

Package ‘healthyAddress’

February 7, 2024

Title Convert Addresses to Standard Inputs

Version 0.2.0

Description Efficient tools for parsing and standardizing Australian addresses from textual data. It utilizes optimized algorithms to accurately identify and extract components of addresses, such as street names, types, and postcodes, especially for large batched data in contexts where sending addresses to internet services may be slow or inappropriate. The core functionality is built on fast string processing techniques to handle variations in address formats and abbreviations commonly found in Australian address data. Designed for data scientists, urban planners, and logistics analysts, the package facilitates the cleaning and normalization of address information, supporting better data integration and analysis in urban studies, geography, and related fields.

License GPL-2

Encoding UTF-8

RoxygenNote 7.2.0

Imports data.table, fastmatch, fst, hutils, hutilscpp, magrittr, qs, utils

Suggests tinytest

Depends R (>= 3.5.0)

NeedsCompilation yes

Author Hugh Parsonage [aut, cre]

Maintainer Hugh Parsonage <hugh.parsonage@gmail.com>

Repository CRAN

Date/Publication 2024-02-07 17:50:09 UTC

R topics documented:

.digit256	2
.permitted_street_type_ord	2
extract_flatNumberFirstLast	3
extract_postcode	3
HashStreetName	4

match_StreetType	4
match_word	5
nany_lowercase	6
postcode2ste	6
read_ste_fst	7
standardize_address	8
toupper_basic	10
unique_Postcodes	10

Index	11
--------------	-----------

<i>.digit256</i>	<i>Extract the n-th digit of a duocentehexaquingesimal number</i>
------------------	---

Description

Extract the n-th digit of a duocentehexaquingesimal number

Usage

.digit256(x, d)

Arguments

x	integer(n)
d	integer(1) One of 0:3. Other integers silently return x.

Value

For b = 256 if

$$x = a_0 + a_1b + a_2b^2 + a_3b^3$$

then *.digit(x, d)* = a_d

<i>.permitted_street_type_ord</i>	<i>Street types allowed.</i>
-----------------------------------	------------------------------

Description

Street types allowed.

Usage

.permitted_street_type_ord()

Value

A character vector, the permitted street codes. In order of (approximate) occurrence; more common street types appear in the head of the vector.

`extract_flatNumberFirstLast`

Extract the flat number, number first/last from an address

Description

Extract the flat number, number first/last from an address

Usage

```
extract_flatNumberFirstLast(address)
```

Arguments

address A character vector from which the numbers are to be extracted.

Value

A data.table of three components: the flat number, the number first, and number last.

`extract_postcode`

Extract the postcode from the suffix of a string

Description

Extract the postcode from the suffix of a string

Usage

```
extract_postcode(x)
```

Arguments

x A character vector.

Value

An integer vector the same length as x, giving the postcode as it appears in the last 3 or 4 characters in each string. Returns NA_integer_ for other strings.

There is no guarantee made that the postcode is a real postcode.

Examples

```
extract_postcode("3000")
extract_postcode("Melbourne Vic 3000")
```

HashStreetName	<i>Hash a street name quickly and accurately</i>
----------------	--

Description

Hash a street name quickly and accurately

Usage

```
HashStreetName(x)
unHashStreetName(x)
```

Arguments

x A character vector of uppercase street names (without the street type).

Value

For HashStreetName, an integer vector the same length as x, a hash of the input; for unHashStreetName the inverse operation.

If the original x does not contain a recognized street name, the result of unHashStreetName will be NA.

Examples

```
HashStreetName("FLINDERS")
```

match_StreetType	<i>Find the street type within an address</i>
------------------	---

Description

Find the street type within an address

Usage

```
match_StreetType(address)
```

Arguments

address A character vector, every string an address.

Value

A list of two elements. The first element are the indices of street type in `.permitted_street_type_ord()` that is found in the address. The second element are the corresponding string positions of the street so identified.

Examples

```

cds <- .permitted_street_type_ord()
head(cds)
match_StreetType("712 FLINDERS STREET MELBOURNE 3004")
#           012345678901234
match_StreetType("712 FLINDERS ST MELBOURNE 3004")

```

match_word

Find word within a sentence

Description

Find word within a sentence

Usage

```
match_word(x, tbl)
```

Arguments

x A character vector of uppercase sentences.
tbl A table of words. Long vectors are not permitted.

Value

An integer vector the same length as x, where the i-th entry is the integer position of the first word in tbl detected in `x[i]`. Non-matches return NA. Words are strings of uppercase separated by spaces.

many_lowercase	<i>Uppercase character vectors</i>
----------------	------------------------------------

Description

Ensures all elements of a character vector are uppercase; no lowercase characters.

Usage

```
many_lowercase(x, nThread = getOption("healthyAddress.nThread", 1L))
```

Arguments

x	A character vector, of ASCII elements.
nThread	Number of threads to use.

Value

many_lowercase FALSE if any char in x is a lowercase letter.

Examples

```
many_lowercase("ABC")
many_lowercase("ABC 123 /--")
many_lowercase("ABC 123 /-- z")
```

postcode2ste	<i>In what states do postcodes lie?</i>
--------------	---

Description

While for most postcodes, the state enclosing it is easy to evaluate (e.g. most postcodes in 2000-2999 are in NSW), the general case is non-trivial. In particular, some postcodes straddle state borders.

Usage

```
postcode2ste(Postcodes, result = c("integer", "character"))
```

Arguments

Postcodes	An integer vector of postcodes.
result	One of "integer" or "character". If "character" the abbreviated state names(s) are returned.

Value

A vector, the minimal states that cover all postcodes given. For example, if all postcodes lie within a single state a scalar integer/string of that state is returned.

Examples

```
vic_poa <- c(3021L, 3084L, 3013L, 3147L, 3030L,
            3123L, 3070L, 3004L, 3250L, 3630L)

postcode2ste(vic_poa)
postcode2ste(vic_poa, result = "character")
postcode2ste(c(vic_poa, 2000L))
postcode2ste(3644L)
```

read_ste_fst	<i>Get internal data</i>
--------------	--------------------------

Description

Get internal data

Usage

```
read_ste_fst(
  ste = c("ACT", "NSW", "NT", "OT", "QLD", "SA", "TAS", "VIC", "WA"),
  columns = NULL,
  data_env = getOption("healthyAddress.data_env"),
  data_dir = getOption("healthyAddress.data_dir", tempfile()),
  rbind = TRUE
)
```

Arguments

ste	The abbreviated state name.
columns	Character vector of columns to select. If NULL, all columns are selected.
data_env	The environment in which objects are cached. Mainly for internal use.
data_dir	The file directory into which the downloaded files should be stored. Defaults to a temporary directory. It is recommended to set the option <code>healthyAddress.data_dir</code> so that subsequent calls to this function do not result in unnecessary downloads.
rbind	Whether or not to bind the list result should multiple states be requested.

Value

A data.table containing all the addresses in the given states.

standardize_address *Standard address*

Description

Standardize an address from a free text expression into its components as used in the PSMA (formerly, "Public Sector for Mapping Agencies") database.

Usage

```
standardize_address(
  Address,
  AddressLine2 = NULL,
  return.type = c("data.table", "integer"),
  integer_StreetType = FALSE,
  hash_StreetName = FALSE,
  check = 1L,
  nThread = getOption("healthyAddress.nThread", 1L)
)

standard_address2(Address, nThread = getOption("healthyAddress.nThread", 1L))

standard_address3(Line1, Line2, Postcode = NULL, KeepStreetName = FALSE)
```

Arguments

Address	A character vector, either a full address or (if AddressLine2 is not NULL) the first line of an Australian address.
AddressLine2	Either NULL (the default) or a character vector, the same length as Address giving the second line of the Address.
return.type	Either "data.table" or "integer". "data.table" implies a table of columns separating the address components. "integer" means an integer vector creating a bijection between the address and the PSMA internal id.
integer_StreetType	Should the street type be returned as an integer vector?
hash_StreetName	Should STREET_NAME be returned as an integer hash, as in HashStreetName?
check	An integer, whether the inputs should be checked for possibly invalid addresses or addresses that may not be parsed correctly.
nThread	Number of threads to use.
Line1, Line2, Postcode	For addresses split by line. Line1 is assumed to end with the street type. The second line is only used to determine Postcode, and then only if it is NULL, the default.
KeepStreetName	Should an additional character vector be included in the result of the street name?

Details

By convention observed in the PSMA, street names such as 'THE ESPLANADE' have a street name of 'THE ESPLANADE' and an absent street type code.

Non-addresses passed have unspecified behaviour, though usually the numbers of the standard address will be 0 or NA. Postcodes may be negative in some circumstances where a postcode is not detected, though this should not be relied on.

For maximum performance, consider setting `integer_StreetType` and `hash_StreetName` to `TRUE`. It has been observed that joining two tables together has been faster when using the hash of the standardized street name, rather than the street name, even when taking into account the hashing process.

For performance reasons, addresses with more than 32 words are not supported.

If a postcode-like number exists at the end of a Address, but is not in fact a postcode, then NA will be in each field, except postcode, which will have the value -1.

Value

A data.table containing columns indicating the components of the standard address:

`FLAT_NUMBER` The flat or unit number. This includes things like SHOP number.

`NUMBER_FIRST` As used in the PSMA, this identified the first (or only) number in the address range.

`NUMBER_LAST` As used in the PSMA, if an address is marked as having a range of street numbers, the last of the range.

`NUMBER_SUFFIX` A raw vector. The suffix observed after the numbers. The PSMA technically has multiple suffixes for each number component.

`H0` If `hash_StreetName = TRUE`, the DJB2 hash (as used in [HashStreetName](#) of the street name.). Observed to have performance benefits.

`STREET_NAME` The (uppercase) of the street name. Streets such as 'THE ESPLANADE' or 'THE AVENUE' are treated as entirely made up of a street name and have a `STREET_TYPE_CODE` of zero.

`STREET_TYPE_CODE` An integer, the street type code marking the type of street such as ROAD, STREET, AVENUE, etc. They code corresponds approximately to the rank of their frequency in addresses.

`STREET_TYPE` If `integer_StreetType = FALSE`, then the (uppercase) standard name of the street type.

`POSTCODE` An integer vector, the postcode observed.

toupper_basic	<i>Uppercase</i>
---------------	------------------

Description

Uppercase

Usage

`toupper_basic(x)`

Arguments

`x` A character vector

Value

The same as `toupper(x)` for ASCII entries. For implementation reasons, strings wider than 32767 characters (bytes) will be ignored.

unique_Postcodes	<i>Unique postcodes of</i>
------------------	----------------------------

Description

Unique postcodes of

Usage

`unique_Postcodes(x)`

Arguments

`x` An integer vector of postcodes.

Value

A (sorted) integer vector of the unique, non-NA values in `x`.

Index

.digit256, 2
.permitted_street_type_ord, 2
extract_flatNumberFirstLast, 3
extract_postcode, 3
HashStreetName, 4, 8, 9
match_StreetType, 4
match_word, 5
nany_lowercase, 6
postcode2ste, 6
read_ste_fst, 7
standard_address2
 (standardize_address), 8
standard_address3
 (standardize_address), 8
standardize_address, 8
toupper_basic, 10
unHashStreetName (HashStreetName), 4
unique_Postcodes, 10