Package 'arcgisgeocode'

May 7, 2026

Title A Robust Interface to ArcGIS 'Geocoding Services'

Version 0.4.0

Description A very fast and robust interface to ArcGIS 'Geocoding Services'. Provides capabilities for reverse geocoding, finding address candidates, character-by-character search autosuggestion, and batch geocoding. The public 'ArcGIS World Geocoder' is accessible for free use via 'arcgisgeocode' for all services except batch geocoding. 'arcgisgeocode' also integrates with 'arcgisutils' to provide access to custom locators or private 'ArcGIS World Geocoder' hosted on 'ArcGIS Enterprise'. Learn more in the 'Geocode service' API reference <https://developers.arcgis.com/rest/geocode/api-reference/overview-world-geocoding-service.htm>.

License Apache License (>= 2)

URL https://github.com/r-arcgis/arcgisgeocode, https://developers.arcgis.com/r-bridge/api-reference/arcgisgeocode

Imports arcgisutils (>= 0.3.3.9000), cli, httr2 (>= 1.1.1), jsonify, RcppSimdJson (>= 0.1.13), rlang (>= 1.1.0), sf

Suggests data.table, dplyr, testthat (>= 3.0.0)

Config/rextendr/version 0.3.1.9001

Config/testthat/edition 3

Encoding UTF-8

Language en

RoxygenNote 7.3.2

SystemRequirements Cargo (Rust's package manager), rustc >= 1.67, xz

Depends R (>= 4.2)

LazyData true

NeedsCompilation yes

Author Josiah Parry [aut, cre] (ORCID: <https://orcid.org/0000-0001-9910-865X>)

Maintainer Josiah Parry <josiah.parry@gmail.com>

Repository CRAN

Date/Publication 2025-10-07 05:10:02 UTC

Contents

`esri_wkids` *Esri well-known IDs*

Description

An integer vector containing the WKIDs of Esri authority spatial references. Esri WKIDs were identified from the `{arcgeocoder}` package from @dieghernan.

Usage

```
esri_wkids
```

Format

An object of class `integer` of length 2886.

Value

a numeric vector of well-known IDs

`find_address_candidates` *Find Address Candidates*

Description

Given an address, returns geocode result candidates.

Usage

```
find_address_candidates(
  single_line = NULL,
  address = NULL,
  address2 = NULL,
  address3 = NULL,
  neighborhood = NULL,
  city = NULL,
  subregion = NULL,
  region = NULL,
  postal = NULL,
  postal_ext = NULL,
  country_code = NULL,
  search_extent = NULL,
  location = NULL,
  category = NULL,
  crs = NULL,
  max_locations = NULL,
  for_storage = FALSE,
  match_out_of_range = NULL,
  location_type = NULL,
  lang_code = NULL,
  source_country = NULL,
  preferred_label_values = NULL,
  magic_key = NULL,
  geocoder = default_geocoder(),
  token = arc_token(),
  .progress = TRUE
)
```

Arguments

`single_line`	a character vector of addresses to geocode. If provided other `address` fields cannot be used. If `address` is not provided, `single_line` must be.
`address`	a character vector of the first part of a street address. Typically used for the street name and house number. But can also be a place or building name. If `single_line` is not provided, `address` must be.

address2	a character vector of the second part of a street address. Typically includes a house number, sub-unit, street, building, or place name. Optional.
address3	a character vector of the third part of an address. Optional.
neighborhood	a character vector of the smallest administrative division associated with an address. Typically, a neighborhood or a section of a larger populated place. Optional.
city	a character vector of the next largest administrative division associated with an address, typically, a city or municipality. A city is a subdivision of a subregion or a region. Optional.
subregion	a character vector of the next largest administrative division associated with an address. Depending on the country, a subregion can represent a county, state, or province. Optional.
region	a character vector of the largest administrative division associated with an address, typically, a state or province. Optional.
postal	a character vector of the standard postal code for an address, typically, a three– to six-digit alphanumeric code. Optional.
postal_ext	a character vector of the postal code extension, such as the United States Postal Service ZIP+4 code, provides finer resolution or higher accuracy when also passing postal. Optional.
country_code	default NULL. An ISO 3166 country code. See iso_3166_codes() for valid ISO codes. Optional.
search_extent	an object of class bbox that limits the search area. This is especially useful for applications in which a user will search for places and addresses within the current map extent. Optional.
location	an sfc_POINT object that centers the search. Optional.
category	a scalar character. Place or address type that can be used to filter suggest results. Optional.
crs	the CRS of the returned geometries. Passed to sf::st_crs(). Ignored if locations is not an sfc_POINT object.
max_locations	the maximum number of results to return. The default is 15 with a maximum of 50. Optional.
for_storage	default FALSE. Whether or not the results will be saved for long term storage.
match_out_of_range	set to TRUE by service by default. Matches locations Optional.
location_type	default "rooftop". Must be one of "rooftop" or "street". Optional.
lang_code	default NULL. An ISO 3166 country code. See iso_3166_codes() for valid ISO codes. Optional.
source_country	default NULL. An ISO 3166 country code. See iso_3166_codes() for valid ISO codes. Optional.
preferred_label_values	default NULL. Must be one of "postalCity" or "localCity". Optional.
magic_key	a unique identifier returned from suggest_places(). When a magic_key is provided, results are returned faster. Optional.

geocoder	default default_geocoder().
token	an object of class httr2_token as generated by auth_code() or related function
.progress	default TRUE. Whether a progress bar should be provided.

Details

Utilizes the /findAddressCandidates endpoint.

The endpoint can only handle one request at a time. To make the operation as performant as possible, requests are sent in parallel using httr2::req_perform_parallel(). The JSON responses are then processed using Rust and returned as an sf object.

Value

An sf object with 60 columns.

Examples

```
candidates_from_single <- find_address_candidates(
  single_line = "Bellwood Coffee, 1366 Glenwood Ave SE, Atlanta, GA, 30316, USA"
)

candidates_from_parts <- find_address_candidates(
  address = c("Bellwood Coffee", "Joe's coffeehouse"),
  address2 = c("1366 Glenwood Ave SE", "510 Flat Shoals Ave"),
  city = "Atlanta",
  region = "GA",
  postal = "30316",
  country_code = "USA"
)

str(candidates_from_parts)
```

---

geocode_addresses *Batch Geocode Addresses*

Description

Gecocode a vector of addresses in batches.

Usage

```
geocode_addresses(
  single_line = NULL,
  address = NULL,
  address2 = NULL,
  address3 = NULL,
```

```
  neighborhood = NULL,
  city = NULL,
  subregion = NULL,
  region = NULL,
  postal = NULL,
  postal_ext = NULL,
  country_code = NULL,
  location = NULL,
  search_extent = NULL,
  category = NULL,
  crs = NULL,
  max_locations = NULL,
  for_storage = FALSE,
  match_out_of_range = NULL,
  location_type = NULL,
  lang_code = NULL,
  source_country = NULL,
  preferred_label_values = NULL,
  batch_size = NULL,
  geocoder = default_geocoder(),
  token = arc_token(),
  .progress = TRUE
)
```

Arguments

single_line	a character vector of addresses to geocode. If provided other address fields cannot be used. If address is not provided, single_line must be.
address	a character vector of the first part of a street address. Typically used for the street name and house number. But can also be a place or building name. If single_line is not provided, address must be.
address2	a character vector of the second part of a street address. Typically includes a house number, sub-unit, street, building, or place name. Optional.
address3	a character vector of the third part of an address. Optional.
neighborhood	a character vector of the smallest administrative division associated with an address. Typically, a neighborhood or a section of a larger populated place. Optional.
city	a character vector of the next largest administrative division associated with an address, typically, a city or municipality. A city is a subdivision of a subregion or a region. Optional.
subregion	a character vector of the next largest administrative division associated with an address. Depending on the country, a subregion can represent a county, state, or province. Optional.
region	a character vector of the largest administrative division associated with an address, typically, a state or province. Optional.
postal	a character vector of the standard postal code for an address, typically, a three– to six-digit alphanumeric code. Optional.

postal_ext	a character vector of the postal code extension, such as the United States Postal Service ZIP+4 code, provides finer resolution or higher accuracy when also passing postal. Optional.
country_code	default NULL. An ISO 3166 country code. See iso_3166_codes() for valid ISO codes. Optional.
location	an sfc_POINT object that centers the search. Optional.
search_extent	an object of class bbox that limits the search area. This is especially useful for applications in which a user will search for places and addresses within the current map extent. Optional.
category	a scalar character. Place or address type that can be used to filter suggest results. Optional.
crs	the CRS of the returned geometries. Passed to sf::st_crs(). Ignored if locations is not an sfc_POINT object.
max_locations	the maximum number of results to return. The default is 15 with a maximum of 50. Optional.
for_storage	default FALSE. Whether or not the results will be saved for long term storage.
match_out_of_range	set to TRUE by service by default. Matches locations Optional.
location_type	default "rooftop". Must be one of "rooftop" or "street". Optional.
lang_code	default NULL. An ISO 3166 country code. See iso_3166_codes() for valid ISO codes. Optional.
source_country	default NULL. An ISO 3166 country code. See iso_3166_codes() for valid ISO codes. Optional.
preferred_label_values	default NULL. Must be one of "postalCity" or "localCity". Optional.
batch_size	the number of addresses to geocode per request. Uses the suggested batch size property of the geocoder.
geocoder	default default_geocoder().
token	an object of class httr2_token as generated by auth_code() or related function
.progress	default TRUE. Whether a progress bar should be provided.

Details

Addresses are partitioned into batches of up to batch_size elements. The batches are then sent to the geocoding service in parallel using httr2::req_perform_parallel(). The JSON responses are then processed using Rust and returned as an sf object.

Utilizes the /geocodeAddresses endpoint.

Value

an sf object

Examples

```
# Example dataset from the Urban Institute
## Not run:
fp <- paste0(
  "https://urban-data-catalog.s3.amazonaws.com/",
  "drupal-root-live/2020/02/25/geocoding_test_data.csv"
)
to_geocode <- read.csv(fp)
geocode_addresses(
  address = to_geocode$address,
  city = to_geocode$city,
  region = to_geocode$state,
  postal = to_geocode$zip
)

## End(Not run)
```

---

geocode_server *Create a GeocodeServer*

Description

Create an object of class `GeocodeServer` from a URL. This object stores the service definition of the geocoding service as a list object.

Usage

```
geocode_server(url, token = arc_token())
```

Arguments

`url`	the URL of a geocoding server.
`token`	an object of class `httr2_token` as generated by `auth_code()` or related function

Value

an object of class `GeocodeServer`.

Examples

```
server_url <- "https://geocode.arcgis.com/arcgis/rest/services/World/GeocodeServer"
geocode_server(server_url)
```

iso_3166_codes *ISO 3166 Country Codes*

Description

Create a data.frame of ISO 3166 2 and 3 digit Country codes.

Usage

```
iso_3166_codes()
```

Details

Country codes provided by rust_iso3166.

Value

a data.frame with columns country, code_2, code_3.

Examples

```
head(iso_3166_codes())
```

list_geocoders *List Available Geocoder Services*

Description

Evaluates the logged in user from an authorization token and returns a data.frame containing the available geocoding services for the associated token.

For users who have not signed into a private portal or ArcGIS Online, the public ArcGIS World Geocoder is used. Otherwise, the first available geocoding service associated with your authorization token is used.

Usage

```
list_geocoders(token = arc_token())

default_geocoder(token = arc_token())

world_geocoder()
```

Arguments

token	an object of class httr2_token as generated by auth_code() or related function

Details

The default_geocoder() will return the ArcGIS World Geocoder if no token is available. list_geocoder() requires an authorization token.

The ArcGIS World Geocoder is made publicly available for some uses. The world_geocoder() is used as the default GeocodeServer object in default_geocoder() when no authorization token is found. The find_address_candidates(), reverse_geocode(), and suggest_places() can be used without an authorization token. The geocode_addresses() funciton requires an authorization token to be used for batch geocoding.

To manually create a GeocodeServer object, see geocode_server().

Value

a data.frame with columns url, northLat, southLat, eastLon, westLon, name, suggest, zoomScale, placefinding, batch.

Examples

```
# Default geocoder object
# ArcGIS World Geocoder b/c no token
default_geocoder()

# Requires an Authorization Token
## Not run:
list_geocoders()

## End(Not run)
```

---

reverse_geocode *Reverse Geocode Locations*

Description

Determines the address for a given point.

Usage

```
reverse_geocode(
  locations,
  crs = sf::st_crs(locations),
  ...,
  lang_code = NULL,
  feature_type = NULL,
  location_type = c("rooftop", "street"),
  preferred_label_values = c("postalCity", "localCity"),
  for_storage = FALSE,
  geocoder = default_geocoder(),
  token = arc_token(),
```

```
  .progress = TRUE
)
```

Arguments

locations	an sfc_POINT object of the locations to be reverse geocoded.
crs	the CRS of the returned geometries. Passed to sf::st_crs(). Ignored if locations is not an sfc_POINT object.
...	unused.
lang_code	default NULL. An ISO 3166 country code. See iso_3166_codes() for valid ISO codes. Optional.
feature_type	limits the possible match types returned. Must be one of "StreetInt", "DistanceMarker", "StreetAddress", "StreetName", "POI", "Subaddress", "PointAddress", "Postal", or "Locality". Optional.
location_type	default "rooftop". Must be one of "rooftop" or "street". Optional.
preferred_label_values	default NULL. Must be one of "postalCity" or "localCity". Optional.
for_storage	default FALSE. Whether or not the results will be saved for long term storage.
geocoder	default default_geocoder().
token	an object of class httr2_token as generated by auth_code() or related function
.progress	default TRUE. Whether a progress bar should be provided.

Details

This function utilizes the /reverseGeocode endpoint of a geocoding service. By default, it uses the public ArcGIS World Geocoder.

- Intersection matches are only returned when feature_types = "StreetInt". See REST documentation for more.

Location Type:

- Specifies whether the output geometry shuold be the rooftop point or the street entrance location.
- The location_type parameter changes the geometry's placement but does not change the attribute values of X, Y, or DisplayX, and DisplayY.

Storage:

Very Important

The argument for_storage is used to determine if the request allows you to persist the results of the query. It is important to note that there are contractual obligations to appropriately set this argument. **You cannot save or persist results** when for_storage = FALSE (the default).

Execution:

The /reverseGeocode endpoint can only handle one address at a time. To make the operation as performant as possible, requests are sent in parallel using httr2::req_perform_parallel(). The JSON responses are then processed using Rust and returned as an sf object.

Value

An sf object.

Examples

```
# Find addresses from locations
reverse_geocode(c(-117.172, 34.052))
```

`storage` *Storing Geocoding Results*

Description

The results of geocoding operations cannot be stored or persisted unless the `for_storage` argument is set to `TRUE`. The default argument value is `for_storage = FALSE`, which indicates the results of the operation can't be stored, but they can be temporarily displayed on a map, for instance. If you store the results, in a database, for example, you need to set this parameter to true.

Details

See the official documentation for more context.

`suggest_places` *Search Suggestion*

Description

This function returns candidate locations based on a partial search query. It is designed to be used in an interactive search experience in a client facing application.

Usage

```
suggest_places(
  text,
  location = NULL,
  category = NULL,
  search_extent = NULL,
  max_suggestions = NULL,
  country_code = NULL,
  preferred_label_values = NULL,
  geocoder = default_geocoder(),
  token = arc_token()
)
```

Arguments

text	a scalar character of search key to generate a place suggestion.
location	an sfc_POINT object that centers the search. Optional.
category	a scalar character. Place or address type that can be used to filter suggest results. Optional.
search_extent	an object of class bbox that limits the search area. This is especially useful for applications in which a user will search for places and addresses within the current map extent. Optional.
max_suggestions	default NULL. The maximum number of suggestions to return. The service default is 5 with a maximum of 15.
country_code	default NULL. An ISO 3166 country code. See iso_3166_codes() for valid ISO codes. Optional.
preferred_label_values	default NULL. Must be one of "postalCity" or "localCity". Optional.
geocoder	default default_geocoder().
token	an object of class httr2_token as generated by auth_code() or related function

Details

Unlike the other functions in this package, suggest_places() is not vectorized as it is intended to provide search suggestions for individual queries such as those made in a search bar.

Utilizes the /suggest endpoint.

Value

A data.frame with 3 columns: text, magic_key, and is_collection.

Examples

```
# identify a search point
location <- sf::st_sfc(sf::st_point(c(-84.34, 33.74)), crs = 4326)

# create a search extent from it
search_extent <- sf::st_bbox(sf::st_buffer(location, 10))

# find suggestions from it
suggestions <- suggest_places(
  "bellwood",
  location,
  search_extent = search_extent
)

# get address candidate information
# using the text and the magic key
find_address_candidates(
```

```
  suggestions$text,
  magic_key = suggestions$magic_key
)
```

Index