Package: naxp (via r-universe)

November 4, 2026

Title Encoded ASCII Expressions

Version 0.1.0

Description Converts alphanumeric codes such as postcodes to whole numbers and back again, following the 'naxp' standard <https://naxp.org/spec/>. A naxp ('encoded ASCII expression') is a pattern, defined using a similar syntax to regular expressions, that maps each text it accepts to a number from 1 to a fixed maximum, and any other text to 0. The same pattern maps text to numbers identically across all 'naxp' libraries in all languages on all platforms (subject only to the availability of types able to store large enough integers). Standard naxps are defined for common code systems and are built into all 'naxp' libraries. Two naxps can be compared to determine, among other things, whether stored values keep their meaning when one replaces the other. Wraps the 'naxp' 'C++' library <https://naxp.org/api/#cpp>.

License Apache License (== 2.0)

URL https://naxp.org/, https://r-naxp.logmu.org/, https://github.com/logmu-org/r-naxp

BugReports https://github.com/logmu-org/r-naxp/issues

Depends R (>= 4.1.0)

Suggests jsonlite, knitr, pillar, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

LinkingTo cpp11

Config/roxygen2/markdown TRUE

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

SystemRequirements C++17

Roxygen list(markdown = TRUE)

Repository https://logmu-org.r-universe.dev

Date/Publication 2026-11-02 16:39:40 UTC

RemoteUrl https://github.com/logmu-org/r-naxp

RemoteRef HEAD

RemoteSha 71c1220637ed0102c310c6bdfa22832f7c65a963

Contents

naxp_accepts 2
naxp_compare 3
naxp_emit 4
naxp_encode 5
naxp_encoded 6
naxp_parse 7
naxp_properties 9
naxp_standard 10

Index **11**

`naxp_accepts` *Check text against a naxp*

Description

`naxp_accepts()` says whether a naxp accepts each text, and `naxp_canonical_form()` gives the form each accepted text decodes to.

Usage

```
naxp_accepts(text, naxp)

naxp_canonical_form(text, naxp)
```

Arguments

`text`	A character vector.
`naxp`	A naxp from `naxp_parse()`, a pattern string or a naxp_encoded vector, whose naxp is used. For `naxp_decode()`, it is needed only where `x` is not a naxp_encoded vector.

Details

Neither needs the text's encoded value, so both work for a naxp of any size.

Value

- `naxp_accepts()`: a logical vector the same length as `text`, with its names. It is `FALSE`, never `NA`, for `NA` and for invalid text.
- `naxp_canonical_form()`: a character vector the same length as `text`, with its names, holding `NA` for `NA` and for invalid text.

Examples

```
postcode_naxp <- naxp_parse(r"(\9\9\9\9 \s!! \A\A)")

naxp_accepts(c("1012AB", "1012 AB", "1012-AB"), postcode_naxp)
naxp_canonical_form(c("1012AB", "1012 AB", "1012-AB"), postcode_naxp)
```

---

naxp_compare *Compare two naxps*

Description

`naxp_compare()` says how `b` stands to `a`: what happens to text and encoded values held under `a` if `b` replaces it. `naxp_first_divergent_value()` finds the lowest encoded value the two decode differently.

Usage

```
naxp_compare(a, b, budget = NULL)

naxp_first_divergent_value(a, b)
```

Arguments

`a`	The naxp the data was encoded with: a naxp from `naxp_parse()`, a pattern string or a naxp_encoded vector, whose naxp is used.
`b`	The naxp proposed to replace it, likewise.
`budget`	`NULL` for the library's default budget of 200 000 product states, or a positive whole number.

Details

`naxp_compare()` gives three set relationships, each from `a`'s point of view:

- `accepted_text`: how the strings `a` accepts stand to those `b` accepts.
- `encoding`: how the (text, encoded value) pairs `a` defines stand to those `b` defines. Replacing `a` with `b` is safe for stored values exactly when this is `"equal"` or `"subset_of"`.
- `canonical_forms`: how the canonical forms `a` prints stand to those `b` prints.

Each is `"equal"`, `"subset_of"`, `"superset_of"` or `"incomparable"`.

The encoding relationship is decided by walking the two naxps together, which has a budget. No naxp anybody has reason to write comes near the default, but a pair that does cannot be decided, and `naxp_compare()` then raises an error rather than guess.

Value

- `naxp_compare()`: a named character vector with elements `accepted_text`, `encoding` and `canonical_forms`.
- `naxp_first_divergent_value()`: a single encoded value, or 0 where every value both naxps hold decodes alike. Values only one naxp holds do not count. It is an integer where both naxps' encoded values are integers, and otherwise a double.

Examples

```
must_have_space <- naxp_parse(r"(\9\9\9\9 \s \A\A)")
prefers_space <- naxp_parse(r"(\9\9\9\9 \s!! \A\A)")
prefers_no_space <- naxp_parse(r"(\9\9\9\9 \s!? \A\A)")

naxp_compare(must_have_space, prefers_space)

naxp_compare(must_have_space, prefers_no_space)
naxp_first_divergent_value(must_have_space, prefers_no_space)
```

---

naxp_emit *Emit a naxp as source code*

Description

Writes a naxp as source code in another language: a fragment of declarations that answer the same questions this package does, for that one naxp, with no dependency on any naxp library. The header comment, includes, namespace or wrapping class around it are yours to add.

Usage

```
naxp_emit(
  naxp,
  language = c("csharp", "javascript", "c", "cpp"),
  prefix = "",
  value_type = c("uint64", "int64", "uint32", "int32", "uint16", "int16", "uint8",
    "int8"),
  initial_indent = "",
  new_line = "\n",
  indent = "\t"
)
```

Arguments

`naxp`	A naxp from `naxp_parse()`, a pattern string or a `naxp_encoded` vector, whose naxp is used.
`language`	The language to emit: `"csharp"`, `"javascript"`, `"c"` (C99) or `"cpp"` (C++17).
`prefix`	What every generated name starts with, so several naxps can share one scope. `""` gives the bare names. Each language applies its own casing convention to it.

`value_type`	The integer type the generated code uses for encoded values. The naxp's maximum encoded value must fit it.
`initial_indent`	What every line starts with ahead of its own indentation, so the fragment can sit inside an indented wrapper.
`new_line`	What ends every line.
`indent`	What one level of indentation is written as.

Value

A single string: the fragment.

Examples

```
cat(naxp_emit(r"(\9\9\9\9 \s!! \A\A)", "javascript", prefix = "postcodeNL"))
```

`naxp_encode` *Encode and decode text*

Description

`naxp_encode()` encodes text as encoded values under a naxp, and `naxp_decode()` decodes them back to the canonical form of the text.

Usage

```
naxp_encode(text, naxp)

naxp_decode(x, naxp = NULL)
```

Arguments

`text`	A character vector.
`naxp`	A naxp from `naxp_parse()`, a pattern string or a naxp_encoded vector, whose naxp is used. For `naxp_decode()`, it is needed only where `x` is not a naxp_encoded vector.
`x`	A naxp_encoded vector, or an integer or double vector of encoded values. Bare values must be whole numbers from 0 to the naxp's maximum encoded value, or `NA`.

Details

Each text a naxp accepts encodes to a whole number from 1 to its maximum encoded value. Several texts may share a value, such as a postcode written with and without a space, and each value decodes to one canonical form.

Text the naxp does not accept is invalid and encodes to 0, as in every naxp library. So does `NA`, and so does any text that is not ASCII, whatever its marked encoding. Encoded values are never `NA`.

`naxp_decode()` returns `NA` for 0, because no other string can stand for it: a naxp may accept the empty string.

Encoded values are held as integers where the naxp's maximum encoded value is at most 2^31 - 1, and as doubles where it is at most 2^53. Above 2^53, which is the largest whole number R holds exactly, both functions raise an error.

Value

- `naxp_encode()`: a naxp_encoded vector the same length as `text`, with its names.
- `naxp_decode()`: a character vector the same length as `x`, with its names, holding `NA` where a value is 0 or `NA`.

See Also

`naxp_accepts()` and `naxp_canonical_form()` to check text without encoding it.

Examples

```
postcode_naxp <- naxp_parse(r"(\9\9\9\9 \s!! \A\A)")

postcodes <- naxp_encode(c("1012AB", "1012 AB", "not a postcode"), postcode_naxp)
postcodes
as.integer(postcodes)

naxp_decode(postcodes)
naxp_decode(684114L, postcode_naxp)
```

---

naxp_encoded *Encoded values*

Description

A `naxp_encoded` vector holds encoded values together with the naxp that encoded them, so it can always be decoded. `naxp_encode()` returns one, and `naxp_encoded()` makes one from bare values, such as values read from a database or a file.

Usage

```
naxp_encoded(values, naxp)
```

Arguments

`values`	An integer or double vector of whole numbers from 0 to the naxp's maximum encoded value. `NA` is an error.
`naxp`	A naxp from `naxp_parse()`, a pattern string or a naxp_encoded vector, whose naxp is used.

Details

The values are integers where the naxp's maximum encoded value is at most 2^31 - 1, and doubles where it is at most 2^53. Above 2^53 is an error. 0 stands for text that is not valid. Encoded values are never NA.

A naxp_encoded vector prints as the text it decodes to, with 0 shown as <invalid (0)>. as.character() gives the text, with NA for 0, and as.integer() or as.numeric() give the bare values.

Subsetting, rep(), rev(), unique() and sort() return naxp_encoded vectors with the same naxp, and split() returns a list of them. A naxp_encoded vector can be a data frame column. order() and match() return integer indices, and table() counts each distinct value, labelled with its text, excluding 0. Assigning into a naxp_encoded vector accepts another with the same naxp, or text, which is encoded. c() combines naxp_encoded vectors with the same naxp.

Encoded values are compared only for equality. == and != compare a naxp_encoded vector with one with the same naxp, with text, which is encoded first, or with numbers as bare values, so x == 0 finds the invalid text. Arithmetic, <, > and summaries such as max() raise errors. sort() and order() sort by encoded value.

Value

A naxp_encoded vector the same length as values, with its names.

Examples

```
postcode_naxp <- naxp_parse(r"(\9\9\9\9 \s!! \A\A)")

postcodes <- naxp_encoded(c(684114, 0, 6760000), postcode_naxp)
postcodes
postcodes == "1012 AB"
postcodes == 0

data.frame(postcode = postcodes)
```

---

naxp_parse *Parse a naxp*

Description

Parses the pattern of a naxp, checking that it is valid, and returns a naxp for the other functions in the package.

Usage

```
naxp_parse(pattern)
```

Arguments

pattern	A single string: the pattern of the naxp.

Details

A naxp holds only its pattern. The compiled naxp is kept for the rest of the session, so parsing the same pattern again is cheap, and a naxp saved with `saveRDS()` or sent to a parallel worker is compiled again when first used.

Every function that takes a naxp also accepts a pattern string, which it parses in the same way, or a naxp_encoded vector, whose naxp it uses.

Value

A naxp: an object of class `naxp` holding the pattern.

Errors

An invalid pattern raises a condition of class `naxp_error`, which is also an `error`. Its fields are named as in the other naxp libraries:

- `code`: the error code, such as `"NAXP1025"`, which is the same in every version.
- `message`: what is wrong, and where practical what to write instead.
- `offset`: where the error starts, in characters from the start of the pattern, counting from 0 as the other naxp libraries do.
- `length`: how many characters the error covers.
- `pattern`: the pattern.

Each error code is explained at https://naxp.org/error-codes/.

See Also

`naxp_encode()` to encode text, and `naxp_pattern()` for a naxp's properties.

Examples

```
postcode_naxp <- naxp_parse(r"(\9\9\9\9 \s!! \A\A)")
postcode_naxp

# An invalid pattern raises a naxp_error
tryCatch(
  naxp_parse(r"(\9\9\9\9 [A-)"),
  naxp_error = function(error) error$code
)
```

`naxp_properties` *Properties of a naxp*

Description

The pattern a naxp was parsed from, its largest encoded value and the length of its longest canonical form.

Usage

```
naxp_pattern(naxp)

naxp_max_encoded_value(naxp)

naxp_max_length(naxp)
```

Arguments

`naxp`	A naxp from `naxp_parse()`, a pattern string or a naxp_encoded vector, whose naxp is used.

Details

Encoded values run from 1 to the maximum with no gaps, so the maximum is also the number of distinct encoded values. It is an integer where it is at most 2^31 - 1 and a double where it is at most 2^53. Above 2^53, which is the largest whole number R holds exactly, `naxp_max_encoded_value()` raises an error, although printing the naxp still shows the maximum.

Value

- `naxp_pattern()`: a single string.
- `naxp_max_encoded_value()`: a single integer or double.
- `naxp_max_length()`: a single integer.

Examples

```
postcode_naxp <- naxp_parse(r"(\9\9\9\9 \s!! \A\A)")

naxp_pattern(postcode_naxp)
naxp_max_encoded_value(postcode_naxp)
naxp_max_length(postcode_naxp)

# An encoded vector stands for the naxp that encoded it
postcodes <- naxp_encode(c("1012AB", "9999 ZZ"), postcode_naxp)
naxp_pattern(postcodes)
```

naxp_standard *Standard naxps*

Description

`naxp_standard()` gives one of the standard naxps built into every naxp library, and `naxp_standards()` lists them all.

Usage

```
naxp_standard(id, variant = "default")

naxp_standards()
```

Arguments

`id`	A single string: the standard's id, such as `"uk-postcode"`.
`variant`	A single string: the variant's id. `"default"` or `""` gives the default.

Details

The standard naxps are for widely used code systems such as postcodes. Each has an id, such as `"uk-postcode"`, and one or more variants, one of which is `"default"`. The id and the variant are matched without regard to ASCII case. Once published, a standard naxp never changes, so values encoded with it keep their meaning.

The standard naxps are described at https://naxp.org/standard/.

Value

- `naxp_standard()`: a naxp.
- `naxp_standards()`: a data frame with one row per variant, each standard's default first, and character columns `id`, `name` (for display), `variant`, `pattern` and `notes` (what the variant accepts and prints, in Markdown).

Examples

```
postcodes_as_text <- c("B11AA", "B1 1AA", "B1   1AA")

default_naxp <- naxp_standard("uk-postcode")
naxp_encode(postcodes_as_text, default_naxp)

padding_tolerant_naxp <- naxp_standard("uk-postcode", "padding-tolerant")
naxp_encode(postcodes_as_text, padding_tolerant_naxp)

naxp_standards()[, c("id", "variant")]
```

Index

as.character(), *7*
as.integer(), *7*
as.numeric(), *7*

c(), *7*

match(), *7*
max(), *7*

naxp_accepts, 2
naxp_accepts(), *6*
naxp_canonical_form *(*naxp_accepts*)*, 2
naxp_canonical_form(), *6*
naxp_compare, 3
naxp_decode *(*naxp_encode*)*, 5
naxp_emit, 4
naxp_encode, 5
naxp_encode(), *6*, *8*
naxp_encoded, *2–5*, 6, *6*, *8*, *9*
naxp_first_divergent_value *(*naxp_compare*)*, 3
naxp_max_encoded_value *(*naxp_properties*)*, 9
naxp_max_length *(*naxp_properties*)*, 9
naxp_parse, 7
naxp_parse(), *2–6*, *9*
naxp_pattern *(*naxp_properties*)*, 9
naxp_pattern(), *8*
naxp_properties, 9
naxp_standard, 10
naxp_standards *(*naxp_standard*)*, 10

order(), *7*

rep(), *7*
rev(), *7*

saveRDS(), *8*
sort(), *7*
split(), *7*

table(), *7*

unique(), *7*