

# Package ‘arcpbf’

July 10, 2024

**Title** Process ArcGIS Protocol Buffer FeatureCollections

**Version** 0.1.3

**Description** Fast processing of ArcGIS FeatureCollection protocol buffers in R.  
It is designed to work seamlessly with 'htr2' and integrates with 'sf'.

**License** Apache License (>= 2)

**URL** <https://r.esri.com/arcpbf/>, <https://github.com/R-ArcGIS/arcpbf>

**BugReports** <https://github.com/R-ArcGIS/arcpbf/issues>

**Encoding** UTF-8

**Language** en

**RoxygenNote** 7.2.3

**Config/rextendr/version** 0.3.1.9000

**SystemRequirements** Cargo (Rust's package manager), rustc

**Suggests** htr2, sf, testthat (>= 3.0.0)

**Imports** arcgisutils (>= 0.3.0), rlang

**Config/testthat/edition** 3

**NeedsCompilation** yes

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

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

**Repository** CRAN

**Date/Publication** 2024-07-10 07:40:06 UTC

## Contents

open_pbf . . . . .	2
post_process_pbf . . . . .	2
process_pbf . . . . .	3
read_pbf . . . . .	5
resp_body_pbf . . . . .	6
<b>Index</b>	<b>8</b>

open\_pbf *Read a pbf file as a raw vector*

---

### Description

Read a pbf file as a raw vector

### Usage

```
open_pbf(path)
```

### Arguments

path            the path to the .pbf file.

### Value

a raw vector

### Examples

```
count_fp <- system.file("count.pbf", package = "arcpbf")
oid_fp <- system.file("ids.pbf", package = "arcpbf")
tbl_fp <- system.file("small-table.pbf", package = "arcpbf")
fc_fp <- system.file("small-points.pbf", package = "arcpbf")
count_raw <- open_pbf(count_fp)
oid_raw <- open_pbf(oid_fp)
tbl_raw <- open_pbf(tbl_fp)
fc_raw <- open_pbf(fc_fp)
```

---

post\_process\_pbf *Post process pbf results*

---

### Description

Applies post-processing to the results of process\_pbf()

### Usage

```
post_process_pbf(x, use_sf = TRUE)
```

### Arguments

x                    an object as returned by process\_pbf() or read\_pbf()  
use\_sf                default TRUE. Whether or not to return an sf object.

**Details**

If `x` is a list object, the results will be row-binded. This is appropriate if each element in the list is a `data.frame` or a feature result with geometry. However, if each element is *not* the same, the post-processing *will* error. If you cannot be certain that all elements that you will be post processing will be the same, post-process each list element independently.

**Value**

An object of class `data.frame`, `sf`, or a scalar integer vector.

See [process\\_pbf\(\)](#) for more details.

**Examples**

```
tbl_fp <- system.file("small-table.pbf", package = "arcpbf")
fc_fp <- system.file("small-points.pbf", package = "arcpbf")

# table feature collection
fc <- read_pbf(tbl_fp)
head(post_process_pbf(fc))

# feature collection with geometry
fc <- read_pbf(fc_fp)
head(post_process_pbf(fc))
```

---

process\_pbf

*Process a FeatureCollection PBF*

---

**Description**

Process a pbf from a raw vector or a list of raw vectors.

**Usage**

```
process_pbf(proto)
```

**Arguments**

`proto` either a raw vector or a list of raw vectors containing a `FeatureCollection pbf`

**Details**

There are three types of PBF `FeatureCollection` responses that may be returned.

**Feature Result:**

In the case the PBF is a `FeatureResult` and `use_sf = FALSE`, a `data.frame` is returned with the spatial reference stored in the `crs` attribute. Otherwise an `sf` object is returned.

**Count Result:**

The PBF can also return a count result, for example if the **query parameter** `returnCountOnly` is set to `true`. In this case, a scalar integer vector is returned.

**Object ID Result:**

In the case that the query parameter `returnIdsOnly` is `true`, a `data.frame` is returned containing the object IDs and the column name set to the object ID field name in the feature service.

**Value**

- For count results, a scalar integer.
- For object ID results a `data.frame` with one column.
- For pbf's that contain geometries, a list of 3 elements:
  - `attributes` is a `data.frame` of the fields of the `FeatureCollection`
  - `geometry` is an `sf` object **without a computed bounding box or coordinate reference system set**
  - `sr` is a named list of the spatial reference of the feature collection

**Important:** Use `post_process_pbf()` to convert to an `sf` object with a computed bounding box and CRS.

**Examples**

```
count_fp <- system.file("count.pbf", package = "arcpbf")
oid_fp <- system.file("ids.pbf", package = "arcpbf")
tbl_fp <- system.file("small-table.pbf", package = "arcpbf")
fc_fp <- system.file("small-points.pbf", package = "arcpbf")

# count response
count_raw <- open_pbf(count_fp)
process_pbf(count_raw)

# object id response
oid_raw <- open_pbf(oid_fp)
head(process_pbf(oid_raw))

# table feature collection
tbl_raw <- open_pbf(tbl_fp)
process_pbf(tbl_raw)

# feature collection with geometry
fc_raw <- open_pbf(fc_fp)
process_pbf(fc_raw)
```

---

`read_pbf`*Read a FeatureCollection Protocol Buffer*

---

**Description**

Given a binary file containing a FeatureCollection protocol buffer (pbf), read its contents into R as an R object.

**Usage**

```
read_pbf(path, post_process = TRUE, use_sf = TRUE)
```

**Arguments**

<code>path</code>	a scalar character of the path to the pbf file
<code>post_process</code>	default TRUE. Apply <code>post_process_pbf()</code> to the pbf body.
<code>use_sf</code>	default TRUE. Whether or not to return an sf object.

**Value**

Either a data.frame, list, scalar integer, or sf object if `post_process = TRUE` and `use_sf = TRUE`. See [process\\_pbf\(\)](#) for more.

**Examples**

```
count_fp <- system.file("count.pbf", package = "arcpbf")
oid_fp <- system.file("ids.pbf", package = "arcpbf")
tbl_fp <- system.file("small-table.pbf", package = "arcpbf")
fc_fp <- system.file("small-points.pbf", package = "arcpbf")

# count response
read_pbf(count_fp)

# object id response
head(read_pbf(oid_fp))

# table feature collection
read_pbf(tbl_fp)

# feature collection with geometry
read_pbf(fc_fp)
```

---

resp\_body\_pbf                      *Extract PBFs from httr2\_response objects*

---

### Description

Processes httr2\_response objects that return FeatureCollection PBFs.

### Usage

```
resp_body_pbf(resp, post_process = TRUE, use_sf = TRUE)
```

```
resps_data_pbf(resps, post_process = TRUE, use_sf = TRUE)
```

### Arguments

resp	A response object.
post_process	default TRUE. Apply post_process_pbf() to the pbf body.
use_sf	default TRUE. Whether or not to return an sf object.
resps	a list of httr2_response objects such as created by httr2::req_perform_parallel()

### Details

Responses of type application/x-protobuf are automatically processed using process\_pbf() with optional post-processing applied. These functions assume that the body of the responses are an Esri FeatureCollection protocol buffer.

#### Lists of responses:

When running multiple requests in parallel using `httr2::req_perform_parallel()` the responses are returned as a list of responses. `resps_data_pbf()` processes the responses in a vectorized manner.

Results are post-processed by default and return sf objects if applicable. This may not be desirable if heterogeneous response types are expected. For example, if one list element contains a count result and another contains an object ID result.

See `post_process_pbf()` for more details.

Note: Knowledge Graph protocol buffers and other protobuf formats are not supported and will result in an error if used with these functions.

### Value

A processed FeatureCollection pbf. Either a scalar integer, named list, data.frame, or an sf object if post-processing is applied.

**Examples**

```
if (rlang::is_installed(c("httr2", "sf")) && interactive()) {
  base_url <- file.path(
    "https://services.arcgis.com/P3ePLMYS2RVChkJx",
    "arcgis", "rest", "services",
    "ACS_Population_by_Race_and_Hispanic_Origin_Boundaries",
    "FeatureServer", "2", "query", fsep = "/"
  )

  # create the base request
  req <- httr2::request(base_url)

  # fill query parameters
  req <- httr2::req_url_query(
    req,
    where = "1=1",
    outFields = "objectid",
    resultRecordCount = 1,
    f = "pbf"
  )

  # make the request
  resp <- httr2::req_perform(req)

  # parse the request
  resp_body_pbf(resp)

  # simulate response from multi_req_perform
  resps <- list(resp, resp, resp)

  # process them all at once
  resps_data_pbf(resps)
}
```

# Index

`httr2::req_perform_parallel()`, 6

`open_pbf`, 2

`post_process_pbf`, 2

`post_process_pbf()`, 4, 6

`process_pbf`, 3

`process_pbf()`, 3, 5

`read_pbf`, 5

`resp_body_pbf`, 6

`resps_data_pbf (resp_body_pbf)`, 6