Package 'arcgisutils'

February 23, 2024

Title ArcGIS Utility Functions

Version 0.2.0

Description Developer oriented utility functions designed to be used as the building blocks of R packages that work with ArcGIS Location Services. It provides functionality for authorization, Esri JSON construction and parsing, as well as other utilities pertaining to geometry and Esri type conversions. To support 'ArcGIS Pro' users, authorization can be done via 'arcgisbinding'. Installation instructions for 'arcgisbinding' can be found at <https://r.esri.com/r-bridge-site/arcgisbinding/installing-arcgisbinding.html>.

License Apache License (>= 2)

URL https://github.com/R-ArcGIS/arcgisutils, https://r.esri.com/arcgisutils/

BugReports https://github.com/R-ArcGIS/arcgisutils/issues

Imports cli, dbplyr, httr2 (>= 1.0.0), jsonify, Rcpp, RcppSimdJson, rlang, sf, utils

Suggests arcgisbinding, testthat (>= 3.0.0)

LinkingTo Rcpp

Config/testthat/edition 3

Encoding UTF-8

Language en

RoxygenNote 7.3.1

NeedsCompilation yes

Author Josiah Parry [aut, cre] (<https://orcid.org/0000-0001-9910-865X>), Kenneth Vernon [ctb] (<https://orcid.org/0000-0003-0098-5092>), Martha Bass [ctb] (<https://orcid.org/0009-0004-0268-5426>)

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

Repository CRAN

Date/Publication 2024-02-22 23:00:08 UTC

R topics documented:

`arc_agent` *Set user-agent for arcgisutils*

Description

Override the default user-agent set by httr2 to indicate that a request came from arcgisutils.

Usage

```
arc_agent(req)
```

Arguments

`req`	an httr2 request

Value

an httr2 request object

Examples

```
req <- httr2::request("http://example.com")
arc_agent(req)
```

arc_base_req *Generate base request*

Description

This function takes a url and creates a basic httr2 request that adds the user-agent and adds an authorization token to the `X-Esri-Authorization` header.

Usage

```
arc_base_req(url, token = NULL, error_call = rlang::caller_env())
```

Arguments

`url`	a valid url that is passed to `httr2::request()`
`token`	an object of class `httr2_token` as generated by `auth_code()` or related function
`error_call`	the caller environment to be used when propagating errors.

Examples

```
arc_base_req("https://arcgis.com")
```

arc_host *Determines Portal Host*

Description

Returns a scalar character indicating the host to make requests to.

Usage

```
arc_host()
```

Details

By default, the host is ArcGIS Online <`https://www.arcgis.com`>. If the environment variable `ARCGIS_HOST` is set, it will be returned.

Value

A scalar character, "`https://www.arcgis.com`" by default.

Examples

```
arc_host()
```

arc_token *Manage authorization tokens*

Description

These functions are used to set, fetch, and check authorization tokens.

Usage

```
arc_token(token = "ARCGIS_TOKEN")

set_arc_token(token, ...)

unset_arc_token(token = NULL)

obj_check_token(token, call = rlang::caller_env())

check_token_has_user(token, call = rlang::caller_env())
```

Arguments

token	for arc_token(), the name of a token to fetch. For set_arc_token(), it is an httr2_token that will be set. For unset_arc_token(), a character vector of token names to be unset.
...	named arguments to set httr2_token. Must be valid names and must be an httr2_token.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages.

Details

It is possible to have multiple authorization tokens in one session. These functions assist you in managing them.

arc_token() is used to fetch tokens by name. The default token is ARCGIS_TOKEN. However, they can be any valid character scalar. set_arc_token() will create store a token with the name ARCGIS_TOKEN. However, you can alternatively set the tokens by name using a key-value pair. The key is what you would pass to arc_token() to fetch the httr2_token object. To remove a token that has been set, use unset_arc_token().

`obj_check_token()` is a developer oriented function that can be used to check if an object is indeed an `httr2_token`. To check if a token has expired, `validate_or_refresh_token()` will do so.

`check_token_has_user()` is a developer oriented function that checks to see if a token has a `username` field associated with it.

For developers:

`set_arc_token()` uses a package level environment to store the tokens. The tokens are fetched from the environment using `arc_token()`.

Examples

```
# create fake tokens
token_a <- httr2::oauth_token("1234", arcgis_host = arc_host())
token_b <- httr2::oauth_token("abcd", arcgis_host = arc_host())

# set token to the default location
set_arc_token(token_a)

# fetch token from the default location
arc_token()

# set token by name
set_arc_token(org_a = token_a, org_b = token_b)

# fetch token by name
arc_token("org_a")
arc_token("org_b")

# unset tokens
unset_arc_token()
unset_arc_token(c("org_a", "org_b"))
```

---

as_esri_geometry *Create Esri geometry objects*

Description

These functions convert R objects to Esri json representations. There are three types of representations. These are, from smallest to largest, a geometry object, a feature and a feature set.

Usage

```
as_esri_geometry(x, crs = 4326, ..., call = rlang::caller_env())

as_esri_features(x, ..., call = rlang::caller_env())

as_esri_featureset(x, ...)

as_geometry(x, crs, ...)
```

```
as_features(x, ..., call = rlang::caller_env())

as_featureset(x, ...)
```

Arguments

x	an sf or sfc class object
crs	the coordinate reference system of the FeatureSet. Must be interpretable by `sf::st_crs()`
...	additional arguments passed on to methods.
call	The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be `NULL` or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages.

Details

The `_esri_` infix indicates that the input object will be converted directly into the Esri JSON representation. If there is no `_esri_` infix, the object will be converted into the appropriate list structure requiring only `jsonify::to_json(x, unbox = TRUE)` to convert to Esri JSON.

- `as_esri_geometry()` converts an `sfg` object to a geometry object
- `as_esri_features()` converts an `sfc` or `sf` object to a list of features
- `as_esri_featureset()` converts an `sf`, `sfc`, or `data.frame` to a feature set object

Geometry object contain the coordinates for a geometry. Features are geometries that have associated attributes with them. This would be similar to a row in an sf object. FeatureSets are a list of features that have additional metadata fields such as `spatialReference`, `geomtryType`, and `fields`. FeatureSets correspond to an `sf` object.

Geometry objects are defined for 5 different types. These are:

- Point: `esriGeometryPoint`
- Multipoint: `esriGeometryMultipoint`
- Polyline: `esriGeometryPolyline`
  - note that polyline encompasses both LINESTRING and MULTILINESTRING
- Polygon: `esriGeometryPolygon`
  - note that polygon encompasses both POLYGON and MULTIPOLYGON
- Envelope: `esriGeometryEnvelope`
  - envelopes correspond with bounding boxes but can have a Z dimension

Field handling:

- Vectors that inherit `Date` or `POSIXt` are converted into milliseconds since the Unix epoch in UTC timezone using `date_to_ms()`.
- `factors` are converted to character vectors using `as.character()` to match the behavior in `infer_esri_type()` which defines type mapping for Esri field types and R vector classes.

Value

a json Esri geometry object

Examples

```
library(sf)
as_esri_geometry(st_point(c(0, 1, 3, 4)))
as_esri_geometry(st_multipoint(x = matrix(runif(4), ncol = 2)))
as_esri_geometry(st_linestring(x = matrix(runif(2), ncol = 2)))
as_esri_geometry(st_linestring(x = matrix(runif(2), ncol = 2)))

# polygon
m <- matrix(
  c(0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 2, 2, 1, 2, 3, 1, 3, 2, 0, 0, 0),
  ncol = 3,
  byrow = TRUE
)
as_esri_geometry(st_polygon(list(m)))
```

---

as_extent *Convert an object to an extent*

Description

Given an sf or sfc object create a list that represents the extent of the object. The result of this function can be parsed directly into json using `jsonify::to_json(x, unbox = TRUE)` or included into a list as the extent component that will be eventually converted into json using the above function.

Usage

```
as_extent(x, crs = sf::st_crs(x), call = rlang::caller_env())
```

Arguments

`x`	an sf or sfc object
`crs`	the CRS of the object. Must be parsable by `sf::st_crs()`

`call` The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error.

You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message.

Can also be `NULL` or a defused function call to respectively not display any call or hard-code a code to display.

For more information about error calls, see Including function calls in error messages.

Value

An extent json object. Use `jsonify::to_json(x, unbox = TRUE)` to convert to json.

Examples

```
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
as_extent(nc)
```

---

as_layer *Create Esri layer objects*

Description

These functions are used to generate list objects that can be converted into json objects that are used in REST API requests. Notably they are used for adding R objects as items to a portal.

Usage

```
as_layer(
  x,
  name,
  title,
 layer_definition = as_layer_definition(x, name, "object_id", infer_esri_type(x)),
  id = NULL,
  layer_url = NULL,
  legend_url = NULL,
  popup_info = NULL,
  call = rlang::caller_env()
)

as_layer_definition(
  x,
  name,
  object_id_field,
  fields = infer_esri_type(x),
  display_field = NULL,
```

```
  drawing_info = NULL,
  has_attachments = FALSE,
  max_scale = 0,
  min_scale = 0,
  templates = NULL,
  type_id_field = NULL,
  types = NULL,
  call = rlang::caller_env()
)

as_feature_collection(
  layers = list(),
  show_legend = TRUE,
  call = rlang::caller_env()
)
```

Arguments

x	an object of class data.frame. This can be an sf object or tibble or any other subclass of data.frame.
name	a scalar character of the name of the layer. Must be unique.
title	A user-friendly string title for the layer that can be used in a table of contents.
layer_definition	a layer definition list as created by as_layer_definition(). A default is derived from x and the name object.
id	A number indicating the index position of the layer in the WMS or map service.
layer_url	default NULL. A string URL to a service that should be used for all queries against the layer. Used with hosted tiled map services on ArcGIS Online when there is an associated feature service that allows for queries.
legend_url	default NULL. A string URL to a legend graphic for the layer. Used with WMS layers. The URL usually contains a GetLegendGraphic request.
popup_info	default NULL. A list that can be converted into a popupInfo object defining the pop-up window content for the layer. There is no helper for popupInfo objects.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages.
object_id_field	a scalar character vector indicating the name of the object ID field in the dataset.
fields	a data.frame describing the fields in x. These values are inferred by default via infer_esri_type().

display_field	default NULL. A scalar character containing the name of the field that best summarizes the feature. Values from this field are used by default as the titles for pop-up windows.
drawing_info	default NULL. See REST documentation in details for more. There are no helpers or validators for drawingInfo objects.
has_attachments	default FALSE.
max_scale	default NULL. A number representing the maximum scale at which the layer definition will be applied. The number is the scale's denominator; thus, a value of 2400 represents a scale of 1/2,400. A value of 0 indicates that the layer definition will be applied regardless of how far you zoom in.
min_scale	default NULL. A number representing the minimum scale at which the layer definition will be applied.
templates	default NULL. See REST documentation in details for more.
type_id_field	default NULL. See REST documentation in details for more.
types	An array of type objects available for the dataset. This is used when the type_id_field is populated. NOTE there are no helper functions to create type objects. Any type list objects must match the json structure when passed to jsonify::to_json(x, unbox = TRUE).
layers	a list of layers as created by as_layer().
show_legend	default FALSE. Logical scalar indicating if this layer should be shown in the legend in client applications.

Details

A featureCollection defines a layer of features that will be stored on a web map. It consists of an array of layers. The layer contains the features (attributes and geometries) as a featureSet (see as_featureset()) and additional metadata which is stored in the layerDefinitionobject. The layerDefinition most importantly documents the fields in the object, the object ID, and additional metadata such as name, title, and display scale.

Additional documentation for these json object:

- layer
- layerDefinition
- featureCollection

Value

A list object containing the required fields for each respective json type. The results can be converted to json using jsonify::to_json(x, unbox = TRUE)

See Also

as_featureset()

Examples

```
ld <- as_layer_definition(iris, "iris", "objectID")
l <- as_layer(iris, "iris name", "Iris Title")
fc <- as_feature_collection(layers = list(l))
```

---

auth_code *Authorization*

Description

Authorize your R session to connect to an ArcGIS Portal. See details.

Usage

```
auth_code(client = Sys.getenv("ARCGIS_CLIENT"), host = arc_host())

auth_client(
  client = Sys.getenv("ARCGIS_CLIENT"),
  secret = Sys.getenv("ARCGIS_SECRET"),
  host = arc_host(),
  expiration = 120
)

auth_binding()

auth_user(
  username = Sys.getenv("ARCGIS_USER"),
  password = Sys.getenv("ARCGIS_PASSWORD"),
  host = arc_host(),
  expiration = 60
)

refresh_token(token, client = Sys.getenv("ARCGIS_CLIENT"), host = arc_host())

validate_or_refresh_token(
  token,
  client = Sys.getenv("ARCGIS_CLIENT"),
  host = arc_host(),
  refresh_threshold = 0,
  call = rlang::caller_env()
)
```

Arguments

client	an OAuth 2.0 developer application client ID. By default uses the environment variable ARCGIS_CLIENT.
host	default "https://www.arcgis.com"

secret — an OAuth 2.0 developer application secret. By default uses the environment variable ARCGIS_SECRET.

expiration — the duration of the token in minutes.

username — default Sys.getenv("ARCGIS_USER"). Your username to login. **Do not** hard code this value.

password — default Sys.getenv("ARCGIS_PASSWORD"). Your password to login. **Do not** hard code this value.

token — an httr2_token as created by auth_code() or similar

refresh_threshold — default 0. If token expiry is within this threshold (in seconds) the token will be refreshed only if a refresh_token is available. Token refreshing is only possible with auth_code() flow.

call — The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error.

You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.

Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display.

For more information about error calls, see Including function calls in error messages.

Details

ArcGIS Online and Enterprise Portals utilize OAuth2 authorization via their REST APIs.

- auth_code() is the recommend OAuth2 workflow for interactive sessions
- auth_client() is the recommended OAuth2 workflow for non-interactive sessions
- auth_user() uses legacy username and password authorization using the generateToken endpoint. It is only recommended for legacy systems that do not implement OAuth2.
- auth_binding() fetches a token from the active portal set by arcgisbinding. Uses arcgisbinding::arc.check_por to extract the authorization token. Recommended if using arcgisbinding.

Value

an httr2_token

Examples

```
## Not run:
  auth_code()
  auth_client()
  auth_user()
  auth_binding()

## End(Not run)
```

compact *General utility functions*

Description

General utility functions

Usage

```
compact(.x)

a %||% b

check_dots_named(dots, call = rlang::caller_env())
```

Arguments

.x	a list
a	an R object
b	an R object
dots	a list collected from dots via rlang::list2(...)
call	default rlang::caller_env(). The caller environment passed to cli::cli_abort()

Details

- compact() removes any NULL list elements
- %||% is a special pipe operator that returns b if a is NULL

Value

- compact() a list
- %||% the first non-null item or NULL if both are NULL

Examples

```
# remove null elements
compact(list(a = NULL, b = 1))

# if NULL return rhs
NULL %||% 123

# if not NULL return lhs
123 %||% NULL
```

`detect_errors` *Detect errors in parsed json response*

Description

The requests responses from ArcGIS don't return the status code in the response itself but rather from the body in the json. This function checks for the existence of an error. If an error is found, the contents of the error message are bubbled up.

Usage

```
detect_errors(response, error_call = rlang::caller_env())
```

Arguments

`response`	a `httr2::response` object.
`error_call`	default `rlang::caller_env()`. The environment from which to throw the error from.

Value

Nothing. Used for it's side effect. If an error code is encountered in the response an error is thrown with the error code and the error message.

Examples

```
## Not run:
  response <- list(
    error = list(
      code = 400L,
      message = "Unable to generate token.",
      details = "Invalid username or password."
    )
  )

  detect_errors(response)

## End(Not run)
```

determine_dims *Determine the dimensions of a geometry object*

Description

Given an sfc or sfg object determine what dimensions are represented.

Usage

```
determine_dims(x)

has_m(x)

has_z(x)
```

Arguments

x	an object of class sfc or sfg

Value

determine_dims() returns a scalar character of the value "xy", "xyz", or "xyzm" depending on what dimensions are represented.

has_m() and has_z() returns a logical scalar of TRUE or FALSE if the geometry has a Z or M dimension.

Examples

```
geo <- sf::st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)[["geometry"]]

determine_dims(geo)
has_z(geo)
has_m(geo)
```

determine_esri_geo_type *Determine Esri Geometry type*

Description

Takes an sf or sfc object and returns the appropriate Esri geometry type.

Usage

```
determine_esri_geo_type(x, call = rlang::caller_env())
```

Arguments

x	an object of class data.frame, sf, sfc, or sfg.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages.

Details

Geometry type mapping:

- POINT: esriGeometryPoint
- MULTIPOINT: esriGeometryMultipoint
- LINESTRING: esriGeometryPolyline
- MULTILINESTRING: esriGeometryPolyline
- POLYGON: esriGeometryPolygon
- MULTIPOLYGON: esriGeometryPolygon

Value

returns a character scalar of the corresponding Esri geometry type

Examples

```
determine_esri_geo_type(sf::st_point(c(0, 0)))
```

---

fetch_layer_metadata *Retrieve metadata*

Description

Utility functions for feature service metadata.

Usage

```
fetch_layer_metadata(url, token = NULL, call = rlang::caller_env())
```

Arguments

url	the url of the item.
token	an httr2_token from one of the provided auth_ functions
call	default rlang::caller_env(). The calling environment passed to detect_errors().

Details

- `fetch_layer_metadata()` given a request, fetches the metadata by setting the query parameter `f=json`

Value

returns a list object

Examples

```
# url is broken into parts to fit within 100 characters to avoid CRAN notes
url_parts <- c(
  "https://services.arcgis.com/P3ePLMYs2RVChkJx/ArcGIS/rest/services",
  "/USA_Counties_Generalized_Boundaries/FeatureServer/0"
)

furl <- paste0(url_parts, collapse = "")
meta <- fetch_layer_metadata(furl)
head(names(meta))
```

`infer_esri_type` *Esri field type mapping*

Description

Infers Esri field types from R objects.

Usage

```
infer_esri_type(
  .data,
  arg = rlang::caller_arg(.data),
  call = rlang::caller_env()
)

remote_ptype_tbl(fields, call = rlang::caller_env())

get_ptype(field_type, call = rlang::caller_env())
```

Arguments

`.data`	an object of class `data.frame`.
`arg`	An argument name in the current function.
`call`	The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error.

	You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be `NULL` or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages.
`fields`	a data.frame containing, at least, the columns `type` and `name`. Typically retrieved from the `field` metadata from a `FeatureLayer` or `Table`. Also can use the output of `infer_esri_type()`.
`field_type`	a character of a desired Esri field type. See details for more.

Details

- `get_ptype()` takes a scalar character containing the Esri field type and returns a prototype of the pertinent R type
- `infer_esri_type()` takes a data frame-like object and infers the Esri field type from it.
- `remote_ptype_tbl()` takes a data frame of fields as derived from `list_fields()` and creates a lazy table proto type intended to be used with `dbplyr` integration

Field type mapping::

Esri field types are mapped as

- `esriFieldTypeSmallInteger`: integer
- `esriFieldTypeSingle`: double
- `esriFieldTypeGUID`: integer
- `esriFieldTypeOID`: integer
- `esriFieldTypeInteger`: integer
- `esriFieldTypeBigInteger`: double
- `esriFieldTypeDouble`: double
- `esriFieldTypeString`: character
- `esriFieldTypeDate`: date

R types are mapped as

- `double`: esriFieldTypeDouble
- `integer`: esriFieldTypeInteger
- `character`: esriFieldTypeString
- `date`: esriFieldTypeDate
- `raw`: esriFieldTypeBlob

Value

- `get_pytpe()` returns an object of the class of the prototype.
- `infer_esri_ptype()` returns a `data.frame` with columns `name`, `type`, `alias`, `nullable`, and `editable` columns
  - This resembles that of the `fields` returned by a FeatureService

Examples

```
get_ptype("esriFieldTypeDouble")

infer_esri_type(iris)
```

`is_date` *Date handling*

Description

Esri date fields are represented as milliseconds from the Unix Epoch.

Usage

```
is_date(x, tz)

date_to_ms(x, tz = "UTC")

from_esri_date(x)
```

Arguments

`x`	an object of class `Date` or `POSIXt`. In the case of `is_date()`, any R object.
`tz`	a character string. The time zone specification to be used for the conversion, *if one is required*. System-specific (see time zones), but `""` is the current time zone, and `"GMT"` is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

- `is_date()`: checks if an object is a `Date` or `POSIXt` class object.
- `date_to_ms()` converts a date object to milliseconds from the Unix Epoch in the specified time zone.

Value

- `is_date()` returns a logical scalar
- `date_to_ms()` returns a numeric vector of times in milliseconds from the Unix Epoch in the specified time zone.

Examples

```
today <- Sys.Date()

is_date(today)

date_to_ms(today)
```

`parse_esri_json` *Parse Esri JSON*

Description

Parses an Esri FeatureSet JSON object into an R object. If there is no geometry present, a data.frame is returned. If there is geometry, an sf object is returned.

Usage

```
parse_esri_json(string, ..., call = rlang::caller_env())
```

Arguments

`string`	the raw Esri JSON string.
`...`	additional arguments passed to `RcppSimdJson::fparse`
`call`	The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be `NULL` or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages.

Value

A data.frame. If geometry is found, returns an sf object.

Examples

```
esri_json <- '{
    "geometryType": "esriGeometryPolygon",
    "spatialReference": {
        "wkid": 4326
    },
```

```
    "hasZ": false,
    "hasM": false,
    "features": [
        {
            "attributes": {
                "id": 1
            },
            "geometry": {
                "rings": [
                    [
                        [0.0, 0.0],
                        [1.0, 0.0],
                        [1.0, 1.0],
                        [0.0, 1.0],
                        [0.0, 0.0]
                    ]
                ]
            }
        }
    ]
}'

parse_esri_json(esri_json)
```

`validate_crs` *Validate CRS object*

Description

Takes a representation of a CRS and ensures that it is a valid one. The CRS is validated using `sf::st_crs()` if it cannot be validated, a null CRS is returned.

Usage

```
validate_crs(crs, arg = rlang::caller_arg(crs), call = rlang::caller_env())
```

Arguments

`crs`	a representation of a coordinate reference system.
`arg`	An argument name in the current function.
`call`	The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be `NULL` or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages.

Details

See `sf::st_crs()` for more details on valid representations.

Value

Returns a list of length 1 with an element named `spatialReference` which is itself a named list.

If the provided CRS returns a valid well-known ID (WKID) `spatialReference` contains a named element called `wkid` which is the integer value of the WKID. If the WKID is not known but the CRS returned is a valid well-known text representation the `wkid` field is `NA` and another field `wkt` contains the valid wkt.

Examples

```
# using epsg code integer or string representation
validate_crs(3857)
validate_crs("EPSG:4326")

# using a custom proj4 string
proj4string <- "+proj=longlat +datum=WGS84 +no_defs"

crs <- validate_crs(proj4string)

# using wkt2 (from above result)
crs <- validate_crs(crs$spatialReference$wkt)
```

Index