Package: punycoder (via r-universe)

October 30, 2026

Type Package

Title Unicode and Punycode Domain Name Processing

Version 1.3.0

Description High-performance Unicode and Punycode processing for internationalized domain names. The 'puny_encode()' / 'puny_decode()' helpers are a low-level, RFC 3492 compliant Punycode codec for domain labels (the 'xn--' ASCII-Compatible Encoding of RFC 5890/5891); they perform the raw transform plus letter-digit-hyphen checks and do not apply Unicode IDNA normalization. 'host_normalize()' is the Unicode Technical Standard #46 host-normalization entry point, mapping a host name to a canonical lowercase ASCII comparison form (non-transitional profile, pinned default Unicode version, selectable per call from the set the build ships). Aimed at host normalization and data analysis workflows. Used as the Punycode and IDNA engine by the 'pslr' and 'rurl' packages.

Depends R (>= 3.5.0)

Imports Rcpp (>= 1.0.0)

LinkingTo Rcpp

SystemRequirements GNU libidn2 (optional, for native punycode backend)

License MIT + file LICENSE

URL https://gitlab.com/bart-turczynski/punycoder, https://bart-turczynski.gitlab.io/punycoder/, https://CRAN.R-project.org/package=punycoder

BugReports https://gitlab.com/bart-turczynski/punycoder/-/issues

Encoding UTF-8

Language en-US

Suggests testthat (>= 3.0.0), knitr, rmarkdown, oysteR, rosv

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

X-schema.org-keywords punycode, punycode encode, punycode decode, IDNA, IDN, internationalized domain names, UTS-46, RFC 3492, Unicode, host normalization, hostname, domain names, TLD

NeedsCompilation yes

Author Bart Turczynski [aut, cre] (ORCID: <https://orcid.org/0000-0002-8788-7980>)

Maintainer Bart Turczynski <bartek@turczynski.pl>

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

Date/Publication 2026-09-30 15:20:02 UTC

RemoteUrl https://github.com/cran/punycoder

RemoteRef HEAD

RemoteSha bbb8e4df71dd095780800eea3d4d394d6fd1be2c

Contents

host_normalize 2
is_idn 4
is_punycode 5
normalization_profile_info 6
print.punycoder_validation 7
print.punycoder_validation_summary 8
puny_decode 8
puny_encode 9
summary.punycoder_validation 10
unicode_versions 11
validate_domain 12

Index **14**

`host_normalize` *Normalize hosts to canonical comparison form*

Description

Converts DNS hostnames to their canonical comparison form following the ratified canonical-host normalization contract: Unicode NFC, case mapping, UTS-46 label mapping and validation (non-transitional, with `UseSTD3ASCIIRules`, `CheckHyphens`, `CheckBidi`, and `CheckJoiners`), conversion to lowercase ASCII A-labels, and DNS length verification, while preserving whether the input carried a single terminal root dot.

Usage

```
host_normalize(
  x,
  check_hyphens = TRUE,
  use_std3 = TRUE,
  verify_dns_length = TRUE,
  unicode_version = NULL
)

host_normalise(
  x,
  check_hyphens = TRUE,
  use_std3 = TRUE,
  verify_dns_length = TRUE,
  unicode_version = NULL
)
```

Arguments

x	Character vector of hostnames. NA elements pass through as NA (missing, not invalid). Names are preserved.
check_hyphens	Logical scalar. When TRUE (the default) the UTS #46 CheckHyphens rule rejects "--" in the 3rd/4th positions and leading or trailing hyphens. FALSE drops that check.
use_std3	Logical scalar. When TRUE (the default) UseSTD3ASCIIRules restricts ASCII to letters, digits, and hyphen. FALSE admits other ASCII (e.g. "_") that the selected Unicode table set marks STD3-disallowed-but-valid.
verify_dns_length	Logical scalar. When TRUE (the default) each A-label must be 1-63 octets and the whole host <= 253. FALSE drops the length limits (empty labels are still rejected as structural errors).
unicode_version	Character scalar naming a Unicode table set this build ships, or NULL (the default) for the pinned one. See unicode_versions(). An unshipped version is an error.

Details

Unlike puny_encode(), invalid input is reported by returning NA_character_ (never by aborting), so a caller can layer its own policy. See normalization_profile_info() for the machine-readable identity of the profile a given call applies.

A build ships one or more Unicode table sets and pins one of them as the default; unicode_version selects among them and unicode_versions() lists what is available. Naming a version the build does not ship is an error, not a fall back to the default — a silent fallback would let a caller record a profile identity describing a normalization that never ran. The Unicode version is a parameter of UTS #46 conformance (all three conformance clauses are phrased *"Given a version of Unicode..."*), so selecting one stays conformant.

This is a **UTS #46 profile, not IDNA2008 / RFC 5891 conformance.** UTS #46 is compatibility processing and deliberately differs from IDNA2008 — it accepts labels IDNA2008 would reject (e.g. a label whose first character is the symbol U+2615 HOT BEVERAGE becomes `"xn--53h.example"`). The pipeline draws on RFC 3492 (the Punycode transform), NFC per UAX #15, the RFC 5892 ContextJ rules via `CheckJoiners` (ZWJ/ZWNJ only — full RFC 5892 CONTEXTO is **not** checked), the RFC 5893 Bidi rule via `CheckBidi`, and STD 3 (RFC 952 + RFC 1123) host-name rules via `UseSTD3ASCIIRules`. IDNA2003 / Nameprep (RFC 3490/3491/3454) is not used.

The default applies the full strict UTS #46 profile (`uts46-nontransitional-std3-v2`). The `check_hyphens`, `use_std3`, and `verify_dns_length` arguments are UTS #46 processing flags that can each be relaxed independently; pass the *same* values to `normalization_profile_info()` to obtain the identity of the resulting profile. These are standard UTS #46 parameters, **not** a browser mode: `CheckBidi` and `CheckJoiners` always apply and are never knobs, and full WHATWG host policy (where `beStrict = false` flips exactly these three) lives upstack in `rurl`, not here.

Value

A character vector the same length as `x`. Each element is the canonical lowercase ASCII A-label host, or `NA_character_` when the input is `NA` or invalid under the profile.

See Also

`normalization_profile_info()` for the profile identity, `unicode_versions()` for the table sets available, `puny_encode()` for the lower-level RFC 3492 transform.

Examples

```
host_normalize(c("Example.COM", "münchen.de", "example.com."))
host_normalize("a_b.com") # NA: STD3 rejects "_"
host_normalize("a_b.com", use_std3 = FALSE) # "a_b.com"
host_normalize("example.com", unicode_version = unicode_versions()[[1L]])
```

---

is_idn *Test if domain contains internationalized characters*

Description

Determines whether a domain name contains Unicode characters that would require punycode encoding for ASCII compatibility.

Usage

```
is_idn(x)
```

Arguments

`x`	Character vector of domain names to test

Value

A logical vector the same length as x, where TRUE indicates the element contains non-ASCII Unicode characters. Never NA and never an error: an element that is not well-formed UTF-8 is reported as FALSE, matching is_punycode and base R's own validUTF8. Use validUTF8(x) to tell "not internationalized" apart from "not well-formed text".

See Also

is_punycode for detecting punycode domains, puny_encode for encoding Unicode domains.

Examples

```
is_idn("caf\u00E9.com") # TRUE
is_idn("example.com") # FALSE
is_idn(c(
  "caf\u00E9.com",
  "\u043C\u043E\u0441\u043A\u0432\u0430.\u0440\u0444",
  "test.com"
)) # c(TRUE, TRUE, FALSE)
```

---

is_punycode *Test if string is punycode encoded*

Description

Determines whether a given string or domain name is already encoded in punycode format (starts with xn– prefix).

Usage

```
is_punycode(x)
```

Arguments

x	Character vector to test

Value

A logical vector the same length as x, where TRUE indicates the element contains a punycode-encoded label (xn– prefix). Never NA and never an error: an element that is not well-formed UTF-8 is reported as FALSE, matching is_idn and base R's own validUTF8. Use validUTF8(x) to tell "not punycode" apart from "not well-formed text".

See Also

is_idn for detecting Unicode domains, puny_decode for decoding punycode domains.

Examples

```
is_punycode("xn--example") # TRUE
is_punycode("example.com") # FALSE
is_punycode(c("xn--caf-dma.com", "regular.com")) # c(TRUE, FALSE)
```

`normalization_profile_info` *Canonical-host normalization profile identity*

Description

Returns the stable, machine-readable identity of a normalization profile. Called with no arguments it reports the default (fully strict) profile `host_normalize()` applies; the `check_hyphens`, `use_std3`, and `verify_dns_length` arguments report the identity of a specific flag set so a caller can describe the exact profile a given normalization used. Downstream packages key reproducibility on the full per-parameter column set; `profile` is a coarse cache token (distinct per flag set, but no longer load-bearing alone) and the `backend` column is diagnostic only and must never enter a reproducibility or cache key.

Usage

```
normalization_profile_info(
  check_hyphens = TRUE,
  use_std3 = TRUE,
  verify_dns_length = TRUE,
  unicode_version = NULL
)

normalisation_profile_info(
  check_hyphens = TRUE,
  use_std3 = TRUE,
  verify_dns_length = TRUE,
  unicode_version = NULL
)
```

Arguments

`check_hyphens`, `use_std3`, `verify_dns_length`
: Logical scalars selecting the flag set to report. Each defaults to `TRUE` (the strict profile).

`unicode_version`
: Character scalar naming a Unicode table set this build ships, or `NULL` (the default) for the pinned one — pass the same value given to `host_normalize()`. The `unicode_version` column reports the resolved version, so calling with no arguments still reports the pin.

Details

`check_bidi`, `check_joiners`, and `transitional` are fixed by the profile (UTS #46 non-transitional, both bidi and joiner checks always on) and are reported as constant columns rather than arguments.

Value

A one-row `data.frame` with columns `profile`, `unicode_version`, `idna`, `transitional`, `use_std3`, `check_hyphens`, `check_bidi`, `check_joiners`, `verify_dns_length`, and `backend`.

See Also

`host_normalize()`, `unicode_versions()`.

Examples

```
normalization_profile_info()
normalization_profile_info(use_std3 = FALSE)
```

`print.punycoder_validation` *Print method for punycoder validation results*

Description

Prints a count header followed by one block per domain, truncated to the first 10 elements. Error bullets carry the machine-readable error code in brackets; use `summary()` for counts by error code across the whole vector.

Usage

```
## S3 method for class 'punycoder_validation'
print(x, ...)
```

Arguments

`x`	A punycoder_validation object
`...`	Additional arguments (ignored)

Value

Invisibly returns `x`.

See Also

`summary.punycoder_validation` for the aggregate view.

Examples

```
result <- validate_domain(c("example.com", "xn--bad-label-"))
print(result)
```

`print.punycoder_validation_summary` *Print method for punycoder validation summaries*

Description

Print method for punycoder validation summaries

Usage

```
## S3 method for class 'punycoder_validation_summary'
print(x, ...)
```

Arguments

`x`	A punycoder_validation_summary object, as returned by `summary.punycoder_validation`
`...`	Additional arguments (ignored)

Value

Invisibly returns `x`.

Examples

```
print(summary(validate_domain(c("example.com", "-bad.com"))))
```

`puny_decode` *Decode ASCII Punycode to Unicode domain labels (low-level)*

Description

Converts ASCII Punycode (`xn--`) domain names back to their Unicode representation. This is the inverse of `puny_encode()` and is the raw RFC 3492 transform with A-label framing checks. DNS host length limits are intentionally not applied by this raw codec; use `validate_domain()` or `host_normalize()` when you need DNS host validation.

Usage

```
puny_decode(x, strict = getOption("punycoder.strict", TRUE))
```

Arguments

x	Character vector of ASCII punycode domains to decode
strict	Logical; whether to apply strict validation. Defaults to `getOption("punycoder.strict", TRUE)`. In strict mode the raw codec enforces structural checks but not DNS host length limits.

Details

Like `puny_encode()`, this is a **low-level ASCII-Compatible Encoding helper, not an IDNA normalization API**: it does not apply UTS #46 mapping or NFC. For IDNA/UTS-46 host normalization, see `host_normalize()`.

Value

A character vector the same length as `x`, with each element containing the Unicode-decoded domain name. Elements corresponding to `NA` inputs are `NA_character_`. In non-strict mode, domains that fail decoding are also returned as `NA_character_`.

See Also

`puny_encode` for the reverse operation, `host_normalize` for IDNA/UTS-46 host normalization.

Examples

```
# Basic decoding
puny_decode("xn--caf-dma.com")
puny_decode("xn--80adxhks.xn--p1ai")

# Vectorized decoding
ascii_domains <- c("xn--caf-dma.com", "xn--80adxhks.xn--p1ai")
puny_decode(ascii_domains)
```

---

puny_encode *Encode Unicode domain labels to ASCII Punycode (low-level)*

Description

Converts Unicode domain names to their ASCII Punycode (`xn--`) representation: the raw RFC 3492 Bootstring transform wrapped in the RFC 5890/5891 A-label framing, plus letter-digit-hyphen and leading/trailing hyphen checks per label. DNS host length limits are intentionally not applied by this raw codec; use `validate_domain()` or `host_normalize()` when you need DNS host validation.

Usage

```
puny_encode(x, strict = getOption("punycoder.strict", TRUE))
```

Arguments

x	Character vector of Unicode domain names to encode
strict	Logical; whether to apply strict validation. Defaults to getOption("punycoder.strict", TRUE). In strict mode the raw codec enforces structural checks but not DNS host length limits.

Details

This is a **low-level ASCII-Compatible Encoding helper, not an IDNA normalization API.** It does *not* apply Unicode NFC, UTS #46 mapping, case folding, or Bidi/Joiner validation. To map a host name to its canonical comparison form under a UTS #46 profile (the IDNA surface of this package), use host_normalize().

Value

A character vector the same length as x, with each element containing the ASCII punycode-encoded domain name. Elements corresponding to NA inputs are NA_character_. In non-strict mode, domains that fail encoding are also returned as NA_character_.

See Also

puny_decode for the reverse operation, host_normalize for IDNA/UTS-46 host normalization.

Examples

```
# Basic encoding
puny_encode("caf\u00E9.com")
puny_encode("\u043C\u043E\u0441\u043A\u0432\u0430.\u0440\u0444")

# Vectorized encoding
domains <- c(
  "caf\u00E9.com",
  "\u043C\u043E\u0441\u043A\u0432\u0430.\u0440\u0444",
  "\u5317\u4EAC.\u4E2D\u56FD"
)
puny_encode(domains)
```

---

summary.punycoder_validation

Summarize punycoder validation results

Description

Condenses a punycoder_validation object into counts of failures by machine-readable error code. The per-domain detail stays available on the validation object itself ($errors / $error_codes) and in print.punycoder_validation.

Usage

```
## S3 method for class 'punycoder_validation'
summary(object, ...)
```

Arguments

`object`	A punycoder_validation object
`...`	Additional arguments (ignored)

Value

A data frame of class `"punycoder_validation_summary"` with one row per distinct error code, sorted by count descending, and columns:

error_code Character; the stable machine-readable error code.

n Integer; how many domains reported that code.

Input with no errors yields a zero-row data frame with the same columns. The result carries the attributes `n` (number of domains), `n_valid`, `n_invalid`, and `strict`.

See Also

`validate_domain`, `print.punycoder_validation`.

Examples

```
result <- validate_domain(c("example.com", "-bad.com", "bad_label.com"))
summary(result)
```

`unicode_versions` *Unicode table sets available in this build*

Description

punycoder vendors its Unicode data (combining classes, decompositions, UTS #46 mapping and status, `Bidi_Class`, `Joining_Type`) as generated tables compiled into the package, and a build can carry more than one version at once. This reports the versions it carries, in registration order.

Usage

```
unicode_versions()
```

Details

Which one `host_normalize()` uses by default is the *pinned* version, and it is reported by `normalization_profile_info()$`
rather than marked here — that column is the single source of truth downstream packages key on.

Value

A character vector of Unicode version strings, e.g. `"16.0.0"`.

See Also

`host_normalize()` for selecting one, `normalization_profile_info()` for the pinned default and the rest of the profile identity.

Examples

```
unicode_versions()
normalization_profile_info()$unicode_version # the pinned default
```

`validate_domain` *Comprehensive domain name validation*

Description

Validates domain names according to RFC standards, checking for proper format, length restrictions, and character requirements. Supports both Unicode and ASCII domain names.

Usage

```
validate_domain(x, strict = getOption("punycoder.strict", TRUE))
```

Arguments

`x`	Character vector of domain names to validate
`strict`	Logical; whether to apply strict validation. Defaults to `getOption("punycoder.strict", TRUE)`.

Value

An object of class `"punycoder_validation"` (a named list) with components:

domains Character vector of the input domain names.

valid Logical vector indicating whether each domain is valid.

errors List of character vectors, each containing error messages for the corresponding domain (empty for valid domains).

error_codes List of character vectors, each containing stable machine-readable error codes for the corresponding domain (empty for valid domains). Missing input uses `"domain_na"`.

See Also

`puny_encode` for encoding validated domains.

Examples

```
validate_domain("example.com")
validate_domain("caf\u00E9.example.com")
long_label <- paste(rep("x", 250), collapse = "")
validate_domain(c("valid.com", "invalid..com", long_label))
```

Index

host_normalise *(*host_normalize*)*, 2
host_normalize, 2, *9, 10*
host_normalize(), *6–12*

is_idn, 4, *5*
is_punycode, 5, *5*

normalisation_profile_info
        *(*normalization_profile_info*)*, 6
normalization_profile_info, 6
normalization_profile_info(), *3, 4*, *12*

print.punycoder_validation, 7, *10, 11*
print.punycoder_validation_summary, 8
puny_decode, *5*, 8, *10*
puny_encode, *5*, 9, *9*, *12*
puny_encode(), *3, 4*, *8, 9*

summary(), *7*
summary.punycoder_validation, *7, 8*, 10

unicode_versions, 11
unicode_versions(), *3, 4*, *7*

validate_domain, *11*, 12
validate_domain(), *8, 9*
validUTF8, *5*