Package: nanoparquet (via r-universe)

July 2, 2024

Title Read and Write 'Parquet' Files

Version 0.3.1.9000

Description Self-sufficient reader and writer for flat 'Parquet' files. Can read most 'Parquet' data types. Can write many 'R' data types, including factors and temporal types. See docs for limitations.

Depends R (>= 4.0.0)

License MIT + file LICENSE

URL https://github.com/r-lib/nanoparquet, https://r-lib.github.io/nanoparquet/

BugReports https://github.com/r-lib/nanoparquet/issues

Encoding UTF-8

Suggests arrow, bit64, DBI, duckdb, hms, mockery, pillar, processx, rprojroot, spelling, testthat, withr

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Config/testthat/edition 3

Config/Needs/website tidyverse/tidytemplate

Language en-US

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

RemoteUrl https://github.com/r-lib/nanoparquet

RemoteRef HEAD

RemoteSha 163c7249ddde84a7f9956bd01060da384deec6f9

Contents

`nanoparquet-package` *nanoparquet: Read and Write 'Parquet' Files*

Description

Self-sufficient reader and writer for flat 'Parquet' files. Can read most 'Parquet' data types. Can write many 'R' data types, including factors and temporal types. See docs for limitations.

Details

`nanoparquet` is a reader and writer for a common subset of Parquet files.

Features::

- Read and write flat (i.e. non-nested) Parquet files.
- Can read most Parquet data types.
- Can write many R data types, including factors and temporal types to Parquet.
- Completely dependency free.
- Supports Snappy, Gzip and Zstd compression.

Limitations::

- Nested Parquet types are not supported.
- Some Parquet logical types are not supported: `FLOAT16`, `INTERVAL`, `UNKNOWN`.
- Only Snappy, Gzip and Zstd compression is supported.
- Encryption is not supported.
- Reading files from URLs is not supported.
- Being single-threaded and not fully optimized, nanoparquet is probably not suited well for large data sets. It should be fine for a couple of gigabytes. Reading or writing a ~250MB file that has 32 million rows and 14 columns takes about 10-15 seconds on an M2 MacBook Pro. For larger files, use Apache Arrow or DuckDB.

Installation:

Install the R package from CRAN:

```
install.packages("nanoparquet")
```

Usage:

Read:
Call `read_parquet()` to read a Parquet file:

```
df <- nanoparquet::read_parquet("example.parquet")
```

To see the columns of a Parquet file and how their types are mapped to R types by `read_parquet()`, call `parquet_column_types()` first:

```
nanoparquet::parquet_column_types("example.parquet")
```

Folders of similar-structured Parquet files (e.g. produced by Spark) can be read like this:

```
df <- data.table::rbindlist(lapply(
  Sys.glob("some-folder/part-*.parquet"),
  nanoparquet::read_parquet
))
```

Write:
Call `write_parquet()` to write a data frame to a Parquet file:

```
nanoparquet::write_parquet(mtcars, "mtcars.parquet")
```

To see how the columns of the data frame will be mapped to Parquet types by `write_parquet()`, call `parquet_column_types()` first:

```
nanoparquet::parquet_column_types(mtcars)
```

Inspect:
Call `parquet_info()`, `parquet_column_types()`, `parquet_schema()` or `parquet_metadata()` to see various kinds of metadata from a Parquet file:

- `parquet_info()` shows a basic summary of the file.
- `parquet_column_types()` shows the leaf columns, these are are the ones that `read_parquet()` reads into R.
- `parquet_schema()` shows all columns, including non-leaf columns.
- `parquet_metadata()` shows the most complete metadata information: file meta data, the schema, the row groups and column chunks of the file.

```
nanoparquet::parquet_info("mtcars.parquet")
nanoparquet::parquet_column_types("mtcars.parquet")
nanoparquet::parquet_schema("mtcars.parquet")
nanoparquet::parquet_metadata("mtcars.parquet")
```

If you find a file that should be supported but isn't, please open an issue here with a link to the file.

Options:

See also `?parquet_options()`.

- `nanoparquet.class`: extra class to add to data frames returned by `read_parquet()`. If it is not defined, the default is `"tbl"`, which changes how the data frame is printed if the pillar package is loaded.
- `nanoparquet.use_arrow_metadata`: unless this is set to `FALSE`, `read_parquet()` will make use of Arrow metadata in the Parquet file. Currently this is used to detect factor columns.
- `nanoparquet.write_arrow_metadata`: unless this is set to `FALSE`, `write_parquet()` will add Arrow metadata to the Parquet file. This helps preserving classes of columns, e.g. factors will be read back as factors, both by nanoparquet and Arrow.

License:

MIT

Author(s)

Maintainer: Gábor Csárdi <csardi.gabor@gmail.com>

Authors:

- Hannes Mühleisen (ORCID) [copyright holder]

Other contributors:

- Google Inc. [copyright holder]
- Apache Software Foundation [copyright holder]
- Posit Software, PBC [copyright holder]
- RAD Game Tools [copyright holder]
- Valve Software [copyright holder]
- Tenacious Software LLC [copyright holder]
- Facebook, Inc. [copyright holder]

See Also

Useful links:

- https://github.com/r-lib/nanoparquet
- https://r-lib.github.io/nanoparquet/
- Report bugs at https://github.com/r-lib/nanoparquet/issues

`nanoparquet-types` *nanoparquet's type maps*

Description

How nanoparquet maps R types to Parquet types.

R's data types

When writing out a data frame, nanoparquet maps R's data types to Parquet logical types. This is how the mapping is performed.

These rules will likely change until nanoparquet reaches version 1.0.0.

1. Factors (i.e. vectors that inherit the *factor* class) are converted to character vectors using `as.character()`, then written as a `STRSXP` (character vector) type. The fact that a column is a factor is stored in the Arrow metadata (see below), unless the `nanoparquet.write_arrow_metadata` option is set to `FALSE`.
2. Dates (i.e. the `Date` class) is written as `DATE` logical type, which is an `INT32` type internally.
3. `hms` objects (from the hms package) are written as `TIME(true, MILLIS)`. logical type, which is internally the `INT32` Parquet type. Sub-milliseconds precision is lost.

4. `POSIXct` objects are written as `TIMESTAMP(true, MICROS)` logical type, which is internally the `INT64` Parquet type. Sub-microsecond precision is lost.
5. `difftime` objects (that are not `hms` objects, see above), are written as an `INT64` Parquet type, and noting in the Arrow metadata (see below) that this column has type `Duration` with `NANOSECONDS` unit.
6. Integer vectors (`INTSXP`) are written as `INT(32, true)` logical type, which corresponds to the `INT32` type.
7. Real vectors (`REALSXP`) are written as the `DOUBLE` type.
8. Character vectors (`STRSXP`) are written as the `STRING` logical type, which has the `BYTE_ARRAY` type. They are always converted to UTF-8 before writing.
9. Logical vectors (`LGLSXP`) are written as the `BOOLEAN` type.
10. Other vectors error currently.

You can use `parquet_column_types()` on a data frame to map R data types to Parquet data types.

Parquet's data types

When reading a Parquet file nanoparquet also relies on logical types and the Arrow metadata (if present, see below) in addition to the low level data types. The exact rules are below.

These rules will likely change until nanoparquet reaches version 1.0.0.

1. The `BOOLEAN` type is read as a logical vector (`LGLSXP`).
2. The `STRING` logical type and the `UTF8` converted type is read as a character vector with UTF-8 encoding.
3. The `DATE` logical type and the `DATE` converted type are read as a `Date` R object.
4. The `TIME` logical type and the `TIME_MILLIS` and `TIME_MICROS` converted types are read as an `hms` object, see the hms package.
5. The `TIMESTAMP` logical type and the `TIMESTAMP_MILLIS` and `TIMESTAMP_MICROS` converted types are read as `POSIXct` objects. If the logical type has the `UTC` flag set, then the time zone of the `POSIXct` object is set to `UTC`.
6. `INT32` is read as an integer vector (`INTSXP`).
7. `INT64`, `DOUBLE` and `FLOAT` are read as real vectors (`REALSXP`).
8. `INT96` is read as a `POSIXct` read vector with the `tzone` attribute set to `"UTC"`. It was an old convention to store time stamps as `INT96` objects.
9. The `DECIMAL` converted type (`FIXED_LEN_BYTE_ARRAY` or `BYTE_ARRAY` type) is read as a real vector (`REALSXP`), potentially losing precision.
10. The `ENUM` logical type is read as a character vector.
11. The `UUID` logical type is read as a character vector that uses the `00112233-4455-6677-8899-aabbccddeeff` form.
12. `BYTE_ARRAY` is read as a *factor* object if the file was written by Arrow and the original data type of the column was a factor. (See 'The Arrow metadata below.)
13. Otherwise `BYTE_ARRAY` is read a list of raw vectors, with missing values denoted by `NULL`.

Other logical and converted types are read as their annotated low level types:

1. `INT(8, true)`, `INT(16, true)` and `INT(32, true)` are read as integer vectors because they are `INT32` internally in Parquet.
2. `INT(64, true)` is read as a real vector (`REALSXP`).
3. Unsigned integer types `INT(8, false)`, `INT(16, false)` and `INT(32, false)` are read as integer vectors (`INTSXP`). Large positive values may overflow into negative values, this is a known issue that we will fix.
4. `INT(64, false)` is read as a real vector (`REALSXP`). Large positive values may overflow into negative values, this is a known issue that we will fix.
5. `FLOAT16` is a fixed length byte array, and nanoparquet reads it as a list of raw vectors. Missing values are denoted by `NULL`.
6. `INTERVAL` is a fixed length byte array, and nanoparquet reads it as a list of raw vectors. Missing values are denoted by `NULL`.
7. `JSON` and `BSON` are read as character vectors (`STRSXP`).

These types are not yet supported:

1. Nested types (`LIST`, `MAP`) are not supported.
2. The `UNKNOWN` logical type is not supported.

You can use the `parquet_column_types()` function to see how R would read the columns of a Parquet file. Look at the `r_type` column.

The Arrow metadata

Apache Arrow (i.e. the arrow R package) adds additional metadata to Parquet files when writing them in `arrow::write_parquet()`. Then, when reading the file in `arrow::read_parquet()`, it uses this metadata to recreate the same Arrow and R data types as before writing.

`nanoparquet::write_parquet()` also adds the Arrow metadata to Parquet files, unless the `nanoparquet.write_arrow_me` option is set to `FALSE`.

Similarly, `nanoparquet::read_parquet()` uses the Arrow metadata in the Parquet file (if present), unless the `nanoparquet.use_arrow_metadata` option is set to FALSE.

The Arrow metadata is stored in the file level key-value metadata, with key `ARROW:schema`.

Currently nanoparquet uses the Arrow metadata for two things:

- It uses it to detect factors. Without the Arrow metadata factors are read as string vectors.
- It uses it to detect `difftime` objects. Without the arrow metadata these are read as `INT64` columns, containing the time difference in nanoseconds.

See Also

nanoparquet-package for options that modify the type mappings.

`parquet_column_types` *Map between R and Parquet data types*

Description

This function works two ways. It can map the R types of a data frame to Parquet types, to see how `write_parquet()` would write out the data frame. It can also map the types of a Parquet file to R types, to see how `read_parquet()` would read the file into R.

Usage

```
parquet_column_types(x, options = parquet_options())
```

Arguments

`x`	Path to a Parquet file, or a data frame.
`options`	Nanoparquet options, see `parquet_options()`.

Value

Data frame with columns:

- `file_name`: file name.
- `name`: column name.
- `type`: (low level) Parquet data type.
- `r_type`: the R type that corresponds to the Parquet type. Might be `NA` if `read_parquet()` cannot read this column. See nanoparquet-types for the type mapping rules.
- `repetition_type`: whether the column in `REQUIRED` (cannot be `NA`) or `OPTIONAL` (may be `NA`). `REPEATED` columns are not currently supported by nanoparquet.
- `logical_type`: Parquet logical type in a list column. An element has at least an entry called `type`, and potentially additional entries, e.g. `bit_width`, `is_signed`, etc.

See Also

`parquet_metadata()` to read more metadata, `parquet_info()` for a very short summary. `parquet_schema()` for the complete Parquet schema. `read_parquet()`, `write_parquet()`, nanoparquet-types.

parquet_info *Short summary of a Parquet file*

Description

Short summary of a Parquet file

Usage

```
parquet_info(file)
```

Arguments

file	Path to a Parquet file.

Value

Data frame with columns:

- file_name: file name.
- num_cols: number of (leaf) columns.
- num_rows: number of rows.
- num_row_groups: number of row groups.
- file_size: file size in bytes.
- parquet_version: Parquet version.
- created_by: A string scalar, usually the name of the software that created the file. NA if not available.

See Also

parquet_metadata() to read more metadata, parquet_column_types() and parquet_schema() for column information. read_parquet(), write_parquet(), nanoparquet-types.

parquet_metadata *Read the metadata of a Parquet file*

Description

This function should work on all files, even if read_parquet() is unable to read them, because of an unsupported schema, encoding, compression or other reason.

Usage

```
parquet_metadata(file)
```

Arguments

file	Path to a Parquet file.

Value

A named list with entries:

- file_meta_data: a data frame with file meta data:
  - file_name: file name.
  - version: Parquet version, an integer.
  - num_rows: total number of rows.
  - key_value_metadata: list column of a data frames with two character columns called key and value. This is the key-value metadata of the file. Arrow stores its schema here.
  - created_by: A string scalar, usually the name of the software that created the file.
- schema: data frame, the schema of the file. It has one row for each node (inner node or leaf node). For flat files this means one root node (inner node), always the first one, and then one row for each "real" column. For nested schemas, the rows are in depth-first search order. Most important columns are:
  - file_name: file name.
  - name: column name.
  - type: data type. One of the low level data types.
  - type_length: length for fixed length byte arrays.
  - repettion_type: character, one of REQUIRED, OPTIONAL or REPEATED.
  - logical_type: a list column, the logical types of the columns. An element has at least an entry called type, and potentially additional entries, e.g. bit_width, is_signed, etc.
  - num_children: number of child nodes. Should be a non-negative integer for the root node, and NA for a leaf node.
- $row_groups: a data frame, information about the row groups.
- $column_chunks: a data frame, information about all column chunks, across all row groups. Some important columns:
  - file_name: file name.
  - row_group: which row group this chunk belongs to.
  - column: which leaf column this chunks belongs to. The order is the same as in $schema, but only leaf columns (i.e. columns with NA children) are counted.
  - file_path: which file the chunk is stored in. NA means the same file.
  - file_offset: where the column chunk begins in the file.
  - type: low level parquet data type.
  - encodings: encodings used to store this chunk. It is a list column of character vectors of encoding names. Current possible encodings: "PLAIN", "GROUP_VAR_INT", "PLAIN_DICTIONARY", "RLE", "BIT_PACKED", "DELTA_BINARY_PACKED", "DELTA_LENGTH_BYTE_ "DELTA_BYTE_ARRAY", "RLE_DICTIONARY", "BYTE_STREAM_SPLIT".
  - path_in_scema: list column of character vectors. It is simply the path from the root node. It is simply the column name for flat schemas.
  - codec: compression codec used for the column chunk. Possible values are: "UNCOMPRESSED", "SNAPPY", "GZIP", "LZO", "BROTLI", "LZ4", "ZSTD".

- num_values: number of values in this column chunk.
- total_uncompressed_size: total uncompressed size in bytes.
- total_compressed_size: total compressed size in bytes.
- data_page_offset: absolute position of the first data page of the column chunk in the file.
- index_page_offset: absolute position of the first index page of the column chunk in the file, or NA if there are no index pages.
- dictionary_page_offset: absolute position of the first dictionary page of the column chunk in the file, or NA if there are no dictionary pages.

See Also

parquet_info() for a much shorter summary. parquet_column_types() and parquet_schema() for column information. read_parquet() to read, write_parquet() to write Parquet files, nanoparquet-types for the R <-> Parquet type mappings.

Examples

```
file_name <- system.file("extdata/userdata1.parquet", package = "nanoparquet")
nanoparquet::parquet_metadata(file_name)
```

---

parquet_options *Nanoparquet options*

Description

Create a list of nanoparquet options.

Usage

```
parquet_options(
  class = getOption("nanoparquet.class", "tbl"),
  use_arrow_metadata = getOption("nanoparquet.use_arrow_metadata", TRUE),
  write_arrow_metadata = getOption("nanoparquet.write_arrow_metadata", TRUE)
)
```

Arguments

class	The extra class or classes to add to data frames created in read_parquet(). By default nanoparquet adds the "tbl" class, so data frames are printed differently if the pillar package is loaded.
use_arrow_metadata	TRUE or FALSE. If TRUE, then read_parquet() and parquet_column_types() will make use of the Apache Arrow metadata to assign R classes to Parquet columns. This is currently used to detect factor columns, and to detect "difftime" columns. If this option is FALSE:

- "factor" columns are read as character vectors.
- "difftime" columns are read as real numbers, meaning one of seconds, milliseconds, microseconds or nanoseconds. Impossible to tell which without using the Arrow metadata.

`write_arrow_metadata`
: Whether to add the Apache Arrow types as metadata to the file `write_parquet()`.

Value

List of nanoparquet options.

Examples

```
# the effect of using Arrow metadata
tmp <- tempfile(fileext = ".parquet")
d <- data.frame(
  fct = as.factor("a"),
  dft = as.difftime(10, units = "secs")
)
write_parquet(d, tmp)
read_parquet(tmp, options = parquet_options(use_arrow_metadata = TRUE))
read_parquet(tmp, options = parquet_options(use_arrow_metadata = FALSE))
```

---

parquet_schema *Read the schema of a Parquet file*

Description

This function should work on all files, even if `read_parquet()` is unable to read them, because of an unsupported schema, encoding, compression or other reason.

Usage

```
parquet_schema(file)
```

Arguments

`file`
: Path to a Parquet file.

Value

```
Data frame, the schema of the file. It has one row for
each node (inner node or leaf node). For flat files this means one
root node (inner node), always the first one, and then one row for
each "real" column. For nested schemas, the rows are in depth-first
search order. Most important columns are:
- `file_name`: file name.
```

- `name`: column name.
- `type`: data type. One of the low level data types.
- `type_length`: length for fixed length byte arrays.
- `repettion_type`: character, one of `REQUIRED`, `OPTIONAL` or `REPEATED`.
- `logical_type`: a list column, the logical types of the columns. An element has at least an entry called `type`, and potentially additional entries, e.g. `bit_width`, `is_signed`, etc.
- `num_children`: number of child nodes. Should be a non-negative integer for the root node, and `NA` for a leaf node.

See Also

parquet_metadata() to read more metadata, parquet_column_types() to show the columns R would read, parquet_info() to show only basic information. read_parquet(), write_parquet(), nanoparquet-types.

---

read_parquet *Read a Parquet file into a data frame*

Description

Converts the contents of the named Parquet file to a R data frame.

Usage

```
read_parquet(file, options = parquet_options())
```

Arguments

file	Path to a Parquet file.
options	Nanoparquet options, see parquet_options().

Value

A data.frame with the file's contents.

See Also

See write_parquet() to write Parquet files, nanoparquet-types for the R <-> Parquet type mapping. See parquet_info(), for general information, parquet_column_types() and parquet_schema() for information about the columns, and parquet_metadata() for the complete metadata.

Examples

```
file_name <- system.file("extdata/userdata1.parquet", package = "nanoparquet")
parquet_df <- nanoparquet::read_parquet(file_name)
print(str(parquet_df))
```

`write_parquet` *Write a data frame to a Parquet file*

Description

Writes the contents of an R data frame into a Parquet file.

Usage

```
write_parquet(
  x,
  file,
  compression = c("snappy", "gzip", "zstd", "uncompressed"),
  metadata = NULL,
  options = parquet_options()
)
```

Arguments

`x`	Data frame to write.
`file`	Path to the output file. If this is the string `":raw:"`, then the data frame is written to a memory buffer, and the memory buffer is returned as a raw vector.
`compression`	Compression algorithm to use. Currently `"snappy"` (the default), `"gzip"`, `"zstd"`, and `"uncompressed"` are supported.
`metadata`	Additional key-value metadata to add to the file. This must be a named character vector, or a data frame with columns character columns called `key` and `value`.
`options`	Nanoparquet options, see `parquet_options()`.

Details

`write_parquet()` converts string columns to UTF-8 encoding by calling `base::enc2utf8()`. It does the same for factor levels.

Value

`NULL`, unless `file` is `":raw:"`, in which case the Parquet file is returned as a raw vector.

See Also

`parquet_metadata()`, `read_parquet()`.

Examples

```
# add row names as a column, because `write_parquet()` ignores them.
mtcars2 <- cbind(name = rownames(mtcars), mtcars)
write_parquet(mtcars2, "mtcars.parquet")
```

Index