Package: locations (via r-universe)

November 13, 2026

Type Package

Title Location Manager for the JHEEM2 Project

Version 0.5.0.9000

Config/locations/data-version 2026-08-28.3

Description Manages the locations used in the JHEEM2 US HIV model.
Includes geospatial information (latitude, longitude, and
polygon information) and plotting functionality.

License What license is it under?

URL https://github.com/tfojo1/locations

BugReports https://github.com/tfojo1/locations/issues

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 3.5)

Imports ggplot2, R6

Suggests ggmap, testthat (>= 3.0.0)

Config/testthat/edition 3

Repository https://cipher-epi.r-universe.dev

Date/Publication 2026-09-06 15:09:44 UTC

RemoteUrl https://github.com/tfojo1/locations

RemoteRef HEAD

RemoteSha 2481fc440cf1d981bb1005dd903708a88a528d13

Contents

crosswalk_locations 2
get.all.for.type 3
get.cbsa.for.msa.name 4
get.code.by.alias 4
get.contained.locations 5

get.containing.locations . . . 6
get.location.code . . . 7
get.location.code.if.unique . . . 7
get.location.coords . . . 8
get.location.name . . . 8
get.location.name.alias . . . 9
get.location.type . . . 9
get.location.types . . . 10
get.overlapping.locations . . . 10
get.polygons.for.type . . . 11
get.prefix.for.type . . . 12
get_location_history . . . 12
get_locations . . . 13
is.location.valid . . . 14
location.plot . . . 14
location.type.comprises . . . 16
locations.data.version . . . 16
locations_default_date . . . 17
register.code.aliases . . . 17
register.lat.and.long . . . 18
register.locations . . . 18
register.name.aliases . . . 19
register.relationship.between.types . . . 19
register.sub.and.super.locations . . . 20
register.types . . . 20
resolve_location . . . 21
sanitize . . . 22

Index **23**

`crosswalk_locations` *Crosswalk Temporal Locations*

Description

Return explicit, directional crosswalk edges from one historic location to target versions valid at a requested date. A code, durable location ID, or location-version ID may identify 'from', but it must resolve to at most one source version with crosswalk edges.

Usage

```
crosswalk_locations(
  from,
  to_as_of = locations_default_date(),
  measure = "none"
)
```

Arguments

`from`	One official code, durable ‘location_id‘, or explicit ‘location_version_id‘.
`to_as_of`	One target [Date] or ‘YYYY-MM-DD‘ character value.
`measure`	One of ‘"none"‘, ‘"land_area"‘, ‘"water_area"‘, or ‘"population"‘. Unavailable measures produce an error; area is never substituted for population.

Value

A data frame with one row per target edge. ‘fraction_of_from‘ is the directional allocation fraction for the requested measure; ‘fraction_of_to‘ describes the fraction of the target footprint represented by the same overlap. With ‘measure = "none"‘, measurement columns are ‘NA‘ or blank. No succession or reverse crosswalk is inferred.

Stable result columns

‘crosswalk_id‘, ‘from_location_id‘, ‘from_location_version_id‘, ‘from_code‘, ‘from_name‘, ‘to_location_id‘, ‘to_location_version_id‘, ‘to_code‘, ‘to_name‘, ‘to_as_of‘, ‘relation_kind‘, ‘coverage‘, ‘measure_type‘, ‘numerator‘, ‘denominator‘, ‘fraction_of_from‘, ‘fraction_of_to‘, ‘reference_date‘, ‘population_universe‘, ‘method‘, ‘edge_source_id‘, ‘edge_source_url‘, ‘measure_source_id‘, and ‘measure_source_url‘. ‘denominator‘ is the source-footprint total for the requested measure.

See Also

Other temporal location API: `get_location_history()`, `get_locations()`, `locations_default_date()`, `resolve_location()`

`get.all.for.type` *get.all.for.type*

Description

Returns a vector of all valid location codes of a certain type

Usage

```
get.all.for.type(type)
```

Arguments

`type`	A string type name

Value

if ’type’ is a registered type, it will return a vector of all location codes for that type. If ’type’ is not registered, it will return NA

`get.cbsa.for.msa.name` *get.cbsa.for.msa.name*

Description

We are reading data in from .csv files, and these files do not have proper cbsa codes; instead the are indexed by cbsa name. We need to take a vector of msa names and return a named vector of cbsa location codes.

Usage

```
get.cbsa.for.msa.name(names)
```

Arguments

`names`	A vector of msa names to convert into cbsa location codes

Value

A named vector of string location codes, with NAs where no result was found

`get.code.by.alias` *get.code.by.alias*

Description

Get a location code by its code alias and type

Usage

```
get.code.by.alias(locations, types)
```

Arguments

`locations`	A character vector of location code aliases
`types`	A single type or a vector of types the same length as locations

Value

A named list of canonical location codes, or 'NA' for aliases that are not registered for the requested type.

`get.contained.locations` *get.contained.locations*

Description

Get Locations that Fall Within or Overlap a Location

Usage

```
get.contained.locations(
  locations,
  sub.type,
  return.list = F,
  throw.error.if.unregistered.type = T,
  include.partial = F
)
```

Arguments

`locations`	A character vector of location codes
`sub.type`	The type (geographic resolution) of locations requested for the sub-locations
`return.list`	A single logical value indicating whether the return value should be a list with one element for each location, or whether all sub-locations should be 'unlisted' into a vector
`throw.error.if.unregistered.type`	A single logical value indicating whether the function should throw an error if sub.type has not been registered as a location type
`include.partial`	A single logical value indicating whether locations that are only partially contained should also be returned. The default is FALSE to preserve complete-containment semantics.

Value

If return.list==T, a list with length(locations) and names=locations. Each element is itself a character vector with zero or more locations corresponding to sub-locations. If return.list=F, returns a character vector (arbitrary length) containing all sub-locations that fall within ANY of the given locations. When include.partial is TRUE, directly registered partial overlaps are included.

`get.containing.locations` *get.containing.locations*

Description

Get Locations that Enclose a Location

Usage

```
get.containing.locations(
  locations,
  super.type,
  return.list = F,
  throw.error.if.unregistered.type = T,
  include.partial = F
)
```

Arguments

`locations`	A character vector of location codes
`super.type`	The type (geographic resolution) of locations requested for the super-locations
`return.list`	A single logical value indicating whether the return value should be a list with one element for each location, or whether all super-locations should be 'unlisted' into a vector
`throw.error.if.unregistered.type`	A single logical value indicating whether the function should throw an error if super.type has not been registered as a location type
`include.partial`	A single logical value indicating whether locations that only partially contain the input should also be returned. The default is FALSE to preserve complete-containment semantics.

Value

If return.list==T, a list with length(locations) and names=locations. Each element is itself a character vector with zero or more locations corresponding to super-locations. If return.list=F, returns a character vector (arbitrary length) containing all super-locations that contain ANY of the given locations. When include.partial is TRUE, directly registered partial overlaps are included.

`get.location.code` *get.location.code*

Description

Get the location code for a name and a type

Usage

```
get.location.code(location.names, types, search.aliases = TRUE)
```

Arguments

`location.names`	A list of names to get the location code for
`types`	A corresponding list of types
`search.aliases`	A boolean flag indicating whether we want to check the aliases if the name check fails. Default = TRUE

Value

A list of location.names, with values corresponding to the identified location code. Multiple results per location.names is possible. If if the location.name is not registered or NA, NA is returned

---

`get.location.code.if.unique` *get.location.code.if.unique*

Description

Get the location code for a name and a type. The difference between this function and get.location.code is that get.location.code returns a list with multiple entries, and this function returns a named vector of results only if the location name matches one location uniquely. This function uses get.location.code and converts the output.

Usage

```
get.location.code.if.unique(location.names, types, search.aliases = TRUE)
```

Arguments

`location.names`	A list of names to get the location code for
`types`	A corresponding list of types (or one type to be applied to each)
`search.aliases`	A boolean flag indicating whether we want to check the aliases if the name check fails. Default = TRUE

Value

A named vector of location.codes, with names corresponding to the location names. If there are multiple results or no results, the result in the vector will be NA

---

get.location.coords *get.location.coords*

Description

Get the latitude and longitude of a location, provided it is known

Usage

```
get.location.coords(locations)
```

Arguments

locations	A character vector of location codes

Value

A character vector of string location coordinates, separated by commas (latitude first, longitude second), with length(locations) and names=locations. If location codes are not registered (or if they were NA), the corresponding returned name is NA

---

get.location.name *get.location.name*

Description

Get the Name of a Location

Usage

```
get.location.name(locations)
```

Arguments

locations	A character vector of location codes

Value

A character vector of location names, with length(locations) and names=locations. If location codes are not registered (or if they were NA), the corresponding returned name is NA

`get.location.name.alias` *get.location.name.alias*

Description

Get an Name alias Associated with a Location

Usage

```
get.location.name.alias(
  locations,
  alias.name,
  throw.error.if.unregistered.alias = T
)
```

Arguments

`locations`	A character vector of location codes
`alias.name`	A single character value representing a previously registered name alias
`throw.error.if.unregistered.alias`	A single logical value indicating whether the function should throw an error if no alias with name alias.name has been registered

Value

A character vector of aliases, with length(locations) and names=locations. If location codes are not registered (or if they were NA), or if no value for the alias.name has been registered to a location, the corresponding returned alias is NA

`get.location.type` *get.location.type*

Description

Get the Type (Geographic Resolution) of a Location

Usage

```
get.location.type(locations)
```

Arguments

`locations`	A character vector of location codes

Value

A character vector of location types, with length(locations) and names=locations. If location codes are not registered (or if they were NA), the corresponding returned type is NA

`get.location.types` *get.location.types*

Description

Get a vector of type names, or a list of registered location types, prefixes and longform prefix names.

Usage

```
get.location.types(simple = TRUE)
```

Arguments

`simple`	A boolean; TRUE if asking for simple list of types, FALSE if you want a list of types and prefixes. Defaults to TRUE

Value

If simple=TRUE, a simple vector of named types; else a list the same length as the registered number of types, with each list item being the name, the prefix and the longform prefix name. If there are no registered types it will return an empty list

---

`get.overlapping.locations` *get.overlapping.locations*

Description

Get all locations that overlap a Location

Usage

```
get.overlapping.locations(
  locations,
  type,
  return.list = F,
  throw.error.if.unregistered.type = T
)
```

Arguments

locations	A character vector of location codes
type	The type (geographic resolution) of locations requested for the sub-locations
return.list	A single logical value indicating whether the return value should be a list with one element for each location, or whether all sub-locations should be 'unlisted' into a vector
throw.error.if.unregistered.type	A single logical value indicating whether the function should throw an error if sub.type has not been registered as a location type

Value

If return.list==T, a list with length(locations) and names=locations. Each element is itself a character vector with zero or more locations corresponding to sub-locations. If return.list=F, returns a character vector (arbitrary length) containing all sub-locations that fall within ANY of the given locations

---

get.polygons.for.type *get.polygons.for.type*

Description

Return the polygon data for a given location type

Usage

```
get.polygons.for.type(type)
```

Arguments

type	A single location type

Value

A data.frame containing all the polygon data for a given type, NA if it doesn't exist

get.prefix.for.type *get.prefix.for.type*

Description

Get the prefix for a given type

Usage

```
get.prefix.for.type(location.types)
```

Arguments

location.types	A character vector of location types

Value

A character vector of prefixes, with length(location.types) and names=prefixes. If the types are not registered (or if they were NA), the corresponding returned type is NA

get_location_history *Inspect a Location's Temporal History*

Description

Return every version and overlapping code assignment for one or more durable location IDs. Historic codes remain historic records; succession and crosswalk edges are not traversed implicitly.

Usage

```
get_location_history(location_id)
```

Arguments

location_id	One or more durable IDs returned by [get_locations()] or [resolve_location()].

Value

A data frame with the same stable columns as [get_locations()], ordered chronologically within each requested durable ID. Status is classified relative to [locations_default_date()].

See Also

Other temporal location API: `crosswalk_locations()`, `get_locations()`, `locations_default_date()`, `resolve_location()`

`get_locations` *Query Temporal Locations*

Description

Return location versions of one type classified relative to an explicit date. This is an additive temporal API and does not use the mutable legacy location manager.

Usage

```
get_locations(type, as_of = locations_default_date(), status = "current")
```

Arguments

`type`	One location type. Matching is case-insensitive. The first temporal slice supports ‘"COUNTY"‘.
`as_of`	One [Date] or ‘YYYY-MM-DD‘ character value.
`status`	One of ‘"current"‘, ‘"historical"‘, ‘"future"‘, or ‘"all"‘.

Value

A data frame with one row per overlapping location-version and code assignment. Durable IDs, version IDs, code and version validity intervals, status reference date, and source metadata are always explicit.

Stable result columns

‘location_id‘, ‘location_version_id‘, ‘location_code_id‘, ‘type‘, ‘entity_kind‘, ‘status‘, ‘status_as_of‘, ‘preferred_name‘, ‘code_system_id‘, ‘code‘, ‘version_valid_from‘, ‘version_valid_from_precision‘, ‘version_valid_to‘, ‘version_valid_to_precision‘, ‘end_reason‘, ‘code_valid_from‘, ‘code_valid_from_precision‘, ‘code_valid_to‘, ‘code_valid_to_precision‘, ‘version_source_id‘, ‘version_source_vintage‘, ‘version_source_url‘, ‘code_source_id‘, ‘code_source_vintage‘, and ‘code_source_url‘. Validity intervals are half-open: the start is included and the end is excluded. Blank bounds mean the source does not identify that bound, not that the package inferred a date.

See Also

Other temporal location API: `crosswalk_locations()`, `get_location_history()`, `locations_default_date()`, `resolve_location()`

is.location.valid *is.location.valid*

Description

Check to see if the passed-in value matches any location code or alias

Usage

```
is.location.valid(locations, suggest.options = F)
```

Arguments

locations	A character vector representing potential locations
suggest.options	A boolean indicating whether to check the aliases for potential matches

Value

A vector of boolean values whether the passed-in value is a location codes. If false, this function will display a list of possibilities.

location.plot *location.plot*

Description

Create a plot of various points in the US from a data frame

Usage

```
location.plot(
  data,
  color,
  fill,
  size = NA,
  title = NA,
  bb = c(left = -125, bottom = 24, right = -66, top = 50),
  bb.edge = 0.1,
  size.range = c(1, 5),
  color.range = c("blue", "red"),
  pch = 19,
  size.label = "",
  color.label = "",
  alpha = 1,
  map_water_color = "#C0C0C0",
  stadia_api_key = Sys.getenv("STADIA_MAPS_API_KEY")
)
```

Arguments

data	A data frame with a column labeled 'location', containing a location code, and columns to be mapped by color or size.
color	The name of a column in the data frame to be used in the aesthetic as the color of the location. If used with polygons, this is the outline color.
fill	The name of a column in the data frame to be used as the fill color for points or polygons
size	If left blank, the data will be plotted as polygons, and those locations without polygons will be skipped. If filled in, it represents the name of a column in the data frame to be used to scale the size of the point, and only those locations with latitude and longitude values will be plotted.
title	Defaults to blank; a character string representing the title on the graph.
bb	The bounding box value; Can take one of two values: either the character string "AUTO" which will use the dimensions of the locations to automatically bound the graph (using the bb.edge parameter, see below, as an edge buffer), or a named character vector of the format c(left=-125,bottom=24,right=-66, top=50), outlining the dimensions of the bounding box in latitude and longitude. Defaults to the continental US.
bb.edge	If param 'bb' is 'AUTO', this will be used to create a buffer space around the outside of the automatically generated bounding box, as a proportion of the size of the plotted locations. Defaults to 0.1, or 10% of the bounding box.
size.range	If size is used, this will scale the values in the size column between the two size values specified. Defaults to c(1,5).
color.range	The color ranges to span for the color and fill parameters. Defaults to c("blue","red"). Can also include three values (for gradient2) or more (for gradientn).
pch	The shape of the point when plotting by points. Defaults to 19, a circle.
size.label	A label to give to the size in the legend. Defaults to blank.
color.label	A label to give to the color/fill in the legend. Defaults to blank.
alpha	An alpha blending/transparency value. Between 0 and 1. Defaults to 1 (no transparency).
map_water_color	A hex color value for the color of any water on the map. Defaults to a light gray.
stadia_api_key	The API key to access the Stadia/Stamen toner map tiles. Defaults to the STADIA_MAPS_API_KEY environment variable.

Value

Returns the ggplot2 object/plot

location.type.comprises *location.type.comprises*

Description

This function allows us to understand the relationship between two different location types. Some location types contain other location types completely (as states contain counties, for instance), while some are not comparable or disjoint in other ways (states and CBSAs). This function takes two location types (super.location.type and sub.location.type), and returns TRUE or FALSE depending on whether the types overlap perfectly or not.

Usage

```
location.type.comprises(super.location.type, sub.location.type)
```

Arguments

super.location.type	The location type into which we are attempting to arrange another sub-type
sub.location.type	The location type we are trying to fit inside the super-type

Value

A Boolean TRUE or FALSE, representing whether members of the sub.location.type fit perfectly in the super.location.type

---

locations.data.version *Location Data Version*

Description

Return the version of the packaged location-data bundle. This version is independent of the package version because data corrections and source refreshes do not necessarily change the R API.

Usage

```
locations.data.version()
```

Details

The value identifies the complete bundled dataset; it is not a claim that every source in the bundle has the same geographic vintage. Consult ‘data-raw/SOURCES.csv‘ in the source repository for source-level vintages.

Value

A single character value containing the location-data version.

`locations_default_date` *Default Temporal Location Date*

Description

Return the pinned reference date used by the bundled temporal location dataset. The value changes only with an explicit data release; it is not the current system date.

Usage

```
locations_default_date()
```

Value

One [Date] value.

See Also

Other temporal location API: `crosswalk_locations()`, `get_location_history()`, `get_locations()`, `resolve_location()`

`register.code.aliases` *register.code.aliases*

Description

Register location code aliases for specific location

Usage

```
register.code.aliases(location = NA, location.aliases = NA)
```

Arguments

`location`	A single, previously registered location code or a registered location code alias.
`location.aliases`	A character vector of location code aliases for this location name. Code aliases are unique by type

register.lat.and.long *register.lat.and.long*

Description

Register a valid latitude and longitude for a location code

Usage

```
register.lat.and.long(location = NA, lat = NA, long = NA)
```

Arguments

location	A previously registered location code, or a registered location code alias.
lat	A single numeric value between -90 and 90
long	A single numeric value between -180 and 180

register.locations *register.locations*

Description

Register information about locations.

Usage

```
register.locations(type, locations, location.names)
```

Arguments

type	The geographic resolution at which to register the locations. Can be be either a single character value, or a vector of the same length as locations
locations	A character vector of location codes
location.names	A character vector of the same length as locations with corresponding names

Details

There is no error checking here; we assume that multiple locations with the same name are possible at different resolutions/types.

register.name.aliases *register.name.aliases*

Description

Register name aliases for specific location

Usage

```
register.name.aliases(
  location = NA,
  location.aliases = NA,
  location.aliases.names = NA
)
```

Arguments

location	A single, previously registered location code or a registered location code alias.
location.aliases	A character vector of aliases for this location name
location.aliases.names	A character vector names for the particular alias 'short','no spaces', 'full', etc

Details

There is no error checking here; we assume that multiple locations with the same name are possible at different resolutions/types.

register.relationship.between.types *register.relationships.between.types*

Description

Register a relationship between location types; by default the types are independent; setting the value to TRUE indicates that the super.type contains the sub.type completely

Usage

```
register.relationship.between.types(super.type, sub.type, value)
```

Arguments

super.type	A character vector of super types
sub.type	A character vector of sub types of the same length
value	A vector of boolean values indicating the whether the sub.type is contained completely in the super type, or not

register.sub.and.super.locations *register.sub.and.super.locations*

Description

Register hierarchical sub-super relationships

Usage

```
register.sub.and.super.locations(
  sub.locations,
  super.locations,
  super.completely.encloses.sub
)
```

Arguments

sub.locations	A character vector of locations codes/location code aliases
super.locations	A character vector of location codes/location code aliases of the same length as sub.locations, with corresponding super.locations
super.completely.encloses.sub	Either a single logical value or a vector the same length as sub.locations and super.locations, indicating whether the location is completely enclosed

Details

Where super.completely.encloses.sub==T, the function will automatically recognize that locations completely enclosed within the given sub.locations are also completely enclosed within the corresponding super.locations, and, conversely that locations which completely enclose the given super.locations also completely enclose the corresponding sub.locations

register.types *register.types*

Description

Register location type, prefix, and prefix.longform

Usage

```
register.types(type, prefix, prefix.longform)
```

Arguments

type	A character vector representing types to be added
prefix	A character vector of unique prefixes for the location codes for types of this kind
prefix.longform	A character vector of longform names for that particular unique prefixes

Details

The prefix is restricted to letters, numbers, period and '-'.

---

resolve_location *Resolve a Temporal Location Code*

Description

Resolve official code-history rows without using legacy aliases or silently canonicalizing a historic code to a current geography. With no 'as_of' value, all matching history is returned. With 'as_of', both the code and location version must be valid on that date.

Usage

```
resolve_location(code, code_system = NULL, as_of = NULL)
```

Arguments

code	One or more official location codes.
code_system	Optional code-system ID, such as '"census_county_geoid"'.
as_of	Optional one [Date] or 'YYYY-MM-DD' character value.

Value

A data frame with the same stable columns as [get_locations()]. An unmatched code returns a zero-row data frame with those columns. When 'as_of' is omitted, 'status' is classified relative to [locations_default_date()] while all matching history remains present.

Code resolution

Matching uses the normalized official code table only. It does not consult the legacy manager's compatibility aliases. More than one row is expected when a code was reused by different entities or spanned multiple versions.

See Also

Other temporal location API: `crosswalk_locations()`, `get_location_history()`, `get_locations()`, `locations_default_date()`

sanitize *sanitize*

Description

There are values that we will accept as location codes (eg. 'c.12580' will work for Baltimore, but Baltimore's actual code is 'C.12580'). This function converts codes we will accept into codes that are correct. The function should fail on unrecognized location code.

Usage

```
sanitize(codes)
```

Arguments

codes	A character vector of location codes, or acceptable location codes

Value

A named character vector of location codes

Index

∗ **temporal location API**
    crosswalk_locations, 2
    get_location_history, 12
    get_locations, 13
    locations_default_date, 17
    resolve_location, 21

crosswalk_locations, 2, *12, 13*, *17*, *21*

get.all.for.type, 3
get.cbsa.for.msa.name, 4
get.code.by.alias, 4
get.contained.locations, 5
get.containing.locations, 6
get.location.code, 7
get.location.code.if.unique, 7
get.location.coords, 8
get.location.name, 8
get.location.name.alias, 9
get.location.type, 9
get.location.types, 10
get.overlapping.locations, 10
get.polygons.for.type, 11
get.prefix.for.type, 12
get_location_history, *3*, 12, *13*, *17*, *21*
get_locations, *3*, *12*, 13, *17*, *21*

is.location.valid, 14

location.plot, 14
location.type.comprises, 16
locations.data.version, 16
locations_default_date, *3*, *12, 13*, 17, *21*

register.code.aliases, 17
register.lat.and.long, 18
register.locations, 18
register.name.aliases, 19
register.relationship.between.types, 19
register.sub.and.super.locations, 20
register.types, 20
resolve_location, *3*, *12, 13*, *17*, 21

sanitize, 22