899.

Details

A dataset containing the text of Miyazawa Kenji's novel "Porano no Hiroba" which was published in 1934, the year after Kenji's death. Copyright of this work has expired since more than 70 years have passed after the author's death.

The UTF-8 plain text is sourced from https://www.aozora.gr.jp/cards/000081/card1935.html and is cleaned of meta data.

Source

https://www.aozora.gr.jp/cards/000081/files/1935_ruby_19924.zip

Examples

```
head(polano)
```

read_rewrite_def *Read rewrite definition file*

Description

Reads a rewrite definition file used for Japanese text normalization.

This function parses a tab-delimited definition file and returns a list of rewrite rules and ignored characters suitable for use with `strj_rewrite_as_def()`.

Usage

```
read_rewrite_def(
  def_path = system.file("def/rewrite.def", package = "audubon")
)
```

Arguments

`def_path`	A file path to a rewrite definition file.

Value

A list containing rewrite rules and ignored characters.

Examples

```
str(read_rewrite_def())
```

strj-hira-kana *Convert Japanese kana characters*

Description

Converts Japanese text between hiragana and katakana representations.

These functions transform kana characters while preserving non-kana characters. The conversion is based on a JavaScript implementation and handles certain historical or contracted kana forms that are not covered by standard Unicode transliteration alone.

Usage

```
strj_hiraganize(text)

strj_katakanize(text)
```

Arguments

`text`	A character vector containing Japanese text.

Details

The conversion behavior is largely compatible with ICU-based transliteration, with additional support for selected combined or historical kana characters.

Value

A character vector with kana characters converted to the target script.

Examples

```
strj_hiraganize(
  c(
    paste0(
      "\u3042\u306e\u30a4\u30fc\u30cf\u30c8",
      "\u30fc\u30f4\u30a9\u306e\u3059\u304d",
      "\u3068\u304a\u3063\u305f\u98a8"
    ),
    "\u677f\u57a3\u6b7b\u30b9\U0002a708"
  )
)
strj_katakanize(
  c(
    paste0(
      "\u3042\u306e\u30a4\u30fc\u30cf\u30c8",
      "\u30fc\u30f4\u30a9\u306e\u3059\u304d",
      "\u3068\u304a\u3063\u305f\u98a8"
    ),
    "\u672c\u65e5\u309f\u304b\u304d\u6c37\u89e3\u7981"
  )
)
```

`strj_fill_iter_mark` *Fill Japanese iteration marks*

Description

Replaces Japanese iteration marks in character strings with the corresponding repeated characters.

This function scans each input string and expands iteration marks such as odoriji by inferring the characters to be repeated from the surrounding context. The implementation is heuristic and intended for practical text normalization rather than complete linguistic accuracy.

Usage

```
strj_fill_iter_mark(text)
```

Arguments

`text`	A character vector containing Japanese text.

Details

The restoration is based on local character context and may be incomplete for iteration marks that refer to longer or more complex spans.

Value

A character vector in which iteration marks are replaced with the inferred repeated characters.

Examples

```
strj_fill_iter_mark(c(
  "\u3042\u3044\u3046\u309d\u3003\u304b\u304d",
  "\u91d1\u5b50\u307f\u3059\u309e",
  "\u306e\u305f\u308a\u3033\u3035\u304b\u306a",
  "\u3057\u308d\uff0f\u2033\uff3c\u3068\u3057\u305f"
))
```

`strj_normalize` *Convert text following the rules of 'NEologd'*

Description

Converts characters into normalized style following the rule that is recommended by the Neologism dictionary for 'MeCab'.

Usage

```
strj_normalize(text)
```

Arguments

`text`	A character vector containing Japanese text.

Value

A character vector with normalized text.

See Also

https://github.com/neologd/mecab-ipadic-neologd/wiki/Regexp.ja

Examples

```
strj_normalize(
  paste0(
    "\u2015\u2015\u5357\u30a2\u30eb\u30d7\u30b9",
    "\u306e\u3000\u5929\u7136\u6c34-\u3000\uff33",
    "\uff50\uff41\uff52\uff4b\uff49\uff4e\uff47*",
    "\u3000\uff2c\uff45\uff4d\uff4f\uff4e+",
    "\u3000\u30ec\u30e2\u30f3\u4e00\u7d5e\u308a"
  )
)
```

`strj_parse_date` *Parse Japanese calendar dates*

Description

Parses Japanese calendar date strings into POSIXct objects.

This function parses date strings formatted with the Japanese calendar system and converts them to POSIXct values using locale-aware ICU parsing.

Usage

```
strj_parse_date(date, format = default_format(), tz = NULL)
```

Arguments

`date`	A character vector containing Japanese calendar date strings.
`format`	A date-time format string following ICU conventions.
`tz`	A time zone used for the resulting POSIXct values.

Details

Partial date specifications are interpreted according to ICU parsing rules and may result in completion with the current date or time components.

Value

A POSIXct vector representing the parsed dates.

Examples

```
## Not run:
strj_parse_date("\u4ee4\u548c2\u5e747\u67086\u65e5")

## End(Not run)
```

`strj_rewrite_as_def` *Rewrite Japanese text using normalization rules*

Description

Rewrites Japanese text according to a set of normalization rules modeled after Sudachi dictionary definitions.

Usage

```
strj_rewrite_as_def(text, as = read_rewrite_def())
```

Arguments

`text`	A character vector containing Japanese text.
`as`	A rewrite definition object as returned by `read_rewrite_def()`.

Details

This function applies character-level rewrite rules to normalize variant forms while optionally ignoring specified characters. The implementation is a simplified and heuristic adaptation of Sudachi-style normalization.

The rewrite process is based on fixed replacement rules and does not aim to fully reproduce Sudachi's normalization behavior.

Value

A character vector with rewritten and normalized text.

Examples

```
strj_rewrite_as_def(
  paste0(
    "\u2015\u2015\u5357\u30a2\u30eb",
    "\u30d7\u30b9\u306e\u3000\u5929",
    "\u7136\u6c34-\u3000\uff33\uff50",
    "\uff41\uff52\uff4b\uff49\uff4e\uff47*",
    "\u3000\uff2c\uff45\uff4d\uff4f\uff4e+",
    "\u3000\u30ec\u30e2\u30f3\u4e00\u7d5e\u308a"
  )
)
strj_rewrite_as_def(
  "\u60e1\u3068\u5047\u9762\u306e\u30eb\u30fc\u30eb",
  read_rewrite_def(system.file("def/kyuji.def", package = "audubon"))
)
```

`strj_romanize` *Romanize Japanese text*

Description

Converts Japanese kana text to Latin script using a selectable romanization system.

This function transliterates Japanese text into romaji according to the specified convention. Non-kana characters are omitted from the output.

Usage

```
strj_romanize(
  text,
 config = c("wikipedia", "traditional hepburn", "modified hepburn", "kunrei", "nihon")
)
```

Arguments

`text`	A character vector containing Japanese text.
`config`	A string specifying the romanization system to use.

Details

Supported romanization systems include variants of Hepburn as well as Kunrei-shiki and Nihon-shiki conventions.

Value

A character vector containing romanized text.

Examples

```
strj_romanize(
  paste0(
    "\u3042\u306e\u30a4\u30fc\u30cf\u30c8",
    "\u30fc\u30f4\u30a9\u306e\u3059\u304d",
    "\u3068\u304a\u3063\u305f\u98a8"
  )
)
```

`strj_tokenize` *Tokenize Japanese text*

Description

Tokenizes Japanese character strings using a selectable segmentation engine and returns the result as a list or a data frame.

This function provides a unified interface to multiple Japanese text segmentation backends. External command-based engines were removed in v0.6.0, and all tokenization is performed using in-process implementations.

`strj_segment()` and `strj_tinyseg()` are aliases for `strj_tokenize()` with the `"budoux"` and `"tinyseg"` engines, respectively.

Usage

```
strj_tokenize(
  text,
  format = c("list", "data.frame"),
  engine = c("stringi", "budoux", "tinyseg"),
  split = FALSE,
  ...
)

strj_segment(text, format = c("list", "data.frame"), split = FALSE)

strj_tinyseg(text, format = c("list", "data.frame"), split = FALSE)
```

Arguments

`text`	A character vector of Japanese text to tokenize.
`format`	A string specifying the output format.
`engine`	A string specifying the tokenization engine to use.
`split`	A logical value indicating whether `text` should be split into individual sentences before tokenization.
`...`	Additional arguments passed to the underlying engine.

Details

The following engines are supported:

- `"stringi"`: Uses ICU-based boundary analysis via stringi.
- `"budoux"`: Uses a rule-based Japanese phrase segmentation algorithm.
- `"tinyseg"`: Uses a TinySegmenter-compatible statistical model.

The legacy `"mecab"` and `"sudachipy"` engines were removed in v0.6.0.

Value

If `format = "list"`, a named list of character vectors, one per input element. If `format = "data.frame"`, a data frame containing document identifiers and tokenized text.

Examples

```
strj_tokenize(
  paste0(
    "\u3042\u306e\u30a4\u30fc\u30cf\u30c8",
    "\u30fc\u30f4\u30a9\u306e\u3059\u304d",
    "\u3068\u304a\u3063\u305f\u98a8"
  )
)
strj_tokenize(
  paste0(
    "\u3042\u306e\u30a4\u30fc\u30cf\u30c8",
    "\u30fc\u30f4\u30a9\u306e\u3059\u304d",
    "\u3068\u304a\u3063\u305f\u98a8"
  ),
  format = "data.frame"
)
```

---

strj_transcribe_num *Transcribe integers into Japanese kanji numerals*

Description

Converts integer values to their Japanese kanji numeral representations.

This function transcribes integers up to the trillions place into kanji numerals. For larger numbers or more comprehensive numeral support, consider using the CRAN package arabic2kansuji.

Usage

```
strj_transcribe_num(int)
```

Arguments

`int`	An integer vector to transcribe.

Value

A character vector containing kanji numeral representations.

Examples

```
strj_transcribe_num(c(10L, 31415L))
```

Index