Package: Rzarrs (via r-universe)

October 18, 2026

Type Package

Title R Bindings to 'zarrs': Zarr V3 Storage for Multidimensional Arrays

Depends R (>= 4.4.0)

Version 0.1.0

Description R bindings to the 'zarrs' Rust library <https://github.com/zarrs/zarrs> for reading Zarr V3 (and compatible Zarr V2) stores from local disk, HTTP/HTTPS, and S3 by default, with optional GCS and Azure Blob support at source-install time. Supports local '.zarr.zip' VCF Zarr archives, multiple codecs (gzip, zstd, sharding), and the common numeric data types. SIMD-accelerated codec paths are selected automatically at runtime by the underlying Rust dependency crates.

License GPL (>= 3)

URL https://github.com/RGenomicsETL/Rzarrs

BugReports https://github.com/RGenomicsETL/Rzarrs/issues

SystemRequirements Cargo (Rust's package manager), rustc >= 1.82.0, GNU Make

Suggests tinytest, Rarr, bench, optparse, rmarkdown

Additional_repositories https://bioconductor.org/packages/3.22/bioc

Encoding UTF-8

RoxygenNote 7.3.3

Config/pak/sysreqs make libclang-dev

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

Date/Publication 2026-09-07 20:17:18 UTC

RemoteUrl https://github.com/RGenomicsETL/Rzarrs

RemoteRef HEAD

RemoteSha 67a6799cc87058658ceb60248ebe05220c7b187c

Contents

codec_capabilities 2
dtype_plan 2
ZarrArray 3
ZarrGroup 4
ZarrObjectStore 5
ZarrStore 6
ZarrVcf 6

Index **8**

`codec_capabilities` *Report Rzarrs codec capabilities*

Description

With no argument, returns the codecs compiled into this Rzarrs build. When passed a `ZarrArray`, returns support status for the codecs declared by that array, including nested sharding codecs.

Usage

```
codec_capabilities(x = NULL)
```

Arguments

`x`	Optional `ZarrArray`.

Value

A named list with `codec` and `supported` vectors.

`dtype_plan` *Plan R materialization for a Zarr dtype*

Description

Plan R materialization for a Zarr dtype.

Usage

```
dtype_plan(dtype)
```

Arguments

`dtype`	Character scalar dtype name.

Details

High-precision extension dtypes such as `float128`, `decimal128`, and `decimal256` are planned as lossy R `double` values. Actual reading still depends on a registered binary-layout data type materializer because these are extension dtypes, not core built-in zarrs element types.

Value

A named list describing the planned R materialization.

`ZarrArray` *Zarr array handle*

Description

A handle to a single Zarr array within a store. Arrays can be opened from a local store (`ZarrStore`) or an object-store backend (`ZarrObjectStore`).

Indexing follows R conventions: indices are **1-based and inclusive** on both ends.

Constructors

`ZarrArray$open(store, path)` Open from a local `ZarrStore`.

`store` A `ZarrStore` object.

`path` Character scalar. Array path within the store, e.g. `"/"` for the root array.

`ZarrArray$open_object_store(store, path)` Open from a `ZarrObjectStore`. Same arguments as above but `store` is a `ZarrObjectStore`.

Methods

`$ndim()` Number of dimensions (integer scalar).

`$shape()` Array shape (integer vector, one element per dimension).

`$chunk_shape()` Chunk shape (integer vector), or `NULL` for irregular chunk grids.

`$dtype()` Data type name in Zarr V3 canonical form, e.g. `"float32"`, `"int32"`, `"bool"` (character scalar).

`$dimension_names()` Dimension names (character vector), or `NULL` if none are stored.

`$metadata_json()` Full array metadata as pretty-printed JSON (character scalar).

`$retrieve(starts = NULL, ends = NULL)` Read array data into R.

`starts` Integer vector of **1-based** start indices (one per dimension), or `NULL` to start at the first element.

`ends` Integer vector of **1-based inclusive** end indices (one per dimension), or `NULL` to include the last element.

Returns a vector of the appropriate R type (`integer`, `double`, or `logical`) with a `dim` attribute set to the shape of the requested region. Data are returned in R (Fortran / column-major) order so that standard R array indexing works as expected.

See Also

ZarrStore, ZarrObjectStore

Examples

```
zarr_path <- system.file("testdata", "int32.zarr", package = "Rzarrs")
arr <- ZarrArray$open(ZarrStore$open(zarr_path), "/")

arr$ndim()        # 2
arr$shape()       # c(4L, 6L)
arr$dtype()       # "int32"

# Full array
full <- arr$retrieve()
dim(full)         # c(4L, 6L)
full[1, 6]        # 6

# Subset: rows 1-2, cols 1-3
sub <- arr$retrieve(c(1L, 1L), c(2L, 3L))
dim(sub)          # c(2L, 3L)
```

`ZarrGroup` *A handle to a Zarr group (root or sub-group) within a store.*

Description

Groups may contain arrays and/or sub-groups. Use '$attributes()' to read the group's JSON attributes as a native R list, and '$children()' to list the immediate child nodes.

Usage

```
ZarrGroup
```

Format

An object of class `Rzarrs::ZarrGroup__bundle` (inherits from `savvy_Rzarrs__sealed`) of length 2.

ZarrObjectStore *Object-store Zarr backend (S3, GCS, Azure, HTTP/HTTPS)*

Description

A handle to a Zarr store backed by any URL scheme supported by the object_store crate: `s3://`, `gs://`, `az://`, `https://`, or `file:///`. URLs ending in `.zarr.zip` or `.zip` are opened as zip objects when zip support is enabled.

Credentials are read automatically from the standard environment variables for each provider (the same ones used by the AWS CLI, `gsutil`, `azcopy`, etc.). No credentials are ever passed as R objects.

Constructor

`ZarrObjectStore$open(url)` Open an object-store backend.

`url` Character scalar. Store URL, e.g. `"s3://my-bucket/path/to/store.zarr"`, `"s3://my-bucket/path/to/stor` or `"https://example.com/store.zarr"`.

Returns a `ZarrObjectStore` object.

Methods

`$url()` Return the URL passed to `open()` as a character scalar.

Credentials

Set the relevant environment variables *before* calling `open()`:

AWS S3 `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_REGION`, `AWS_ENDPOINT_URL`

Google GCS `GOOGLE_APPLICATION_CREDENTIALS`

Azure Blob `AZURE_STORAGE_ACCOUNT`, `AZURE_STORAGE_ACCESS_KEY`

See Also

ZarrStore, ZarrArray

Examples

```
## Not run:
store <- ZarrObjectStore$open("s3://my-bucket/my.zarr")
arr   <- ZarrArray$open_object_store(store, "/")
arr$shape()

## End(Not run)
```

ZarrStore *Local Zarr filesystem store*

Description

A handle to a Zarr store rooted at a local directory. Use `ZarrStore$open(path)` to create one, then pass it to `ZarrArray$open(store, path)` to open individual arrays.

Constructor

`ZarrStore$open(path)` Open a local Zarr store.

`path` Character scalar. Path to the '`.zarr`' directory.

Returns a `ZarrStore` object.

Methods

`$path()` Return the store root path as a character scalar.

See Also

`ZarrObjectStore`, `ZarrArray`

Examples

```
zarr_path <- system.file("testdata", "int32.zarr", package = "Rzarrs")
store <- ZarrStore$open(zarr_path)
store$path()
```

ZarrVcf *High-level VCF Zarr reader*

Description

'ZarrVcf$open(x)' opens a VCF Zarr store (spec versions 0.1–0.4) and returns an instance with methods to access variant, sample, and genotype data. Genotypes use the VCF Zarr integer encoding: allele indexes are 0-based, '-1' is a missing allele ('.' in VCF), and '-2' is the array fill sentinel.

Usage

`ZarrVcf`

Format

An object of class `Rzarrs::ZarrVcf__bundle` (inherits from `savvy_Rzarrs__sealed`) of length 3.

Methods

‘$open(x)‘ Open a VCF Zarr store from a path, URL, ‘ZarrStore‘, or ‘ZarrObjectStore‘.

‘$version()‘ VCF Zarr spec version string.

‘$n_variants()‘, ‘$n_samples()‘ Number of variants and samples.

‘$samples()‘, ‘$contigs()‘, ‘$filters()‘ Character vectors of sample IDs, contig names, filter IDs.

‘$fields()‘ Available array names.

‘$variant_position()‘, ‘$variant_contig()‘, ‘$variant_allele()‘ Per-variant data.

‘$genotypes(variants, samples)‘ 3-D array (variants × samples × ploidy) of integer genotypes using VCF Zarr ‘-1‘/‘-2‘ sentinel values.

‘$call_genotype_phased(variants, samples)‘ Boolean phased matrix.

‘$variant(name)‘, ‘$call(name)‘ Generic accessor for ‘variant_<name>‘ / ‘call_<name>‘ arrays.

Index

∗ **datasets**
ZarrGroup, 4
ZarrVcf, 6

codec_capabilities, 2

dtype_plan, 2

ZarrArray, 3, *5, 6*
ZarrGroup, 4
ZarrObjectStore, *3, 4*, 5, *6*
ZarrStore, *3–5*, 6
ZarrVcf, 6