Package 'geozarr'

September 16, 2026

Title GeoZarr Conventions for Geospatial Data in Zarr Stores

Version 0.2.0

Description Large-scale gridded data stores are increasingly using the Zarr format. GeoZarr is defined in terms of a number of community conventions built on top of the Zarr specification. These conventions specify how Zarr metadata is to be interpreted to attach semantic meaning to the data in the Zarr array providing the metadata. This package implements a number of community conventions on top of the 'zarr' package, enabling the R community to use geospatial data stored in Zarr.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 8.0.0

Imports CFtime (>= 1.7.3), R6

Depends R (>= 4.2), zarr (>= 0.5.1)

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/R-CF/geozarr

BugReports https://github.com/R-CF/geozarr/issues

NeedsCompilation no

Author Patrick Van Laake [aut, cre]

Maintainer Patrick Van Laake <patrick@vanlaake.net>

Repository CRAN

Date/Publication 2026-09-16 00:10:02 UTC

Contents

`as_geozarr` *Convert an R object into a GeoZarr array*

Description

This function creates a GeoZarr object from an R matrix or array. A GeoZarr object is like a Zarr object but with special attributes to establish a coordinate system. Default settings will be taken from the R object (data type, shape). Data is chunked into chunks of length 100 (or less if the array is smaller) and compressed.

Usage

```
as_geozarr(x, name = NULL, location = NULL, registration = "pixel")
```

Arguments

`x`	The R object to convert. Must be a matrix or array of a numeric or logical type.
`name`	Optional. The name of the GeoZarr array to be created. If omitted, an array will be created at the root of the Zarr store.
`location`	Optional. If supplied, either an existing zarr_group in a zarr object, or a character string giving the location on a local file system where to persist the data. If the argument is a `zarr_group`, argument `name` must be provided. If the argument gives the location for a new Zarr store then the location must be writable by the calling code. As per the Zarr specification, it is recommended to use a location that ends in ".zarr" when providing a location for a new store. If argument `name` is given then the `geozarr_array` will be created in the root of the `zarr` store with that name. If the `name` argument is not given, a single-array Zarr store will be created. If the `location` argument is not given, a `zarr` object is created in memory.

registration Either "pixel" (the default) or "node". Pixel registration interprets the coordinates in the "dimnames" of argument x as being the upper-left corner of each grid cell. Node registration interprets them as the centers of grid cells. In both cases the elements in the array are assumed to represent an area.

Details

Depending on the properties of the R object, the GeoZarr object may use the "spatial" or "cs" convention for encoding. The "spatial" encoding is the most compact and it will be used for R objects that have at least X and Y dimensions, identified by the names set on the dimensions, and an optional third axis which is typically an image band or a discrete (class) axis – the third axis may not represent height/depth (Z) or time (T). While the "spatial" convention does work on Zarr arrays with more dimensions, there is no mechanism to attach coordinates to any additional axes. The coordinates must be numeric and regularly spaced and the Y coordinates must be decreasing. In other words, the "spatial" convention will be used for imagery style, north-up arrays with a coordinate system tied to the top-left corner of the array space. For all other cases the "cs" convention will be used.

If the coordinates along the axes (the dimnames of the R object) are not regularly spaced, secondary Zarr arrays will be created in the same group as the main Zarr array with the axis coordinates, if the length of the axis is longer than the option GeoZarr.options$max_explicit – shorter sets of coordinates are stored in the Zarr array cs attributes.

Any time coordinates will be converted to a CFtime format with a reference of "days since 1970-01-01", compatible with the standard system clock.

For more exacting requirements, you should manually construct the GeoZarr object from R objects.

Value

If the location argument is a zarr_group, the new geozarr_array instance is returned. Otherwise, the zarr object that is newly created and which contains the GeoZarr array in the root group, or an error if the zarr object could not be created.

Examples

```
x <- array(1:400, c(5, 20, 4))
dimnames(x) <- list(x = 100000 + 0:4 * 10000, y = 19:0 * 5000, cls = letters[1:4])
z <- as_geozarr(x, "my_data")
z
```

Coordinates *Coordinates*

Description

This class implements the coordinates class. The coordinates are always associated with a coordinate system axis.

This class provides the basic interface for coordinate values and it manages numeric coordinate values, both integer and floating-point. Descendant classes manage packed numerical values, character values and time values.

The optional boundary values for numeric coordinate values that define the finite extent of each element in the coordinates are also managed by this class.

Super class

`zarr::zarr_object` -> `Coordinates`

Active bindings

`name` Set or retrieve the name of the coordinate set.

`direction` Set or retrieve the direction of the coordinates.

`values` (read-only) Retrieve the coordinate values as a full vector of values.

`raw` (read-only) Retrieve the coordinate values as stored, either a full vector or packed.

`bounds` Set or retrieve the boundary values around coordinates. When setting, assign a matrix with 2 rows and as many columns as there are coordinates values. When retrieving, a matrix as just described or `NULL` if boundary values have not been set.

`bounds_raw` (read-only) Retrieve the boundary values around coordinates exactly as they are stored with the coordinate. If the boundary values are packed, a vector of length 2 is returned, otherwise a matrix with 2 rows and as many columns as there are coordinates values or `NULL` if boundary values have not been set.

`range` (read-only) Retrieve the extreme values of the coordinates.

`length` (read-only) The number of elements in this instance once the values are unpacked.

`unit` Character string giving the unit-of-measure of the coordinate values.

`attributes` (read-only) A named `list` with the attributes of this object.

Methods

Public methods:

- `Coordinates$new()`
- `Coordinates$print()`
- `Coordinates$brief()`
- `Coordinates$copy()`
- `Coordinates$subset()`
- `Coordinates$slice()`
- `Coordinates$getDirection()`
- `Coordinates$getUnit()`
- `Coordinates$getMinimumValue()`
- `Coordinates$getMaximumValue()`
- `Coordinates$getRangeMeaning()`

`Coordinates$new()`**:** Create a new coordinates instance for use with a coordinate system axis.

Usage:

```
Coordinates$new(
  name,
  direction,
  unit,
  values,
  bounds = NULL,
  attributes = list()
)
```

Arguments:

`name` Character string. Name of the coordinate set.

`direction` Character string. Direction of the coordinates. Must be one from the set of values given in `AxisDirection`.

`unit` Character string. Unit of measure of the coordinates.

`values` A vector of values.

`bounds` Optional. If the boundaries are regularly spaced, a vector with the offset to the boundary lower and higher than the coordinate value, respectively. Otherwise, a matrix with two rows and as many columns as there are elements, with the offset to the boundary below the coordinate value in row 1 and the offset to the boundary above the coordinate value in row 2. The boundaries represent the finite extent around the boundary value that this value is representative for. If this argument is not provided, the coordinate values are assumed to represent a point value.

`attributes` Optional. A `list` of attributes of the coordinates.

Returns: An instance of this class or an error.

`Coordinates$print()`: Print a summary of the coordinates to the console.

Usage:

```
Coordinates$print(...)
```

Arguments:

`...` Ignored.

Returns: Self, invisible.

`Coordinates$brief()`: Some details of the axis.

Usage:

```
Coordinates$brief()
```

Returns: A 1-row `data.frame` with some details of the axis.

`Coordinates$copy()`: Return an exact copy of these coordinates.

Usage:

```
Coordinates$copy()
```

Returns: A new instance of `Coordinates`.

`Coordinates$subset()`: Return coordinates spanning a smaller coordinate range.

Usage:

`Coordinates$subset(rng)`

Arguments:

`rng` The range of indices to include in the returned coordinates.

Returns: A new `Coordinates` instance covering the indicated range of indices, including boundary values, present.

`Coordinates$slice()`: Given a range of domain coordinate values, returns the indices into the axis that fall within the supplied range. If the axis has boundary values, any coordinate whose boundary values fall entirely or partially within the supplied range will be included in the result.

Usage:

`Coordinates$slice(rng)`

Arguments:

`rng` A numeric vector whose extreme values indicate the indices of coordinates to return.

Returns: An integer vector of length 2 with the lower and higher indices into the axis that fall within the range of coordinates in argument `rng`. Returns `NULL` if no (boundary) values of the axis fall within the range of coordinates.

`Coordinates$getDirection()`: Retrieve the direction of the coordinates. This method is mandatory in the OGC standard for an axis.

Usage:

`Coordinates$getDirection()`

Returns: Character string with the direction of the coordinates.

`Coordinates$getUnit()`: Retrieve the unit of measure of the coordinates. This method is mandatory in the OGC standard for an axis.

Usage:

`Coordinates$getUnit()`

Returns: Character string with the unit of measure of the axis.

`Coordinates$getMinimumValue()`: Retrieve the minimum coordinate value of this set of coordinates, in units of the unit of measure of the coordinates.

Usage:

`Coordinates$getMinimumValue()`

Returns: Negative infinity.

`Coordinates$getMaximumValue()`: Retrieve the maximum coordinate value of this set of coordinates, in units of the unit of measure of the coordinates.

Usage:

`Coordinates$getMaximumValue()`

Returns: Positive infinity.

`Coordinates$getRangeMeaning()`: Retrieve the interpretation of coordinate values, either as a direct value ("EXACT") or as a wrap-around value between the minimum and maximum value of the coordinates ("WRAPAROUND").

Usage:

`Coordinates$getRangeMeaning()`

Returns: The character string 'EXACT'.

`CoordinatesOrdinal` *Ordinal coordinate values*

Description

This class implements ordinal values. Ordinal values are typically assigned to axes that have no other coordinates assigned to them. By default, ordinal values start at 0 (as per the Zarr specification) but after subsetting or other forms of selection the value may be different.

Super classes

`zarr::zarr_object` -> `Coordinates` -> `CoordinatesPacked` -> `CoordinatesOrdinal`

Methods

Public methods:

- `CoordinatesOrdinal$new()`
- `CoordinatesOrdinal$copy()`
- `CoordinatesOrdinal$subset()`
- `CoordinatesOrdinal$clone()`

`CoordinatesOrdinal$new()`: Create an instance of this class.

Usage:

```
CoordinatesOrdinal$new(name, direction, length, low = 0L, attributes = list())
```

Arguments:

`name` Character string. Name of the coordinate set.

`direction` Character string. Direction of the coordinates. Must be one from a set of values.

`length` Integer value giving the number of elements in this instance.

`low` Optional. Integer value giving the lowest index value in this instance. When omitted, defaults to 0L.

`attributes` Optional. A `list` of attributes of the coordinates.

Returns: An instance of this class.

`CoordinatesOrdinal$copy()`: Return an exact copy of these coordinates.

Usage:

```
CoordinatesOrdinal$copy()
```

Returns: A new instance of `CoordinatesOrdinal`.

`CoordinatesOrdinal$subset()`: Return a subset of the coordinate values.

Usage:

```
CoordinatesOrdinal$subset(rng)
```

Arguments:

`rng` The range of indices whose values from these coordinate values to include in the result.

Returns: A new `CoordinatesOrdinal` instance covering the indicated range of values.

`CoordinatesOrdinal$clone()`: The objects of this class are cloneable with this method.

Usage:

```
CoordinatesOrdinal$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CoordinatesPacked` *Packed coordinates*

Description

This class implements the packed coordinates class. The coordinates are always associated with a coordinate system axis.

This class provides the specific interface for packed numeric coordinate values, both integer and floating-point.

Super classes

`zarr::zarr_object` -> `Coordinates` -> `CoordinatesPacked`

Active bindings

`values` (read-only) Retrieve the unpacked coordinate values.

`range` (read-only) Retrieve the extreme values of the coordinates.

Methods

Public methods:

- `CoordinatesPacked$new()`
- `CoordinatesPacked$copy()`
- `CoordinatesPacked$subset()`

`CoordinatesPacked$new()`: Create a new coordinates instance for use with a coordinate system axis for regularly spaced coordinate values.

Usage:

```
CoordinatesPacked$new(
  name,
  direction,
  unit,
  values,
  length,
  bounds = NULL,
  attributes = list()
)
```

Arguments:

`name` Character string. Name of the coordinate set.

`direction` Character string. Direction of the coordinates. Must be one from a set of values.

`unit` Character string. Unit of measure of the coordinates.

`values` A vector of values. There must be two values: the initial coordinate value and the increment.

`length` Integer value giving the number of coordinates that this packed data is for.

`bounds` Optional. If the boundaries are regularly spaced, a vector with the offset to the boundary lower and higher than the coordinate value, respectively. Otherwise, a matrix with two rows and as many columns as there are elements, with the offset to the boundary below the coordinate value in row 1 and the offset to the boundary above the coordinate value in row 2. The boundaries represent the finite extent around the boundary value that this value is representative for. If this argument is not provided, the coordinate values are assumed to represent a point value.

`attributes` Optional. A `list` of attributes of the coordinates.

Returns: An instance of this class or an error.

`CoordinatesPacked$copy()`**:** Return an exact copy of these coordinates.

Usage:

```
CoordinatesPacked$copy()
```

Returns: A new instance of `CoordinatesPacked`.

`CoordinatesPacked$subset()`**:** Return coordinates spanning a smaller coordinate range.

Usage:

```
CoordinatesPacked$subset(rng)
```

Arguments:

`rng` The range of indices to include in the returned coordinates.

Returns: A new `CoordinatesPacked` instance covering the indicated range of indices, including boundary values, present.

`CoordinatesString` *String-type coordinates*

Description

This class implements the string-type coordinates class. The coordinates are always associated with a coordinate system axis.

This class provides the specific interface for string-type coordinate values.

Super classes

`zarr::zarr_object` -> `Coordinates` -> `CoordinatesString`

Methods

Public methods:

- `CoordinatesString$new()`
- `CoordinatesString$copy()`
- `CoordinatesString$subset()`

`CoordinatesString$new()`: Create a new coordinates instance for use with a coordinate system axis for string-type coordinate values.

Usage:

```
CoordinatesString$new(name, direction, unit, values, attributes = list())
```

Arguments:

`name` Character string. Name of the coordinate set.

`direction` Character string. Direction of the coordinates. Must be one from a set of values.

`unit` Character string. Unit of measure of the coordinates.

`values` A vector of values. There must be two values: the initial coordinate value and the increment.

`attributes` Optional. A `list` of attributes of the coordinates.

Returns: An instance of this class or an error.

`CoordinatesString$copy()`: Return an exact copy of these coordinates.

Usage:

```
CoordinatesString$copy()
```

Returns: A new instance of `CoordinatesString`.

`CoordinatesString$subset()`: Return coordinates spanning a smaller coordinate range.

Usage:

```
CoordinatesString$subset(rng)
```

Arguments:

`rng` The range of indices to include in the returned coordinates.

Returns: A new `CoordinatesString` instance covering the indicated range of indices.

`CoordinatesTime` *Time coordinates*

Description

This class implements the coordinates class for time. Time coordinates can use any of the calendars defined by the CF Metadata Conventions.

This class will store the explicit values of the time coordinate, and optionally its boundaries, just like other instances of the base class Coordinates. Additionally, it stores an instance of `CFTime`, which converts the raw values to intelligible coordinates, such as "2026-06-01T12:04:15".

Super classes

zarr::zarr_object -> Coordinates -> CoordinatesTime

Active bindings

values (read-only) Retrieve the coordinate values as timestamps.

bounds Set or retrieve the boundary values around time coordinates. When setting, assign a matrix with 2 rows and as many columns as there are coordinates values. When retrieving, a matrix as just described or NULL if boundary values have not been set.

bounds_raw (read-only) Retrieve the boundary values around time coordinates, packed if possible. If the boundary values are packed, a vector of length 2 is returned, otherwise a matrix with 2 rows and as many columns as there are coordinates values or NULL if boundary values have not been set.

time (read-only) The CFTime instance managing the coordinates.

offsets (read-only) Retrieve the numeric offsets of the time coordinate values.

Methods

Public methods:

- CoordinatesTime$new()
- CoordinatesTime$copy()
- CoordinatesTime$slice()

CoordinatesTime$new(): Create a new coordinates instance for use with a coordinate system axis for time.

Usage:

```
CoordinatesTime$new(
  name,
  direction,
  unit,
  epoch,
  calendar = "standard",
  values,
  bounds = NULL,
  attributes = list()
)
```

Arguments:

name Character string. Name of the coordinate set.

direction Character string. Direction of increasing coordinate values. Must be either "FUTURE" or "PAST".

unit Character string. Unit of measure of the time coordinates. Must be "second", "minute", "hour", "day" or "year", or the abbreviation or plural thereof.

epoch Character string. An ISO 8601 time stamp providing the reference point for time coordinate calculations.

`calendar` Character string, optional. The calendar to use for the time coordinates. Must be one of the calendars supported by the CF Metadata Conventions. If not given, defaults to "standard".

`values` A vector of values. They may be numeric, in which case they are taken to be offsets from the epoch in the given unit and the values may not be packed, or character, in which case they must be ISO 8601 time stamps.

`bounds` Optional. If the boundaries are regularly spaced, a vector with the offset to the boundary lower and higher than the coordinate value, respectively. If the boundaries are irregularly spaced, a matrix with two rows and as many columns as there are elements, with the offset to the boundary below the coordinate value in row 1 and the offset to the boundary above the coordinate value in row 2. The boundaries represent the finite extent around the boundary value that this value is representative for. If this argument is not provided, the coordinate values are assumed to represent a point value.

`attributes` Optional. A `list` of attributes of the coordinates.

Returns: An instance of this class or an error.

`CoordinatesTime$copy()`**:** Return an exact copy of these coordinates.

Usage:

```
CoordinatesTime$copy()
```

Returns: A new instance of `CoordinatesTime`.

`CoordinatesTime$slice()`**:** Retrieve the indices of the time axis falling between two extreme values.

Usage:

```
CoordinatesTime$slice(x, rightmost.closed = FALSE)
```

Arguments:

`x` A vector of two timestamps in between of which all indices into the time axis to extract.

`rightmost.closed` Whether or not to include the upper limit. Default is `FALSE`.

Returns: An integer vector giving the indices in the time axis between values in `x`, or `NULL` if none of the values are valid.

`CoordinateSystem` *Coordinate system*

Description

This class implements the coordinate system class. This class definition implements the same class from the OGC geoAPI Java package, extended with specific code for this package.

Super classes

`zarr::zarr_object` -> `IdentifiedObject` -> `CoordinateSystem`

Active bindings

`axes` (read-only) Retrieve the list of axes that make up this coordinate system.

`crs` Set or retreive the CRS identifier of the coordinate system.

Methods

Public methods:

- `CoordinateSystem$new()`
- `CoordinateSystem$print()`
- `CoordinateSystem$print_axes()`
- `CoordinateSystem$getDimension()`
- `CoordinateSystem$getAxis()`
- `CoordinateSystem$clone()`

`CoordinateSystem$new()`: Create a new coordinate system. The CS must have at least one axis.

Usage:

```
CoordinateSystem$new(name, axes, crs = "", attributes = list())
```

Arguments:

`name` Character string. Name of the coordinate system.

`axes` A `list` of instances of CoordinateSystemAxis. There must be at least one axis in the list.

`crs` Optional, character string with the description of the CRS of this coordinate system, always assumed to be compound.

`attributes` Optional. A `list` of attributes of the coordinate system.

Returns: An instance of `CoordinateSystem` or an error.

`CoordinateSystem$print()`: Print a summary of the coordinate system to the console.

Usage:

```
CoordinateSystem$print(...)
```

Arguments:

`...` Ignored.

Returns: Self, invisibly.

`CoordinateSystem$print_axes()`: Prints details of the axes of the coordinate system to the console. Used internally.

Usage:

```
CoordinateSystem$print_axes(...)
```

Arguments:

`...` Arguments passed on to `data.frame` printing.

`CoordinateSystem$getDimension()`: Retrieve the dimension of the coordinate system, the number of axes that make up the coordinate system.

Usage:

```
CoordinateSystem$getDimension()
```

Returns: Integer value with the dimension of the coordinate system.

`CoordinateSystem$getAxis()`: Retrieve an axis of the coordinate system by its ordinal number.

Usage:

```
CoordinateSystem$getAxis(dimension)
```

Arguments:

`dimension` The dimension whose axis to retrieve. Note that this is 1-based.

Returns: The CoordinateSystemAxis that forms the `dimension` of the coordinate system, or an error if the `dimension` argument in deficient.

`CoordinateSystem$clone()`: The objects of this class are cloneable with this method.

Usage:

```
CoordinateSystem$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CoordinateSystemAxis` *Coordinate system axis*

Description

This class implements the coordinate system axis class. The axis is always associated with a coordinate system.

In an extension over the OGC standard, an axis can have multiple sets of coordinates. The unit-of-measure is associated with the coordinates and thus not with the axis.

Super classes

`zarr::zarr_object` -> `IdentifiedObject` -> `CoordinateSystemAxis`

Active bindings

`abbreviation` Set or retrieve the abbreviation of the axis.

`coordinates_list` (read-only) Retrieve all coordinates in a `list`.

`coordinates` (read-only) Retrieve the currently active coordinates.

`length` (read-only) Retrieve the length of the axis.

Methods

Public methods:

`CoordinateSystemAxis$new()`: Create a new axis instance for use in a coordinate system.

Usage:

```
CoordinateSystemAxis$new(name, abbreviation, coordinates, attributes = list())
```

Arguments:

`name` Character string. Name of the axis.

`abbreviation` Character string. Abbreviation of the axis.

`coordinates` A list with one or more named instances of the Coordinates class, or any of its descendants. The first element in the list will be made active.

`attributes` Optional. A `list` of attributes of the axis.

Returns: An instance of this class or an error.

`CoordinateSystemAxis$print()`: Print a summary of the coordinate system axis to the console.

Usage:

```
CoordinateSystemAxis$print(...)
```

Arguments:

`...` Ignored.

Returns: Self, invisible.

`CoordinateSystemAxis$brief()`: Some details of the axis.

Usage:

```
CoordinateSystemAxis$brief()
```

Returns: A 1-row `data.frame` with some details of the axis.

`CoordinateSystemAxis$copy()`: Return an exact copy of this axis.

Usage:

```
CoordinateSystemAxis$copy()
```

Returns: A new instance of `CoordinateSystemAxis`.

`CoordinateSystemAxis$subset()`: Return an axis spanning a smaller coordinate range. This method returns an axis which spans the range of indices given by the `rng` argument.

Usage:

```
CoordinateSystemAxis$subset(name = "", rng = NULL)
```

Arguments:

`name` The name for the new axis. If an empty string is passed (default), will use the name of this axis.

`rng` The range of indices whose values from this axis to include in the returned axis. If the value of the argument is `NULL`, return a copy of the axis.

Returns: A new `CoordinateSystemAxis` instance covering the indicated range of indices. If the value of the argument `rng` is `NULL`, return a copy of this axis as the new axis.

`CoordinateSystemAxis$slice()`: Given a range of domain coordinate values, returns the indices into this axis that fall within the supplied range. If the axis has bounds, any coordinate whose boundary values fall entirely or partially within the supplied range will be included in the result.

Usage:

```
CoordinateSystemAxis$slice(rng)
```

Arguments:

`rng` A numeric vector whose extreme values indicate the indices of coordinates to return.

Returns: An integer vector of length 2 with the lower and higher indices into the axis that fall within the range of coordinates in argument `rng`. Returns `NULL` if no (boundary) values of the axis fall within the range of coordinates.

`CoordinateSystemAxis$getAbbreviation()`: Retrieve the abbreviation of the axis. This method is mandatory in the OGC standard.

Usage:

```
CoordinateSystemAxis$getAbbreviation()
```

Returns: Character string with the abbreviation of the axis.

`CoordinateSystemAxis$getDirection()`: Retrieve the direction of the axis. This method is mandatory in the OGC standard.

Usage:

```
CoordinateSystemAxis$getDirection()
```

Returns: Character string with the direction of the axis.

`CoordinateSystemAxis$getUnit()`: Retrieve the unit of measure of the axis. This method is mandatory in the OGC standard.

Usage:

```
CoordinateSystemAxis$getUnit()
```

Returns: Character string with the unit of measure of the axis.

`CoordinateSystemAxis$getMinimumValue()`: Retrieve the minimum coordinate value of this axis, in units of the unit of measure of the axis.

Usage:

```
CoordinateSystemAxis$getMinimumValue()
```

Returns: Negative infinity.

`CoordinateSystemAxis$getMaximumValue()`: Retrieve the maximum coordinate value of this axis, in units of the unit of measure of the axis.

Usage:

```
CoordinateSystemAxis$getMaximumValue()
```

Returns: Positive infinity.

`CoordinateSystemAxis$getRangeMeaning()`: Retrieve the minimum coordinate value of this axis, in units of the unit of measure of the axis.

Usage:

```
CoordinateSystemAxis$getRangeMeaning()
```

Returns: The character string 'EXACT'.

`CoordinateSystemAxis$clone()`: The objects of this class are cloneable with this method.

Usage:

```
CoordinateSystemAxis$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`create_geozarr_array` *Create a new GeoZarr array*

Description

This function creates a new GeoZarr array in an existing Zarr store. Only the structure of the array will be created; data and other properties like attributes have to be added through additional method calls. This function is particularly well suited to creating an array that mimics the structure of a netCDF data variable.

Usage

```
create_geozarr_array(name, location, axes, data_type, fill_value)
```

Arguments

name	The name of the GeoZarr array to be created.
location	A zarr_group instance. The new array will be created in this group, with the indicated name.
axes	A named list with the axis information for the array. For every element in the list an axis will be created in the array with a length that equals the number of coordinates contained in the list element. See the Details section and the example for the formatting details of the list.
data_type	Character. The Zarr data type for the array.
fill_value	Optional. The sentinel value used to indicate parts of the array that do not have data assigned. The value of this argument must agree with the data_type. If not provided a default value will be used that is specific to the data_type.

Details

The axes argument provides all the details of the coordinate system of the new array in a compound list. The elements in the list are each a list, named after the names of the axes. The number of axes listed in the top-level list determines the rank of the new array: its number of dimensions. The order of the dimensions in the array that will be created is the same as the order of the axes in this argument. The sub-lists are structured as follows:

- coordinates: A vector of coordinate values. The length of the vector becomes the length of the corresponding dimension in the array.
- boundaries: Optional. For boundaries that are regular, meaning that the lower offset (-\eqn{\infty}, 0) for each coordinate is constant and the same for the higher offset (0, \eqn{\infty}), then a vector of the lower and higher offset, in that order, may be supplied. Otherwise a matrix has to be supplied with as many columns as there are coordinates and the lower offsets in row 1 and the higher offsets in row 2.
- abbreviation: A character from the set (X, Y, Z, T). These values indicate the spatio-temporal dimensions of the array. If omitted, an empty string or a space character the axis will have no spatio-temporal meaning, such as for a categorical axis. Note that both "X" and "Y" must be present in the set of axes. Abbreviations may not be duplicated across axes.
- direction: The identifier of the direction of increasing coordinate values of the axis. Its value must be present in the AxisDirection object in this package. If omitted, the direction will be inferred from the abbreviation and coordinates fields of the axis, which is usually correct.
- unit: The unit-of-measure of the coordinate values. While not mandatory, it is highly recommended to include this information. For a temporal axis the acceptable values are "second", "minute", "hour", "day" and "year", as well as the abbreviation or plural version thereof; if omitted, "days" is used. When temporal coordinates are specific as numeric offsets, this field is required.
- calendar: Temporal axis only. The name of the calendar to use. This package supports all calendars of the CF Metadata Conventions, including the "model" calendars and leap seconds in the "utc" calendar. If omitted, "proleptic_gregorian" is used, which is almost identical to the common calendar used by regular R date-time functions. When temporal coordinates are specific as numeric offsets, this field is required.

- epoch: Temporal axis only. The starting point from which time coordinates are calculated. Must be specified in ISO 8601 format, but with support for model calendars. If omitted, "1970-01-01" is used, as in regular R date-time functions. When temporal coordinates are specific as numeric offsets, this field is required.
- chunk_weight: Optional chunking weight of this axis. Higher values lead to less fragmentation of the axis into separate chunks compared to lower values. If omitted, it is set to 1 for this axis. Typical values are in the range of 1 - 3.

The top-level list may have an additional element called .convention that can hold additional information specific to the convention used for encoding of the array metadata; see below for details.

Depending on the details of the coordinates argument, the GeoZarr object may use the "spatial" or "cs" convention for encoding.

"spatial" convention: The "spatial" convention encoding is the most compact and it will be used for arrays that have X and Y axes and an optional third axis which is typically an image band or a discrete (class) axis – the third axis may not represent height/depth (Z) or time (T). While the "spatial" convention does work on Zarr arrays with more dimensions, it has no mechanism to attach coordinates to any axes other than X and Y. The X and Y coordinates must be numeric and regularly spaced and the Y coordinate values must be decreasing. In other words, the "spatial" convention will be used for imagery style, north-up arrays with a coordinate system tied to the top-left corner of the array space. The axis sub-list only requires the coordinates and abbreviation fields for each of the axes; chunk_weight will be used when specified, all other fields are ignored.

If a .convention element is present in the top-level axes list then it is searched for a registration member. Its value is a character string with a value of "node" or "pixel". If omitted, a value of "pixel" is used.

"cs" convention: The "cs" convention is used for all cases where the "spatial" convention does not apply. All fields for the axis sub-lists in the axes argument are interpreted.

If the coordinates along an axis are not regularly spaced, a secondary Zarr array will be created to hold the coordinate values. By default the secondary array will be stored in the same group as the main array or another existing group may be indicated. If the length of the axis is no more than the option GeoZarr.options$max_explicit its coordinates are stored in the array cs attributes.

Any time coordinates will be converted to a CFtime format with a user-defined epoch and calendar, or default values can be used to align with the common calendar. Time coordinates may be specified as a character vector with coordinates in ISO 8601 format, or as a numeric vector of offsets from an epoch. In this latter case, the vector may not be packed and the fields "unit", "calendar" and "epoch" must be given for the temporal axis. Boundary values may be set around numerical coordinates (both spatial and temporal).

If a .convention element is present in the top-level axes list then it is searched for a coordinate_group member. Its value is a character string giving the relative path from the newly created array to the group where external coordinate arrays are to be stored. That group must already exist. If omitted, a value of ".." is used, meaning that any external coordinate arrays are placed in the same group as the new array.

Value

The new geozarr_array instance.

Examples

```
z <- create_zarr()
grp <- z$add_group("/", "my_data_group")
ext <- z$add_group("/", "coords")
gza <- create_geozarr_array(name = "my_data_array", location = grp,
                            data_type = "float32", fill_value = -9e12,
                            axes = list(
         longitude   = list(coordinates = seq(from = -175, to = 175, by = 10),
                            boundaries = rbind(seq(from = -180, to = 170, by = 10),
                                               seq(from = -170, to = 180, by = 10)),
                            abbreviation = "X",
                            unit = "degrees"),
         latitude    = list(coordinates = c(-90, -75, -60, -50, -40, -30, -20, -10,
                                            0, 10, 20, 30, 40, 50, 60, 75, 90),
                            abbreviation = "Y",
                            unit = "degrees"),
         time        = list(coordinates = sprintf("2026-%02d-%02d",
                                                  rep(1:12, each = 30),
                                                  rep(1:30, times = 12)),
                            abbreviation = "T",
                            unit = "days",
                            calendar = "360_day",
                            epoch = "1850-01-01",
                            chunk_weight = 1.5),
         .convention = list(coordinate_group = "../../coords")
         )
       )
z$hierarchy()
gza
```

---

geozarr_array *GeoZarr Array*

Description

This class implements a GeoZarr array. A GeoZarr array is like a regular Zarr array but it has attributes and/or associated Zarr arrays that provide a coordinate system for the array.

Super classes

zarr::zarr_object -> zarr::zarr_node -> zarr::zarr_array -> geozarr_array

Active bindings

`coordinate_system` (read-only) Retrieve the coordinate system of this array.

Methods

Public methods:

- geozarr_array$new()
- geozarr_array$post_open()
- geozarr_array$build_coordsys()
- geozarr_array$subset()
- geozarr_array$write_external_coordinates()

`geozarr_array$new()`: Initialize a new GeoZarr array.

Usage:

```
geozarr_array$new(name, metadata, parent, store, coord_sys)
```

Arguments:

`name` The name of the GeoZarr array.

`metadata` List with the metadata of the array.

`parent` The parent `zarr_group` instance of this new array, can be missing or `NULL` if the Zarr object should have just this array.

`store` The zarr_store instance to persist data in.

`coord_sys` Optional, an instance of CoordinateSystem providing the coordinate system of the array. If not provided, the coordinate system is constructed from the metadata of the array persisted in the store.

Returns: An instance of `geozarr_array`.

`geozarr_array$post_open()`: Perform any processing after the Zarr hierarchy is in place and out-of-group references can be resolved.

Usage:

```
geozarr_array$post_open()
```

Returns: Self, invisibly.

`geozarr_array$build_coordsys()`: Build the coordinate system of the GeoZarr array if it has not been set yet. This should only be called when the Zarr hierarchy is in place and out-of-group references can be resolved, particularly for the `cs` convention.

Usage:

```
geozarr_array$build_coordsys()
```

Returns: Self, invisibly.

`geozarr_array$subset()`: This method extracts a subset of values from the GeoZarr array, with the range along each axis to extract expressed in coordinate values of the domain of each axis.

Usage:

```
geozarr_array$subset(
  ...,
  .name = NULL,
  .location = NULL,
  .rightmost.closed = FALSE
)
```

Arguments:

`...` One or more arguments of the form `axis = range`. The "axis" part should be the name of an axis or its abbreviation `X`, `Y`, `Z` or `T`. The "range" part is a vector of values representing coordinates along the axis where to extract data. Axis abbreviations and names are case-sensitive and can be specified in any order. If values for the range per axis fall outside of the extent of the axis, the range is clipped to the extent of the axis.

`.name` The name of the GeoZarr array to be created. If omitted, an array will be created at the root of a new in-memory Zarr store.

`.location` Optional. If supplied, either an existing zarr_group in a zarr object, or a character string giving the location on a local file system where to persist the data. If the argument is a `zarr_group`, argument `.name` must be provided. If the argument gives the location for a new Zarr store then the location must be writable by the calling code. As per the Zarr specification, it is recommended to use a location that ends in ".zarr" when providing a location for a new store. If argument `.name` is given then the `geozarr_array` will be created in the root of the `zarr` store with that name. If the `.name` argument is not given, a single-array Zarr store will be created. If the `location` argument is not given, a `zarr` object is created in memory.

`.rightmost.closed` Optional. Single logical value to indicate if the upper boundary of range in each axis should be included.

Details: The range of values along each axis to be subset is expressed in coordinates of the domain of the axis. Any axes for which no selection is made in the `...` argument are extracted in whole. Coordinates can be specified in a variety of ways that are specific to the nature of the axis. For numeric axes it should (resolve to) be a vector of real values from which the range is computed. For time axes a vector of character timestamps, `POSIXct` or `Date` values must be specified. As with numeric values, only the two extreme values in the vector will be used. For character axes the order in the axis will be used, with the first and last value in the supplied range.

If the range of coordinate values for an axis in argument `...` extends the valid range of the axis, the extracted data will start at the beginning for smaller values and extend to the end for larger values. If the values envelope the valid range the entire axis will be extracted in the result. If the range of coordinate values for any axis are all either smaller or larger than the valid range of the axis then nothing is extracted and `NULL` is returned.

The extracted data has the same dimensional structure as the data in the array, with degenerate dimensions preserved. The order of the axes in argument `...` does not reorder the axes in the result.

Arguments following `...` must be explicitly named, like `.rightmost.closed = TRUE`, to avoid the argument being treated as an axis name.

As an example, to extract values of a variable for Australia for the year 2020, where the first axis in GeoZarr array `x` is the longitude, the second axis is the latitude, both in degrees, and the third (and final) axis is time, the values are extracted by `x$subset(X = c(112, 154), Y = c(-9, -44), T = c("2020-01-01", "2021-01-01"))`. Note that this works equally well for projected coordinate reference systems - the key is that the specification in argument `...` uses the same domain of values as the respective axes in `x` use.

Returns: If the `.location` argument is a `zarr_group`, the new Zarr `geozarr_array` is returned, with a subset of data from this GeoZarr array, having the axes and attributes of this GeoZarr array. Otherwise, the `zarr` object that is newly created and which contains the `geozarr_array` instance, or an error if the `zarr` object could not be created. If one or more of the selectors in the `...` argument fall entirely outside of the range of the axis `NULL` is returned.

`geozarr_array$write_external_coordinates()`: Write any external coordinates of this array to the underlying Zarr store. The metadata of this array is scanned for `ref` objects in the `cs` convention attributes. The values of the external coordinates are taken from the CoordinateSystem instance of this array.

Usage:

```
geozarr_array$write_external_coordinates()
```

Returns: Self, invisibly.

`geozarr_group` *GeoZarr Group*

Description

This class implements a GeoZarr group. A GeoZarr group is a group in the hierarchy of a Zarr object having GeoZarr attributes. A GeoZarr group is a container for other groups and arrays.

Super classes

`zarr::zarr_object` -> `zarr::zarr_node` -> `zarr::zarr_group` -> `geozarr_group`

Methods

Public methods:

- `geozarr_group$new()`

`geozarr_group$new()`: Open a GeoZarr group in a Zarr hierarchy.

Usage:

```
geozarr_group$new(name, metadata, parent, store)
```

Arguments:

`name` The name of the group. For a root group, this is the empty string `""`.

`metadata` List with the metadata of the group.

`parent` The parent Zarr group instance of this new group, can be missing or `NULL` for the root group.

`store` The zarr_store instance to persist data in.

Returns: An instance of `geozarr_group`.

geozarr_options *GeoZarr package options*

Description

Use this function to read or modify package options.

Usage

```
geozarr_options(key, value)
```

Arguments

key	Character. A key whose value to modify. If missing, all options are returned.
value	The new value for the option.

Value

A list with all options if argument key is not provided, nothing otherwise.

Examples

```
geozarr_options()
```

IdentifiedObject *Identified object*

Description

This class definition implements the same class from the OGC geoAPI Java package, extended with specific code for this package. This class is ancestor to many other, more useful classes. This class just provides basic identification properties. It is not very useful to instantiate this class directly; instead, use descendant classes.

Super class

zarr::zarr_object -> IdentifiedObject

Active bindings

name (read-only) The name of the object.

attributes (read-only) A named list with the attributes of this object.

Methods

Public methods:

`IdentifiedObject$new()`: Create a new object.

Usage:

```
IdentifiedObject$new(name, attributes = list())
```

Arguments:

`name` Character string. Name of the object.

`attributes` Optional. A `list` of attributes of the object.

Returns: A new instance of the object, or an error if the object could not be created.

`IdentifiedObject$getName()`: Retrieve the name of the object. This method is mandatory in the OGC standard.

Usage:

```
IdentifiedObject$getName()
```

Returns: Character string with the name of the object.

`IdentifiedObject$setAlias()`: Set aliases for the object. The aliases are added at the end of the defined aliases.

Usage:

```
IdentifiedObject$setAlias(alias)
```

Arguments:

`alias` Character vector with aliases for the object.

Returns: Self, invisibly.

`IdentifiedObject$getAlias()`: Retrieve the list of aliases for the name of the object. This method is optional in the OGC standard.

Usage:

```
IdentifiedObject$getAlias()
```

Returns: List of aliases for the name of the object.

`IdentifiedObject$setIdentifiers()`: Set identifiers for the object. The identifiers are added at the end of the defined identifiers.

Usage:

```
IdentifiedObject$setIdentifiers(id)
```

Arguments:

`id` Character vector with identifiers for the object.

Returns: Self, invisibly.

`IdentifiedObject$getIdentifiers()`: Retrieve the list of identifiers of the object. This method is optional in the OGC standard.

Usage:

```
IdentifiedObject$getIdentifiers()
```

Returns: List of identifiers for the object.

`IdentifiedObject$setRemarks()`: Set remarks for the object.

Usage:

```
IdentifiedObject$setRemarks(remarks)
```

Arguments:

`remarks` Character string with remarks for the object.

Returns: Self, invisibly.

`IdentifiedObject$getRemarks()`: Retrieve the remarks of the object. This method is optional in the OGC standard.

Usage:

```
IdentifiedObject$getRemarks()
```

Returns: Character string with remarks or `NA` if no remarks have been set.

`IdentifiedObject$clone()`: The objects of this class are cloneable with this method.

Usage:

```
IdentifiedObject$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`set_convention` *Set the GeoZarr convention with array details in the metadata of the array*

Description

This function will write the GeoZarr convention with all details into the metadata of a Zarr array.

Usage

```
set_convention(
  metadata,
  coord_sys,
  crs = NULL,
  external_group,
  registration = "pixel"
)
```

Arguments

metadata	A list with the basic metadata of the array.
coord_sys	The CoordinateSystem instance of the GeoZarr array.
crs	Optional, a list with 1 or more CRS definitions. Named list elements can be "compound", "spatial" (X-Y), "vertical" or "temporal". Those elements are a list themselves with one or more of the named elements "code", "wkt2" or "projjson" (for the attributes in the proj convention) with their value being the attribute to register. In the cs convention, a "compound" CRS will be associated with the coordinate system of the GeoZarr array, the other options will be associated with the specific CRSs of the coordinate system. In the spatial convention there can only be 1 CRS which must be of type "compound" or "spatial" and which is registered at the root of the "attributes" of the GeoZarr array.
external_group	Optional, the path to the group, relative to the location of the array, that stores any external arrays with coordinate values.
registration	Optional, the registration point of the array for use with the "spatial" convention. Defaults to "pixel".

Value

A list with the metadata updated with convention attributes.

Examples

```
ab <- zarr::array_builder$new()
ab$data_type <- "int32"
ab$shape <- c(1000L, 20L)

crd1 <- CoordinatesPacked$new("X_coordinates", "EAST", "m", c(0, 1000), 1000L)
ax1 <- CoordinateSystemAxis$new("Axis1", "X", crd1)
crd2 <- CoordinatesPacked$new("Y_coordinates", "NORTH", "m", c(0, 1000), 20L)
ax2 <- CoordinateSystemAxis$new("Axis2", "Y", crd2)

cs <- CoordinateSystem$new("CS", list(Axis1 = ax1, Axis2 = ax2))

crs <- list(spatial = list(code = "EPSG:4326"))

set_convention(ab$metadata(), cs, crs)
```

zarr_convention_cs *GeoZarr "cs" convention*

Description

This class implements the GeoZarr "cs" (coordinate set) convention. The convention attaches a full coordinate system to a Zarr array by recording, for each dimension, the axis abbreviation and direction, one or more sets of coordinate values and their optional cell-boundary values, and, where applicable, the parametric formula and its terms needed to derive physical coordinates from stored index coordinates.

The convention is registered via the following CMO:

```
{
 "schema_url": "https://raw.githubusercontent.com/R-CF/zarr_convention_cs/main/schema.json",
 "spec_url":  "https://raw.githubusercontent.com/R-CF/zarr_convention_cs/main/README.md",
  "uuid":       "e4dbf0b7-7a00-4ce6-b23e-484292014ab4",
  "name":       "cs",
  "description": "Coordinate set convention for Zarr arrays"
}
```

The `cs` attribute written to the array metadata has the following structure (simplified):

```
{
  "cs": {
    "crs": [
      {
        "type": "compound" | "planar" | "vertical" | "temporal" | "undefined",
        "description: "optional text",
        "axes": {
          "<dimension_name>": {
            "abbreviation": "X",
            "direction":    "EAST",
            "coordinates":  [ { "unit": "degrees",
                                "values": { "regular": [0.0, 0.5] } } ]
          }
        }
      }
    ],
    "id": "Identifier of a compound CRS",
    "attributes": { <optional attributes as a JSON sub-schema> }
  }
}
```

Build a convention instance, add one or more CRS objects (each covering one or more dimension axes), then call `as_list()` to retrieve everything for inclusion as the "cs" attribute in the Zarr node metadata.

Super class

zarr::zarr_convention -> zarr_convention_cs

Active bindings

name Optional descriptive name for the coordinate set.

cs_crs Optional top-level CRS identifier, an instance of zarr_conv_proj.

crs (read-only) The list of CRS objects accumulated so far.

Methods

Public methods:

- zarr_convention_cs$new()
- zarr_convention_cs$add_crs()
- zarr_convention_cs$as_list()
- zarr_convention_cs$axis()
- zarr_convention_cs$coordinates()
- zarr_convention_cs$values_regular()
- zarr_convention_cs$values_explicit()
- zarr_convention_cs$values_external()
- zarr_convention_cs$boundaries_regular()
- zarr_convention_cs$time()
- zarr_convention_cs$parametric()

zarr_convention_cs$new(): Create a new instance of a "cs" convention agent.

Usage:

```
zarr_convention_cs$new()
```

Returns: A new instance of a "cs" convention agent.

zarr_convention_cs$add_crs(): Add a CRS object to this coordinate set. Each CRS covers one or more axes (dimensions). Multiple calls add further CRS objects to the array, which is necessary when the array spans domains described by separate OGC coordinate reference systems (e.g. a horizontal CRS plus a vertical CRS plus a temporal CRS).

Usage:

```
zarr_convention_cs$add_crs(
  axes,
  type = "unknown",
  id = NULL,
  geolocation = NULL
)
```

Arguments:

axes A named list of axis definitions, keyed by the dimension name that appears in the Zarr array's dimension_names metadata. If the argument is NULL or an empty list, the call is a no-op.

`type` Optional, character string. Type of CRS, one of "compound", "planar", "vertical", "temporal", "unknown" (default).

`id` Optional. An instance of zarr_conv_proj providing the formal OGC description of the CRS.

`geolocation` Optional list. Geolocation grid definition for curvilinear grids. This should only be included for CRS's that represent a planar coordinate system.

Returns: Self, invisibly.

`zarr_convention_cs$as_list()`: Retrieve the `cs` attributes as a list.

Usage:

```
zarr_convention_cs$as_list()
```

Returns: A `list` with the updated attributes for this convention.

`zarr_convention_cs$axis()`: Build an axis definition.

Usage:

```
zarr_convention_cs$axis(coordinates, abbreviation = NULL, direction = NULL)
```

Arguments:

`coordinates` A list of coordinate-set definitions, each produced by `coordinates()`. Must have at least one element.

`abbreviation` Optional character string. Axis abbreviation, e.g. `"X"`, `"Y"`, `"Z"`, or `"T"`.

`direction` Optional character string. Direction of increasing coordinate values taken from Table 48 of the OGC standard "Referencing by Coordinates" (e.g. `"EAST"`, `"NORTH"`, `"UP"`, `"FUTURE"`).

Returns: A named list representing one axis entry.

`zarr_convention_cs$coordinates()`: Build a coordinate-set definition for use in `axis()`.

Usage:

```
zarr_convention_cs$coordinates(
  values,
  name = NULL,
  unit = NULL,
  boundaries = NULL,
  parametric = NULL,
  time = NULL,
  attributes = NULL
)
```

Arguments:

`values` A values definition produced by one of `values_regular()`, `values_explicit()`, or `values_external()`.

`name` Optional, character string. Descriptive name for this set of coordinates.

`unit` Optional, character string. Unit of measure (e.g. `"degrees_east"`, `"m"`, `"1"`).

`boundaries` Optional boundaries definition produced by `boundaries_regular()` or `values_external()`.

`parametric` Optional parametric definition produced by `parametric()`.

`time` Optional time definition produced by `time()`.

`attributes` Optional list with attributes for the coordinates.

Returns: A named list representing one coordinates entry.

`zarr_convention_cs$values_regular()`: Regularly-spaced coordinate values.

Usage:

```
zarr_convention_cs$values_regular(start, increment)
```

Arguments:

`start` Numeric. The coordinate value at shape index 0.

`increment` Numeric. The constant spacing between successive values. May be negative for decreasing coordinates (e.g. north-to-south latitudes); it cannot be 0.

Returns: A `values` list with a `regular` element.

`zarr_convention_cs$values_explicit()`: Explicitly listed coordinate values.

Usage:

```
zarr_convention_cs$values_explicit(values)
```

Arguments:

`values` A vector of coordinate values. May be numeric, integer, or character.

Returns: A `values` list with an `explicit` element.

`zarr_convention_cs$values_external()`: Coordinate values stored in an external array

Usage:

```
zarr_convention_cs$values_external(node, uri)
```

Arguments:

`node` Character string. Path to the 1-dimensional Zarr array containing the coordinate values, relative to the referring node.

`uri` Optional character string. URI of an external store. Omit for arrays in the same local store.

Returns: A `values` list with an `external` element.

`zarr_convention_cs$boundaries_regular()`: Regularly-spaced cell-boundary values.

The two values give the offset *below* and *above* the coordinate value that define the extent of each cell, in the same unit as the coordinates. Both offsets are expressed relative to the coordinate value so `below` must be non-positive and `above` must be non-negative.

Usage:

```
zarr_convention_cs$boundaries_regular(below, above)
```

Arguments:

`below, above` Numeric. Offset from the coordinate value to the lower and upper boundary, respectively.

Returns: A `boundaries` list with a `regular` element.

`zarr_convention_cs$time()`: Provides the time reference information needed to interpret numeric coordinate values on a temporal axis.

Usage:

```
zarr_convention_cs$time(unit, epoch, calendar = NULL)
```

Arguments:

`unit` Character string. The time unit, e.g. `"days"`, `"hours"`, `"seconds"`.

`epoch` Character string. The reference date/time in ISO 8601 format, e.g. `"1970-01-01"`.

`calendar` Optional character string. The CF calendar name, e.g. `"standard"`, `"360_day"`. Defaults to `"proleptic_gregorian"` when omitted.

Returns: A `time` list.

`zarr_convention_cs$parametric()`: Records the CF formula name and the set of formula terms needed to derive physical coordinates from the stored parametric index coordinates. This object sits alongside `values` inside a `coordinates()` call; `values` provides the stored parametric coordinates (e.g. `s_rho`), while `parametric` provides the machinery to recover the physical coordinates (e.g. depth in metres).

Usage:

```
zarr_convention_cs$parametric(formula, terms)
```

Arguments:

`formula` Character string. The CF `standard_name` of the parametric coordinate formula, e.g. `"ocean_s_coordinate_g2"`.

`terms` A named list of formula term values. Each element must be a `values` attribute object. Scalar or short constants should use explicit coordinate values; full-length arrays should use external Zarr arrays.

Returns: A `parametric` list.

`zarr_conv_proj` *GeoZarr "proj" convention*

Description

This class implements the GeoZarr "proj" convention. In particular, the following convention is implemented here:

```
{
  "schema_url": "https://raw.githubusercontent.com/zarr-conventions/geo-proj/refs/tags/v1/schema.jso
   "spec_url": "https://github.com/zarr-conventions/geo-proj/blob/v1/README.md",
   "uuid": "f17cb550-5864-4468-aeb7-f3180cfb622f",
   "name": "proj",
   "description": "Coordinate reference system information for geospatial data"
}
```

Super class

`zarr::zarr_convention` -> `zarr_conv_proj`

Active bindings

`code` The "proj:code" attribute, a character string in "authority:code" format identifying a CRS.

`wkt2` The "proj:wkt2" attribute, a character string giving a CRS in WKT2 format.

`projjson` The "proj:projjson" attribute, a character string giving a CRS in PROJJSON format.

Methods

Public methods:

- `zarr_conv_proj$new()`
- `zarr_conv_proj$write()`

`zarr_conv_proj$new()`: Create a new instance of a "proj" convention agent.

Usage:

```
zarr_conv_proj$new(attributes)
```

Arguments:

`attributes` Optional, a named `list` with one or more of the `proj` attributes. The elements in the list must be one or more of "code", "wkt2" and "projjson".

Returns: A new instance of a "proj" convention agent.

`zarr_conv_proj$write()`: Write the data of this instance in the attributes of a Zarr object.

Usage:

```
zarr_conv_proj$write(attributes)
```

Arguments:

`attributes` A `list` with Zarr attributes for a group or array. The properties will be written at the root level of `attributes`.

Returns: The updated attributes.

`zarr_conv_spatial` *GeoZarr "spatial" convention*

Description

This class implements the GeoZarr "spatial" convention. In particular, the following convention is implemented here:


```
{
  "schema_url": "https://raw.githubusercontent.com/zarr-conventions/spatial/refs/tags/v1/schema.json
   "spec_url": "https://github.com/zarr-conventions/spatial/blob/v1/README.md",
   "uuid": "689b58e2-cf7b-45e0-9fff-9cfc0883d6b4",
   "name": "spatial",
   "description": "Spatial coordinate information"
}
```

Super class

`zarr::zarr_convention` -> `zarr_conv_spatial`

Active bindings

`dimensions` The "spatial:dimensions" attribute, a character vector of dimension names for the Y and X axes, in that order. These names must correspond to the names in the "dimension_names" attribute of the array that this convention relates to.

`bbox` The "spatial:bbox" attribute, a numeric vector of 4 values in order `xmin, ymin, xmax, ymax` giving the boundary coordinates of the data array.

`transform_type` (read-only) The "spatial:transform_type" attribute, a character string giving the type of coordinate transformation. The only valid value (currently) is "affine".

`transform` The "spatial:transform" attribute, a numeric vector of 6 values (1) X resolution; (2) 0; (3) X coordinate of UL corner; (4) 0; (5) Y resolution; (6) Y coordinate of UL corner, giving the transformation coefficients of the coordinates of the data array.

`shape` The "spatial:shape" attribute, an integer vector of length 2 giving the length of the X and Y dimensions.

`registration` The "spatial:registration" attribute, a string with value "node" or "pixel". "pixel" (the default) means that the coordinates of the UL corner of each grid cell are recorded; "node" means that the coordinates of the center of each grid cell are recorded.

Methods

Public methods:

- `zarr_conv_spatial$new()`
- `zarr_conv_spatial$set_coordinates()`
- `zarr_conv_spatial$write()`

`zarr_conv_spatial$new()`**:** Create a new instance of a "spatial" convention agent.

Usage:

```
zarr_conv_spatial$new()
```

Returns: A new instance of a "spatial" convention agent.

`zarr_conv_spatial$set_coordinates()`**:** Set the coordinate system for this instance. This method sets the affine transform coefficients, as well as the grid cell registration and the bounding box.

Usage:

```
zarr_conv_spatial$set_coordinates(x, y, shape, registration = "pixel")
```

Arguments:

`x, y` Coordinates for the `X` and `Y` axes, as a numeric vector of two values: the top-left coordinate of the top-left grid cell and the resolution along the axis, respectively. Note that for `y` the resolution must be negative.

`shape` The length of each axis `x` and `y`.

`registration` Grid cell registration. "pixel" (the default) means that the coordinates in `x` and `y` are interpreted as the UL corner of each grid cell; "node" means that the coordinates are interpreted as the center of each grid cell.

Returns: Self, invisibly.

`zarr_conv_spatial$write()`: Write the data of this instance in the attributes of a Zarr object.

Usage:

```
zarr_conv_spatial$write(attributes)
```

Arguments:

`attributes` A `list` with Zarr attributes for a group or array. The properties will be written at the root level of `attributes`.

Returns: The updated attributes.

`zarr_domain_geozarr` *GeoZarr domain object*

Description

This class implements a GeoZarr domain object. A GeoZarr domain object is a `zarr_domain` descendant object identifying groups and arrays in a `zarr` object that are formatted using GeoZarr conventions.

This domain supports the standard conventions for GeoZarr data sets, specifically the `cs` and `spatial` conventions, as well as two other formats for geospatial Zarr data that predate GeoZarr: the format used by the Python XArray library (Zarr v.2 and v.3) and the NCZarr format (Zarr v.2).

Super class

`zarr::zarr_domain` -> `zarr_domain_geozarr`

Methods

Public methods:

- `zarr_domain_geozarr$new()`
- `zarr_domain_geozarr$build()`

`zarr_domain_geozarr$new()`: Create a new GeoZarr domain instance. The GeoZarr domain instance manages the groups and arrays in the Zarr store that it refers to. This instance provides access to all objects in the Zarr store.

Usage:

```
zarr_domain_geozarr$new()
```

Returns: A `zar_domain_geozarr` object.

`zarr_domain_geozarr$build()`: This method will create a `geozarr_array` for an array node and a `geozarr_group` for a group node with GeoZarr conventions declared in its attributes. Either the "spatial" or "cs" convention has to be declared or the Zarr store has to be formatted using XArray or NCZarr or this domain will decline to manage the node.

Usage:

```
zarr_domain_geozarr$build(name, metadata, parent, store)
```

Arguments:

`name` The name of the node.

`metadata` List with the metadata of the node.

`parent` The parent node of this new node. May be `NULL` for a root node.

`store` The store to persist data in.

Returns: A `geozarr_array` or `geozarr_group` instance if supported, `FALSE` otherwise.

Index