

# Package ‘healthyAddress’

April 28, 2024

**Title** Convert Addresses to Standard Inputs

**Version** 0.4.1

**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

**URL** <https://github.com/HughParsonage/healthyAddress>

**BugReports** <https://github.com/HughParsonage/healthyAddress/issues>

**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](mailto:hugh.parsonage@gmail.com)>

**Repository** CRAN

**Date/Publication** 2024-04-27 23:40:01 UTC

## R topics documented:

.digit256 . . . . .	2
.permitted_street_type_ord . . . . .	3

compress_latlon . . . . .	3
download_latlon_data . . . . .	4
extract_flatNumberFirstLast . . . . .	5
extract_postcode . . . . .	5
HashStreetName . . . . .	6
match_StreetType . . . . .	6
match_word . . . . .	7
mutate_latlon . . . . .	7
nany_lowercase . . . . .	8
postcode2ste . . . . .	9
read_ste_fst . . . . .	9
standardize_address . . . . .	10
toupper_basic . . . . .	12
unique_Postcodes . . . . .	13
<b>Index</b>	<b>14</b>

---

.digit256

*Extract the n-th digit of a duocenthexaquingesimal number*

---

## Description

Extract the n-th digit of a duocenthexaquingesimal 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`

---

.permitted\_street\_type\_ord  
*Street types allowed.*

---

### 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.

---

compress\_latlon      *Compress latitude and longitude to a 32-bit integer*

---

### Description

Although lat and lon are represented by doubles, this is usually slightly wasteful. This function allows you to represent coordinates as single integer, vastly reducing memory footprint.

### Usage

```
compress_latlon(lat, lon, nThread = getOption("healthyAddress.nThread", 1L))
```

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

```
compress_latlon_general(  
  lat,  
  lon,  
  nThread = getOption("healthyAddress.nThread", 1L)  
)
```

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

### Arguments

lat, lon	Coordinates to compress.
nThread	Number of threads to use.
x	An integer vector formed by one of the compression functions.

**Value**

The `_general` version of the compression/decompression use the observed range of the latitude and longitude to form a  $2^{16}$  grid, while the bare versions use the known limits of Australian address coordinates (including the overseas territories). Since, in the latter, the grid will be much less fine, you should expect greater loss of information, possibly exceeding 100 metres.

`compress_latlon` An integer vector.

`decompress_latlon` The original `lat`, `lon`, with some information loss

`compress_latlon_general` An integer vector, with attributes `minmaxLat` and `minmaxLon`.

`decompress_latlon_general` The original `lat`, `lon`, with some information loss.

---

`download_latlon_data` *Download latitude longitude data by address*

---

**Description**

Download latitude longitude data by address

**Usage**

```
download_latlon_data(
  .ste = c("NSW", "VIC", "QLD", "SA", "WA", "TAS", "NT", "ACT", "OT"),
  data_dir = getOption("healthyAddress.data_dir"),
  repo = "https://github.com/HughParsonage/PSMA-202311",
  overwrite = NA
)
```

**Arguments**

<code>.ste</code>	The jurisdiction to download. Default is to download all.
<code>data_dir</code>	The directory for <code>healthyAddress</code> . Data will be downloaded into a subdirector <code>latlon</code> thereof.
<code>repo</code>	The repository from which data will be downloaded. Currently only the default is supported.
<code>overwrite</code>	<code>logical(1)</code> Applicable only if the file already exists prior to invoking the function. If <code>FALSE</code> , an error is raised. If <code>NA</code> , the default, the file is returned, with a message. Set to <code>TRUE</code> if you wish to overwrite the files (possibly having changed repo to reflect updated data).

**Value**

Called for its side effect (downloading the files), but returns the files downloaded.

---

`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	<i>Find word within a sentence</i>
------------	------------------------------------

---

**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.

---

mutate_latlon	<i>Add latitude and longitude columns to a standard address</i>
---------------	---

---

**Description**

Add latitude and longitude columns to a standard address

**Usage**

```
mutate_latlon(DT, data_dir = getOption("healthyAddress.data_dir"))
```

**Arguments**

DT	A data.table from standardize_address
data_dir	The directory in which the latitude longitude data has been downloaded. (See <a href="#">download_latlon_data</a> .)

**Value**

DT with the columns lat and lon added, by reference, the latitude and longitude of the address for that row.

---

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`*In what states do postcodes lie?*

---

**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**

<code>Postcodes</code>	An integer vector of postcodes.
<code>result</code>	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`*Get internal data*

---

**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**

<code>ste</code>	The abbreviated state name.
<code>columns</code>	Character vector of columns to select. If <code>NULL</code> , all columns are selected.
<code>data_env</code>	The environment in which objects are cached. Mainly for internal use.
<code>data_dir</code>	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.
<code>rbind</code>	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 <a href="#">HashStreetName?</a>
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

*Uppercase*


---

### 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, strict = TRUE)
```

```
uniqueN_Postcodes(x, strict = TRUE)
```

**Arguments**

x	An integer vector of postcodes.
strict	(logical, default: TRUE) If TRUE, only postcodes (at time of package development) with actual addresses are returned. Otherwise, any postcode in the range 1:8192 are returned.

**Value**

unique_Postcodes	A (sorted) integer vector of the unique, non-NA values in x.
uniqueN_Postcodes	The number of unique postcodes.

# Index

.digit256, [2](#)  
.permitted\_street\_type\_ord, [3](#)

compress\_latlon, [3](#)  
compress\_latlon\_general  
    (compress\_latlon), [3](#)

decompress\_latlon (compress\_latlon), [3](#)  
decompress\_latlon\_general  
    (compress\_latlon), [3](#)  
download\_latlon\_data, [4](#), [8](#)

extract\_flatNumberFirstLast, [5](#)  
extract\_postcode, [5](#)

HashStreetName, [6](#), [11](#), [12](#)

match\_StreetType, [6](#)  
match\_word, [7](#)  
mutate\_latlon, [7](#)

nany\_lowercase, [8](#)

postcode2ste, [9](#)

read\_ste\_fst, [9](#)

standard\_address2  
    (standardize\_address), [10](#)  
standard\_address3  
    (standardize\_address), [10](#)  
standardize\_address, [10](#)

toupper\_basic, [12](#)

unHashStreetName (HashStreetName), [6](#)  
unique\_Postcodes, [13](#)  
uniqueN\_Postcodes (unique\_Postcodes), [13](#)