Package: stringrs (via r-universe)

May 27, 2026

Title Fast Regex Matching with Rust and Parallel Processing

Version 0.0.0.9000

Description High-performance regex matching for R using Rust via extendr. Supports both standard regex (fast) and fancy-regex (backrefs/lookaheads). Includes smart defaults for parallel processing and multiple output formats.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/rextendr/version 0.4.2

Imports tibble

Suggests testthat (>= 3.0.0), bench, mirai, future, promises, furrr, ggplot2, dplyr, tidyr

Config/testthat/edition 3

SystemRequirements Cargo (Rust package manager)

Config/pak/sysreqs libclang-dev

Repository https://brancengregory.r-universe.dev

Date/Publication 2026-03-28 08:12:18 UTC

RemoteUrl https://github.com/brancengregory/stringrs

RemoteRef HEAD

RemoteSha b49b82e3650b664cf21acb6191794c4595e4fdd1

Contents

`stringrs-package` *stringrs: Fast Regex Matching with Rust and Parallel Processing*

Description

High-performance regex matching for R using Rust via extendr. Supports both standard regex (fast) and fancy-regex (backrefs/lookaheads). Includes smart defaults for parallel processing and multiple output formats.

Author(s)

Maintainer: Brancen Gregory <brancengregory@gmail.com>

`choose_engine` *Choose regex engine based on pattern complexity*

Description

Choose regex engine based on pattern complexity

Usage

```
choose_engine(patterns)
```

`detect_multi` *Detect multiple patterns using optimized flat array output*

Description

Detect multiple patterns using optimized flat array output

Usage

```
detect_multi(strings, patterns, output, engine, parallel, chunk_size)
```

detect_single *Detect single pattern match (uses cached compilation)*

Description

Detect single pattern match (uses cached compilation)

Usage

```
detect_single(strings, pattern, engine, parallel)
```

r_string_detect_fancy_cached *Fancy-regex single pattern with caching*

Description

Fancy-regex single pattern with caching

Usage

```
r_string_detect_fancy_cached(strings, pattern, parallel)
```

r_string_detect_multi_fancy_optimized *Multi-pattern fancy-regex with caching*

Description

Multi-pattern fancy-regex with caching

Usage

```
r_string_detect_multi_fancy_optimized(
  strings,
  patterns,
  parallel_strategy,
  chunk_size
)
```

`r_string_detect_multi_regex_optimized` *OPTIMIZED: Multi-pattern detection using flat boolean array Returns: flat array n_strings × n_patterns as R integer vector (0/1)*

Description

OPTIMIZED: Multi-pattern detection using flat boolean array Returns: flat array n_strings × n_patterns as R integer vector (0/1)

Usage

```
r_string_detect_multi_regex_optimized(
  strings,
  patterns,
  parallel_strategy,
  chunk_size
)
```

`r_string_detect_regex_cached` *OPTIMIZED: Single pattern detection*

Description

OPTIMIZED: Single pattern detection

Usage

```
r_string_detect_regex_cached(strings, pattern, parallel)
```

`string_detect` *Detect string matches with smart defaults (optimized)*

Description

Uses global regex cache, thread-local instances for zero contention, flat array output, and optimized parallel chunking.

Usage

```
string_detect(
  strings,
  patterns,
  output = c("wide", "long"),
  engine = c("auto", "regex", "fancy_regex"),
  parallel = c("auto", "sequential", "string_parallel", "pattern_parallel"),
  chunk_size = NULL
)
```

Arguments

strings	Character vector of strings to search
patterns	Character vector of regex patterns (single or multiple)
output	Format of output: "wide" (default) or "long"
engine	Regex engine: "auto" (default), "regex", or "fancy_regex"
parallel	Parallel strategy: "auto" (default), "sequential", "string_parallel", "pattern_parallel"
chunk_size	Number of strings per chunk for parallel processing. NULL for auto.

Value

For single pattern: logical vector. For multiple patterns: tibble (wide or long format).

Index