Package: arcgislayers (via r-universe)

October 29, 2024

Type Package

Title An Interface to ArcGIS Data Services

Version 0.3.1.9000

Description Enables users of 'ArcGIS Enterprise', 'ArcGIS Online', or 'ArcGIS Platform' to read, write, publish, or manage vector and raster data via ArcGIS location services REST API endpoints <https://developers.arcgis.com/rest/>.

License Apache License (>= 2)

Encoding UTF-8

LazyData true

Imports arcgisutils (>= 0.2.0), arcpbf (>= 0.1.5), cli, httr2 (>= 1.0.5), jsonify, lifecycle, RcppSimdJson, rlang, sf, terra, utils

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests dbplyr, dplyr, rmarkdown, testthat (>= 3.0.0), tidyselect, vctrs

Config/testthat/edition 3

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

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

Repository https://r-arcgis.r-universe.dev

RemoteUrl https://github.com/r-arcgis/arcgislayers

RemoteRef HEAD

RemoteSha 896ac0028ed0d67c51ea83dea11bcec4d7cad969

Contents

`add_features` *Add Features to Feature Layer*

Description

Delete features from a feature layer based on object ID, a where clause, or a spatial filter.

Usage

```
add_features(
  x,
  .data,
  chunk_size = 2000,
  match_on = c("name", "alias"),
  rollback_on_failure = TRUE,
  token = arc_token()
)

update_features(
  x,
  .data,
  match_on = c("name", "alias"),
  token = arc_token(),
  rollback_on_failure = TRUE,
  ...
)

delete_features(
```

```
  x,
  object_ids = NULL,
  where = NULL,
  filter_geom = NULL,
  predicate = "intersects",
  rollback_on_failure = TRUE,
  token = arc_token(),
  ...
)
```

Arguments

x	an object of class FeatureLayer
.data	an object of class sf or data.frame
chunk_size	the maximum number of features to add at a time
match_on	whether to match on the alias or the field name. Default, the alias. See Details for more.
rollback_on_failure	default TRUE. Specifies whether the edits should be applied only if all submitted edits succeed.
token	default arc_token(). An httr2_token.
...	additional query parameters passed to the API.
object_ids	a numeric vector of object IDs to be deleted.
where	a simple SQL where statement indicating which features should be deleted. When the where statement evaluates to TRUE, those values will be deleted.
filter_geom	an object of class bbox, sfc or sfg used to filter query results based on a predicate function.
predicate	Spatial predicate to use with filter_geom. Default "intersects". Possible options are "intersects", "contains", "crosses", "overlaps", "touches", and "within".

Details

[Experimental]

For a more detailed guide to adding, updating, and deleting features, view the tutorial on the R-ArcGIS Bridge website.

Regarding the match_on argument:when publishing an object to an ArcGIS Portal from R, the object's names are provided as the alias. The object's names are subject to change according to the standards of the ArcGIS REST API. For example. "Sepal.Length" is changed to "Sepal_Width" in the name field but the alias remains "Sepal.Length". For that reason, we match on the alias name by default. Change this argument to match based on the field name.

Value

- add_features() returns a data.frame with columns objectId, uniqueId, globalId, success

- update_features() returns a list with an element named updateResults which is a data.frame with columns objectId, uniqueId, globalId, success
- delete_features() returns a list with an element named deleteResults which is a data.frame with columns objectId, uniqueId, globalId, success

Examples

```
## Not run:
  # this is pseudo-code and will not work
  flayer <- arc_open(furl)

  # add sf objects to existing feature service
  add_features(flayer, sfobj)

  # delete all features
  delete_features(flayer, where = "1 = 1")

  # update features
  update_features(flayer, dfobj)

## End(Not run)
```

---

add_item *Publish Content*

Description

Publishes an sf or data.frame object to an ArcGIS Portal as a FeatureCollection.

Usage

```
add_item(
  x,
  title,
  description = "",
  tags = character(0),
  snippet = "",
  categories = character(0),
  async = FALSE,
  type = "Feature Service",
  token = arc_token()
)

publish_item(
  item_id,
  publish_params = .publish_params(),
  file_type = "featureCollection",
  token = arc_token()
```

```
)

publish_layer(
  x,
  title,
  ...,
  publish_params = .publish_params(title, target_crs = sf::st_crs(x)),
  token = arc_token()
)

.publish_params(
  name = NULL,
  description = NULL,
  copyright = NULL,
  target_crs = 3857,
  max_record_count = 2000L
)
```

Arguments

x	an object of class `data.frame`. This can be an `sf` object or `tibble` or any other subclass of `data.frame`.
title	A user-friendly string title for the layer that can be used in a table of contents.
description	a length 1 character vector containing the description of the item that is being added. Note that the value cannot be larger than 64kb.
tags	a character vector of tags to add to the item.
snippet	a length 1 character vector with no more than 2048 characters.
categories	a character vector of the categories of the item.
async	default `FALSE`. Cannot be changed at this time.
type	default `"Feature Service"`. Must not be changed at this time.
token	an `httr2_token` as created by `auth_code()` or similar
item_id	the ID of the item to be published.
publish_params	a list of named values of the `publishParameters`. Must match the values in the /publish endpoint documentation.
file_type	default `"featureCollection"`. Cannot be changed.
...	arguments passed into `add_item()`.
name	a scalar character of the name of the layer. Must be unique.
copyright	an optional character scalar containing copyright text to add to the published Feature Service.
target_crs	the CRS of the Feature Service to be created. By default, `EPSG:3857`.
max_record_count	the maximum number of records that can be returned from the created Feature Service.

Details

[Experimental]

- `add_item()` takes a data.frame like object and uploads it as an item in your portal.
- `publish_item()` takes an ID of an item in your portal and publishes it as a feature service.
- `publish_layer()` is a high-level wrapper that first adds an object as an item in your portal and subsequently publishes it for you.
- `.publish_params()` is a utility function to specify optional publish parameters such as copyright text, and the spatial reference of the published feature collection.

Note that there is *only* support for feature services meaning that only tables and feature layers can be made by these functions.

Publish Parameters:

When publishing an item to a portal, a number of publish parameters can be provided. Most importantly is the `targetSR` which will be the CRS of the hosted feature service. By default this is `EPSG:3857`.

`publish_layer()` will use the CRS of the input object, `x`, by default. If publishing content in two steps with `add_item()` and `publish_item()`, use `.publish_params()` to craft your publish parameters. Ensure that the CRS provided to `target_crs` matches that of the item you added with `add_item()`.

Value

A named list containing the url of the newly published service.

Examples

```
## Not run:
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"))
  x <- nc[1:5, 13]

  token <- auth_code()
  set_arc_token(token)

  publish_res <- publish_layer(
    x, "North Carolina SIDS sample"
  )

## End(Not run)
```

---

arc_open *Open connection to remote resource*

Description

Provided a URL, create an object referencing the remote resource. The resultant object acts as a reference to the remote data source.

Usage

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

Arguments

url	The url of the remote resource. Must be of length one.
token	your authorization token.

Details

To extract data from the remote resource utilize arc_select() for objects of class FeatureLayer or Table. For ImageServers, utilize arc_raster().

[Experimental]

Value

Depending on the provided URL returns a FeatureLayer, Table, FeatureServer, ImageServer, or MapServer. Each of these objects is a named list containing the properties of the service.

See Also

arc_select arc_raster

Examples

```
## Not run:
# FeatureLayer
furl <- paste0(
  "https://services3.arcgis.com/ZvidGQkLaDJxRSJ2/arcgis/rest/services/",
  "PLACES_LocalData_for_BetterHealth/FeatureServer/0"
)

arc_open(furl)

# Table
furl <- paste0(
  "https://services.arcgis.com/P3ePLMYs2RVChkJx/arcgis/rest/services/",
  "USA_Wetlands/FeatureServer/1"
)

arc_open(furl)

# ImageServer
arc_open(
  "https://landsat2.arcgis.com/arcgis/rest/services/Landsat/MS/ImageServer"
)

# FeatureServer
furl <- paste0(
  "https://services3.arcgis.com/ZvidGQkLaDJxRSJ2/arcgis/rest/services/",
  "PLACES_LocalData_for_BetterHealth/FeatureServer"
```

```
)

arc_open(furl)

# MapServer
map_url <- paste0(
  "https://services.arcgisonline.com/ArcGIS/rest/services/",
  "World_Imagery/MapServer"
)

arc_open(map_url)

## End(Not run)
```

`arc_raster` *Read from an Image Server*

Description

Given an `ImageServer` export an image as a terra `SpatRaster` object. See `terra::rast`.

Usage

```
arc_raster(
  x,
  xmin,
  xmax,
  ymin,
  ymax,
  bbox_crs = NULL,
  crs = sf::st_crs(x),
  width = NULL,
  height = NULL,
  format = "tiff",
  ...,
  raster_fn = NULL,
  token = arc_token()
)
```

Arguments

`x`	an `ImageServer` as created with `arc_open()`.
`xmin`	the minimum bounding longitude value.
`xmax`	the maximum bounding longitude value.
`ymin`	that minimum bounding latitude value.
`ymax`	the maximum bounding latitude value.

bbox_crs	the CRS of the values passed to xmin, xmax, ymin, and ymax. If not specified, uses the CRS of x.
crs	the CRS of the resultant raster image and the provided bounding box defined by xmin, xmax, ymin, ymax (passed outSR query parameter).
width	default NULL. Cannot exceed x[["maxImageWidth"]].
height	default NULL. Cannot exceed x[["maxImageHeight"]].
format	default "tiff". Must be one of "jpgpng", "png", "png8", "png24", "jpg", "bmp", "gif", "tiff", "png32", "bip", "bsq", "lerc".
...	additional key value pairs to be passed to httr2::req_body_form().
raster_fn	a scalar string with the name of the service raster function. See list_service_raster_fns() for available raster functions.
token	default arc_token() authorization token.

Details

[Experimental]

Value

An object of class SpatRaster.

Examples

```
## Not run:
img_url <- "https://landsat2.arcgis.com/arcgis/rest/services/Landsat/MS/ImageServer"

landsat <- arc_open(img_url)

arc_raster(
  landsat,
  xmin = -71,
  xmax = -67,
  ymin = 43,
  ymax = 47.5,
  bbox_crs = 4326,
  width = 100,
  height = 100
)

## End(Not run)
```

arc_read *Read an ArcGIS FeatureLayer, Table, or ImageServer*

Description

arc_read() combines the functionality of arc_open() with arc_select() or arc_raster() to read an ArcGIS FeatureLayer, Table, or ImageServer to an sf or SpatRaster object. Optionally, set, check, or modify names for the returned data frame or sf object using the col_names and name_repair parameters. For ease of use and convenience, arc_read() allows users to access and query a FeatureLayer, Table, or ImageServer with a single function call instead of combining arc_open() and arc_select(). The conventions of col_select are based on functions for reading tabular data in the {readr} package.

Usage

```
arc_read(
  url,
  col_names = TRUE,
  col_select = NULL,
  n_max = Inf,
  name_repair = "unique",
  crs = NULL,
  ...,
  fields = NULL,
  alias = c("drop", "label", "replace"),
  token = arc_token()
)
```

Arguments

url	The url of the remote resource. Must be of length one.
col_names	Default TRUE. Column names or name handling rule. col_names can be TRUE, FALSE, NULL, or a character vector: • If TRUE, use existing default column names for the layer or table. If FALSE or NULL, column names will be generated automatically: X1, X2, X3 etc. • If col_names is a character vector, values replace the existing column names. col_names can't be length 0 or longer than the number of fields in the returned layer.
col_select	Default NULL. A character vector of the field names to be returned. By default, all fields are returned.
n_max	Defaults to Inf or an option set with options("arcgislayers.n_max" = <max records>). Maximum number of records to return.
name_repair	Default "unique". See vctrs::vec_as_names() for details. If name_repair = NULL, names are set directly.

crs	the spatial reference to be returned. If the CRS is different than the CRS for the input FeatureLayer, a transformation will occur server-side. Ignored if x is a Table.
...	Additional arguments passed to arc_select() if URL is a FeatureLayer or Table or arc_raster() if URL is an ImageLayer.
fields	Default NULL. a character vector of the field names to returned. By default all fields are returned. Ignored if col_names is supplied.
alias	Use of field alias values. Default c("drop", "label", "replace"),. There are three options: • "drop", field alias values are ignored. • "label": field alias values are assigned as a label attribute for each field. • "replace": field alias values replace existing column names. col_names must TRUE for this option to be applied.
token	your authorization token.

Details

[Experimental]

Value

An sf object, a data.frame, or an object of class SpatRaster.

See Also

arc_select(); arc_raster()

Examples

```
## Not run:
 furl <- "https://sampleserver6.arcgisonline.com/arcgis/rest/services/Census/MapServer/3"

  # read entire service
  arc_read(furl)

  # apply tolower() to column names
  arc_read(url, name_repair = tolower)

  # use paste0 to prevent CRAN check NOTE
  furl <- paste0(
    "https://sampleserver6.arcgisonline.com/arcgis/rest/services/",
    "EmergencyFacilities/FeatureServer/0"
  )

 # use field aliases as column names
 arc_read(furl, col_names = "alias")

 # read an ImageServer directly
 img_url <- "https://landsat2.arcgis.com/arcgis/rest/services/Landsat/MS/ImageServer"
```

```
arc_read(
  img_url,
  width = 100, height = 100,
  xmin = -71, ymin = 43,
  xmax = -67, ymax = 47.5,
  bbox_crs = 4326
)


## End(Not run)
```

`arc_select` *Query a Feature Service*

Description

`arc_select()` takes a `FeatureLayer`, `Table`, of `ImageServer` object and returns data from the layer as an `sf` object or `data.frame` respectively.

Usage

```
arc_select(
  x,
  ...,
  fields = NULL,
  where = NULL,
  crs = sf::st_crs(x),
  geometry = TRUE,
  filter_geom = NULL,
  predicate = "intersects",
  n_max = Inf,
  page_size = NULL,
  token = arc_token()
)
```

Arguments

`x`	an object of class `FeatureLayer`, `Table`, or `ImageServer`.
`...`	additional query parameters passed to the API.
`fields`	a character vector of the field names that you wish to be returned. By default all fields are returned.
`where`	a simple SQL where statement indicating which features should be selected.
`crs`	the spatial reference to be returned. If the CRS is different than the CRS for the input `FeatureLayer`, a transformation will occur server-side. Ignored if x is a `Table`.
`geometry`	default `TRUE`. If geometries should be returned. Ignored for `Table` objects.

filter_geom	an object of class bbox, sfc or sfg used to filter query results based on a predicate function.
predicate	Spatial predicate to use with filter_geom. Default "intersects". Possible options are "intersects", "contains", "crosses", "overlaps", "touches", and "within".
n_max	the maximum number of features to return. By default returns every feature available. Unused at the moment.
page_size	the maximum number of features to return per request. Useful when requests return a 500 error code. See Details.
token	your authorization token.

Details

See reference documentation for possible arguments.

FeatureLayers can contain very dense geometries with a lot of coordinates. In those cases, the feature service may time out before all geometries can be returned. To address this issue, we can reduce the number of features returned per each request by reducing the value of the page_size parameter.

arc_select() works by sending a single request that counts the number of features that will be returned by the current query. That number is then used to calculate how many "pages" of responses are needed to fetch all the results. The number of features returned (page size) is set to the maxRecordCount property of the layer by default. However, by setting page_size to be smaller than the maxRecordCount we can return fewer geometries per page and avoid time outs.

[Experimental]

Value

An sf object, or a data.frame

Examples

```
## Not run:
# define the feature layer url
furl <- paste0(
  "https://services3.arcgis.com/ZvidGQkLaDJxRSJ2/arcgis/rest",
  "/services/PLACES_LocalData_for_BetterHealth/FeatureServer/0"
)

flayer <- arc_open(furl)

arc_select(
  flayer,
  fields = c("StateAbbr", "TotalPopulation")
)

arc_select(
  flayer,
  fields = c("OBJECTID", "PlaceName"),
```

```
  where = "TotalPopulation > 1000000"
)

## End(Not run)
```

`clear_query` *Utility functions*

Description

Utility functions

Usage

```
clear_query(x)

list_fields(x)

pull_field_aliases(x)

list_items(x)

refresh_layer(x)
```

Arguments

x	an object of class `FeatureLayer`, `Table`, or `ImageServer`.

Details

[Experimental]

- `list_fields()` returns a data.frame of the fields in a `FeatureLayer` or `Table`
- `list_items()` returns a data.frame containing the layers or tables in a `FeatureServer` or `MapServer`
- `clear_query()` removes any saved query in a `FeatureLayer` or `Table` object
- `refresh_layer()` syncs a `FeatureLayer` or `Table` with the remote resource picking up any changes that may have been made upstream. Returns an object of class `x`.
- `pull_field_aliases()` returns a named list of the field aliases from a `FeatureLayer` or `Table`

Value

See Details.

Examples

```
## Not run:
furl <- paste0(
  "https://services3.arcgis.com/ZvidGQkLaDJxRSJ2/arcgis/rest/services/",
  "PLACES_LocalData_for_BetterHealth/FeatureServer/0"
)

flayer <- arc_open(furl)

# list fields available in a layer
list_fields(flayer)

# remove any queries stored in the query attribute
clear_query(update_params(flayer, outFields = "*"))

# refresh metadata of an object
refresh_layer(flayer)

map_url <- paste0(
  "https://services.arcgisonline.com/ArcGIS/rest/services/",
  "World_Imagery/MapServer"
)

# list all items in a server object
list_items(arc_open(map_url))

## End(Not run)
```

---

create_feature_server *Create a FeatureServer*

Description

Creates an empty FeatureServer with no additional layers.

Usage

```
create_feature_server(
  service_name,
  description = "",
  crs = 3857,
  capabilities = c("create", "delete", "query", "update", "editing"),
  query_formats = c("json", "geojson"),
  initial_extent = list(xmin = NULL, xmax = NULL, ymin = NULL, ymax = NULL),
  max_record_count = 1000L,
  allow_updates = TRUE,
  copyright = "",
  has_static_data = FALSE,
  xss_prevention = xss_defaults(),
```

```
  token = arc_token()
)

xss_defaults()
```

Arguments

service_name	Feature Service name.
description	default blank. The description of the feature server.
crs	default 3857. A coordinate reference system to set for the feature server. Must be compatible with sf::st_crs().
capabilities	default full capabilities. Character vector of capabilities.
query_formats	default json and geojson. May be restricted by site-wide settings.
initial_extent	optional. A named list with element of xmin, xmax, ymin, and ymax. Values must be in the same CRS as crs.
max_record_count	default 1000. The maximum number of records that can be retrieved from a layer in one request.
allow_updates	default TRUE. Determine if geometries can be updated.
copyright	default blank. Copyright notice to provide in the Feature Server
has_static_data	default FALSE. Indicates if data is changing.
xss_prevention	cross-site-scripting prevention is enabled by default. See details for more.
token	an httr2_token as created by auth_code() or similar

Details

[Experimental]

Value

If a FeatureServer is created successfully, a FeatureServer object is returned based on the newly created feature server's url.

Examples

```
## Not run:
  set_arc_token(auth_code())
  create_feature_server("My empty feature server")

## End(Not run)
```

get_layer *Extract a layer from a Feature or Map Server*

Description

These helpers provide easy access to the layers contained in a FeatureServer or MapServer.

Usage

```
get_layer(x, id = NULL, name = NULL, token = arc_token())

get_all_layers(x, token = arc_token())

get_layers(x, id = NULL, name = NULL, token = arc_token())
```

Arguments

x	an object of class FeatureServer or MapServer
id	default NULL. A numeric vector of unique ID of the layer you want to retrieve. This is a scalar in get_layer().
name	default NULL. The name associated with the layer you want to retrieve. name is mutually exclusive with id. This is a scalar in get_layer().
token	your authorization token.

Details

[Experimental]

The id and name arguments must match the field values of the respective names as seen in the output of list_items()

Value

- get_layer() returns a single FeatureLayer or Table based on its ID
- get_layers() returns a list of the items specified by the id or name argument
- get_all_layers() returns a named list with an element layers and tables. Each a list containing FeatureLayer and Tables respectively.

Examples

```
## Not run:
  # FeatureServer
  furl <- paste0(
    "https://services3.arcgis.com/ZvidGQkLaDJxRSJ2/arcgis/rest/services/",
    "PLACES_LocalData_for_BetterHealth/FeatureServer"
  )
```

```
  fserv <- arc_open(furl)

  fserv
  get_layer(fserv, 0)
  get_layers(fserv, name = c("Tracts", "ZCTAs"))
  get_all_layers(fserv)

## End(Not run)
```

`get_layer_estimates` *Get Estimates*

Description

Get Estimates

Usage

```
get_layer_estimates(x, token = arc_token())
```

Arguments

`x`	an object of class `FeatureLayer`, `Table`, or `ImageServer`.
`token`	your authorization token.

Value

A named list containing all estimate info. If `extent` is present, it is available as an object of class `bbox`.

References

ArcGIS REST Doc

Examples

```
furl <- paste0(
  "https://services.arcgis.com/P3ePLMYs2RVChkJx/ArcGIS/rest/services/",
  "USA_Counties_Generalized_Boundaries/FeatureServer/0"
)

county_fl <- arc_open(furl)
get_layer_estimates(county_fl)
```

`list_service_raster_fns` *List Available Raster Funcitons*

Description

This function returns the `rasterFunctionInfos` field of the `ImageServer`'s metadata as a `data.frame`. If the field does not exist then an error is emitted.

Usage

```
list_service_raster_fns(
  x,
  arg = rlang::caller_arg(x),
  call = rlang::caller_call()
)
```

Arguments

`x`	an `ImageServer`.
`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.

Value

a data.frame of the available raster functions.

Examples

```
# use paste to avoid cran note
furl <- paste0(
  "https://di-usfsdata.img.arcgis.com/arcgis/rest/services",
  "/FIA_BIGMAP_2018_Species_Aboveground_Biomass/ImageServer"
)

service <- arc_open(furl)
raster_fns <- list_service_raster_fns(service)
head(raster_fns)
```

prepare_spatial_filter *Prepare JSON for use as a spatial filter based on feature geometry or bounding box input*

Description

`prepare_spatial_filter()` prepares a named list with ESRI JSON geometry for use as a spatial filter based on a a `sfc`, `sfg`, or `bbox` input object.

`match_spatial_rel()` takes a scalar character vector with a predicate name to a type of ESRI spatial relation.

Usage

```
prepare_spatial_filter(
  filter_geom,
  crs,
  predicate,
  error_call = rlang::caller_env()
)

match_spatial_rel(predicate, error_call = rlang::caller_env())
```

Arguments

`filter_geom`	an object of class `bbox`, `sfc` or `sfg` used to filter query results based on a predicate function.
`crs`	a representation of a coordinate reference system.
`predicate`	Spatial predicate to use with `filter_geom`. Default `"intersects"`. Possible options are `"intersects"`, `"contains"`, `"crosses"`, `"overlaps"`, `"touches"`, and `"within"`.
`error_call`	default `rlang::caller_env()`.

Details

Using `sfc` objects as `filter_geom`

[Experimental]

If an `sfc` object is provided it will be transformed to the layers spatial reference. If the `sfc` is missing a CRS (or is an `sfg` object) it is assumed to use the same spatial reference as the FeatureLayer. If the `sfc` object has multiple features, the features are unioned with `sf::st_union()`. If an `sfc` object has `MULTIPOLYGON` geometry, the features are cast to `POLYGON` geometry and only the first element is used.

Value

prepare_spatial_filter() returns a named list with the geometryType, geometry (as Esri JSON), and spatial relation predicate.

match_spatial_rel() returns one of the following spatial binary predicates:

- esriSpatialRelIntersects
- esriSpatialRelContains
- esriSpatialRelCrosses
- esriSpatialRelOverlaps
- esriSpatialRelTouches
- esriSpatialRelWithin

Examples

```
prepare_spatial_filter(sf::st_point(c(0, 5)), 4326, "intersects")
```

---

query_layer_attachments

Query and download attachments

Description

Get metadata about attachments associated with features in a layer. Query attachment information using query_layer_attachments() and download attachments using download_attachments().

Usage

```
query_layer_attachments(
  x,
  definition_expression = "1=1",
  attachments_definition_expression = NULL,
  object_ids = NULL,
  global_ids = NULL,
  attachment_types = NULL,
  keywords = NULL,
  return_metadata = TRUE,
  ...,
  token = arc_token()
)

download_attachments(
  attachments,
  out_dir,
  ...,
  overwrite = FALSE,
```

```
  .progress = TRUE,
  token = arc_token()
)
```

Arguments

x	an object of class FeatureLayer, Table, or ImageServer.
definition_expression	default 1 = 1. A SQL where clause that is applied to the layer. Only those records that conform to this expression will be returned. This parameter is required if neither object_ids or global_ids have been defined.
attachments_definition_expression	default NULL. A SQL where calsue that is applied to the attachment metadata. only attachments that conform to this expression will be returned.
object_ids	mutually exclusive with definition_expression and global_ids. The object IDs of the features to query attachments of.
global_ids	mutally exclusive with definition_expression and object_ids. The global IDs of the features to query attachments of.
attachment_types	default NULL. A character vector of attachment types to filter on.
keywords	default NULL. A character vector of the keywords to filter on.
return_metadata	default TRUE. Returns metadata stored in the exifInfo field.
...	unused
token	your authorization token.
attachments	a data.frame created by query_layer_attachments(). Must contain the columns name, url, and contentType.
out_dir	the path to the folder to download the file
overwrite	default FALSE. A
.progress	default TRUE. Whether a progress bar should be provided.

Value

query_layer_attachments() returns a data.frame.

download_attachments() returns a list. If an error occurs, the condition is captured and returned in the list. Otherwise the path to the file that was downloaded is returned.

References

ArcGIS REST API Documentation

Examples

```
## Not run:
# create a url path that isn't too wide for CRAN
furl <- paste(
  c(
    "https://services1.arcgis.com/hLJbHVT9ZrDIzK0I",
    "arcgis/rest/services/v8_Wide_Area_Search_Form_Feature_Layer___a2fe9c",
    "FeatureServer/0"
  ),
  collapse = "/"
)
# connect to the layer
layer <- arc_open(furl)

# get the attachment info
att <- query_layer_attachments(layer)

# download them to a path
download_attachments(att, "layer_attachments")

## End(Not run)
```

`truncate_layer` *Truncate a Feature Layer*

Description

Removes all features in a Feature Layer or Table and resets the object ID counter. Truncating a Feature Layer does not change the schema of the data (does not add, remove, or alter existing database columns, constraints, or indexes).

Usage

```
truncate_layer(x, async = FALSE, attachment_only = FALSE, token = arc_token())
```

Arguments

`x`	an object of class `FeatureLayer`, `Table`, or `ImageServer`.
`async`	default `FALSE`. It is recommended to set `TRUE` for larger datasets.
`attachment_only`	default `FALSE`. Deletes all the attachments for this layer. None of the layer features will be deleted when `TRUE`.
`token`	your authorization token.

Value

a named list with the name "success" and a value of `TRUE` or `FALSE`

References

ArcGIS Developers Rest API Doc

Examples

```
## Not run:

  # authorize using code flow
  set_arc_token(auth_code())

  # create a FeatureLayer object
  flayer <- arc_open("your-feature-layer-url")

  # truncate it
  truncate_layer(flayer)

## End(Not run)
```

---

update_params *Modify query parameters*

Description

`update_params()` takes named arguments and updates the query.

Usage

```
update_params(x, ...)
```

Arguments

x	a `FeatureLayer` or `Table` object
...	key value pairs of query parameters and values.

Value

An object of the same class as `x`

Examples

```
## Not run:
furl <- paste0(
  "https://services.arcgis.com/P3ePLMYs2RVChkJx/ArcGIS/rest/services/",
  "USA_Major_Cities_/FeatureServer/0"
)

flayer <- arc_open(furl)
update_params(flayer, outFields = "NAME")

## End(Not run)
```

Index