Package: ndr (via r-universe)

November 8, 2026

Title Named-Dimension Arrays with Coordinate-Based Indexing

Version 0.1.0.9013

Description Provides Variable, DataArray, and Dataset classes for N-dimensional arrays with named dimensions and coordinate-based indexing. Arithmetic operations broadcast by dimension name, not position. Designed as a minimal foundation for lazy, multi-file scientific data access - the core data structures without any file backends.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Imports S7 (>= 0.2.0)

Suggests altarr, CFtime, DBI, GDAL7, jsonlite, RSQLite, zaro (>= 0.0.1.9035), testthat (>= 3.0.0)

Config/testthat/edition 3

Collate 'ndr-package.R' 'variable.R' 'index.R' 'coordinate.R' 'affine.R' 'utils.R' 'cf.R' 'broadcast.R' 'dataarray.R' 'dataset.R' 'lazy.R' 'ops.R' 'sel.R' 'plan.R' 'reduce.R' 'print.R' 'coerce.R' 'open.R' 'zzz.R'

URL https://github.com/hypertidy/ndr, https://hypertidy.github.io/ndr/

BugReports https://github.com/hypertidy/ndr/issues

Remotes rgdal-dev/GDAL7, hypertidy/altarr, hypertidy/zaro

Config/roxygen2/version 8.1.1

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

Date/Publication 2026-11-08 14:15:02 UTC

RemoteUrl https://github.com/hypertidy/ndr

RemoteRef HEAD

RemoteSha 90c70bbc6c8cf0d9a2edc320372c7c85d7fd8499

Contents

$.ndr::Dataset . . . 2
AffineIndex . . . 3
as.data.frame.ndr::DataArray . . . 4
as.list.ndr::Dataset . . . 5
as_geotransform . . . 5
as_variable . . . 6
broadcasting . . . 6
cf_decode_time . . . 7
collect . . . 7
coord_dim . . . 8
coord_length . . . 8
coord_lookup . . . 9
coord_slice . . . 9
coord_values . . . 10
coordinates . . . 10
DataArray . . . 11
Dataset . . . 11
ds_dims . . . 12
ExplicitCoord . . . 13
ImplicitCoord . . . 13
indexes . . . 14
indexing . . . 15
lazy-data . . . 16
names.ndr::Dataset . . . 17
nd_plan . . . 17
ndim . . . 18
open_dataset . . . 19
open_raster . . . 22
ops . . . 22
print-methods . . . 23
reductions . . . 23
set_affine_index . . . 24
Variable . . . 25

Index **27**

`$.ndr::Dataset` *Extract a DataArray from a Dataset*

Description

Use `ds$temperature` or `ds[["temperature"]]` to extract a variable as a DataArray with its relevant coordinates attached.

Usage

```
## S3 method for class '`ndr::Dataset`'
ds$var_name
```

Arguments

ds	A Dataset
var_name	Character name of the variable

Value

A DataArray

`AffineIndex` *Affine (geotransform) index*

Description

An `AffineIndex` owns two dimensions at once, the columns (`dims[1]`, usually x or lon) and rows (`dims[2]`, usually y or lat) of a raster grid, and maps positions to coordinates with the six numbers of a GDAL geotransform. Coordinates are never stored: they are computed from the transform when asked for, as in xarray's `rasterix.RasterIndex`.

Usage

```
AffineIndex(
  dims = character(0),
  shape = integer(0),
  transform = numeric(0),
  crs = character(0)
)
```

Arguments

dims	Character, length 2: the column dim then the row dim.
shape	Integer, length 2: number of columns, number of rows.
transform	Double, length 6: GDAL geotransform.
crs	Character: CRS as WKT, PROJJSON or an authority code (length 0 when unknown).

Details

The transform is in GDAL order, for pixel corners: `x = t[1] + col * t[2] + row * t[3]`, `y = t[4] + col * t[5] + row * t[6]`, so `t[3]` and `t[6]` are the rotation terms. Coordinates reported by the index are pixel centres.

Selecting a regular (evenly spaced) subset keeps the index affine, with the transform moved and scaled to match. An irregular subset of a north-up grid becomes two one-dimensional coordinates. The CRS travels with the index (one place, not two), and two indexes are only equal when their CRS agrees as well as their grid.

Examples

```
ix <- AffineIndex(
  dims = c("x", "y"), shape = c(360L, 180L),
  transform = c(-180, 1, 0, 90, 0, -1), crs = "EPSG:4326"
)
index_sel(ix, list(x = c(140, 150), y = c(-45, -40)))
as_geotransform(index_isel(ix, list(x = 321:330, y = 131:135))[[1]])
```

---

as.data.frame.ndr::DataArray

Coerce to data frame (long format)

Description

Expands a DataArray into a data frame with one row per cell, including coordinate values for each dimension. Compatible with ggplot2 and dplyr.

Usage

```
## S3 method for class '`ndr::DataArray`'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A DataArray
`row.names`	Ignored
`optional`	Ignored
`...`	Ignored

Value

A data.frame

Examples

```
da <- DataArray(
  variable = Variable(
    dims = c("lat", "lon"),
    data = matrix(1:6, 2, 3)
  ),
  coords = list(
    lat = ExplicitCoord(dimension = "lat", values = c(10, 20)),
    lon = ExplicitCoord(dimension = "lon", values = c(100, 110, 120))
  ),
  name = "value"
)
as.data.frame(da)
```

---

as.list.ndr::Dataset *Convert a Dataset to a list of data frames*

Description

Convert a Dataset to a list of data frames

Usage

```
## S3 method for class '`ndr::Dataset`'
as.list(x, ...)
```

Arguments

x	A Dataset
...	Ignored

Value

A named list of data.frames, one per data variable

---

as_geotransform *Get the GDAL geotransform of an AffineIndex*

Description

Get the GDAL geotransform of an AffineIndex

Usage

```
as_geotransform(x)
```

Arguments

x	An AffineIndex

Value

Named double vector of length 6, in GDAL order

---

as_variable *Quick Variable constructor from a named-dim array*

Description

Quick Variable constructor from a named-dim array

Usage

```
as_variable(data, ...)
```

Arguments

data	An R array
...	Dimension names as name = size pairs (ignored, names used) OR a character vector of dim names

Value

A Variable

Examples

```
# From a matrix with named dims
v <- as_variable(matrix(1:12, 3, 4), lat = 3, lon = 4)
```

---

broadcasting *Dimension-aware broadcasting*

Description

The core engine that makes `temperature * land_mask` work when the two variables have different (but compatible) dimensions. Broadcasting aligns by dimension name, inserts size-1 dimensions where needed, and expands arrays to a common shape.

cf_decode_time *Decode CF time values to R Date or POSIXct*

Description

Converts numeric offset values to R Date or POSIXct using CF convention time units and calendar. For standard/gregorian/proleptic_gregorian calendars (and NULL, the common case), uses base R date arithmetic. For non-standard calendars (360_day, noleap/365_day, all_leap/366_day), delegates to the CFtime package if available.

Usage

```
cf_decode_time(values, units, calendar = NULL)
```

Arguments

values	Numeric vector of time values (e.g. days since origin)
units	CF time units string (e.g. "days since 1800-01-01")
calendar	CF calendar type. NULL or "standard"/"gregorian"/ "proleptic_gregorian" use base R. Non-standard calendars require CFtime.

Value

Date vector (for whole-day units on standard calendars), POSIXct (for sub-day units), or character timestamps (non-standard calendars via CFtime).

Examples

```
# OISST: days since 1800-01-01 (standard calendar)
cf_decode_time(c(66443, 66474), "days since 1800-1-1 00:00:00")

# BRAN: days since 1979-01-01
cf_decode_time(c(11323, 11324), "days since 1979-01-01")
```

collect *Read lazy data into memory*

Description

Replaces lazy (altarr) data with an ordinary in-memory array, reading it with one planned read. Objects whose data is already in memory are returned unchanged.

Usage

```
collect(x, ...)
```

Arguments

x	A Variable, DataArray or Dataset.
...	Unused.

Value

An object of the same class with data in memory.

See Also

lazy-data

---

coord_dim *Get the dimension name of a coordinate*

Description

Get the dimension name of a coordinate

Usage

```
coord_dim(x)
```

Arguments

x	A coordinate

Value

Character scalar

---

coord_length *Get the length of a coordinate*

Description

Get the length of a coordinate

Usage

```
coord_length(x)
```

Arguments

x	A coordinate

Value

Integer length

`coord_lookup` *Look up integer index for a coordinate value*

Description

Returns the 1-based index of the element nearest to `value`. For ImplicitCoord, this is O(1) arithmetic. For ExplicitCoord on numeric values, binary search if sorted, linear scan otherwise.

Usage

```
coord_lookup(x, value)
```

Arguments

`x`	A coordinate
`value`	The value(s) to look up

Value

Integer index (1-based)

`coord_slice` *Slice a coordinate by integer indices*

Description

Slice a coordinate by integer indices

Usage

```
coord_slice(x, idx)
```

Arguments

`x`	A coordinate
`idx`	Integer indices (1-based)

Value

A new coordinate of the same type (ImplicitCoord stays implicit if the slice is a contiguous regular subsequence)

coord_values *Get the values of a coordinate*

Description

Get the values of a coordinate

Usage

```
coord_values(x)
```

Arguments

x	A coordinate (ImplicitCoord or ExplicitCoord)

Value

Vector of coordinate values

coordinates *Coordinate classes*

Description

A Coordinate is a one-dimensional Index. It maps integer indices to meaningful values along a dimension. Two representations:

Details

- **ImplicitCoord**: regular grids defined by offset + step. Coordinates are never stored, computed on demand via `offset + (0:(n-1)) * step`. This is the raster/terra cell abstraction, and xarray's RangeIndex.
- **ExplicitCoord**: an arbitrary vector of coordinate values (numeric, character, POSIXct, or anything with sensible comparison operators).

DataArray *DataArray: a Variable with coordinates*

Description

A DataArray wraps a Variable (the data) and attaches coordinates, so you can do things like `sel(da, lat = 0, time = "2020-01-15")`. This is the primary user-facing object for single-variable data.

Usage

```
DataArray(variable = Variable(), coords = list(), name = character(0))
```

Arguments

`variable`	A Variable
`coords`	Named list of indexes (ImplicitCoord, ExplicitCoord, AffineIndex, ...). Each index's dims must be among the Variable's `dims`, with matching sizes.
`name`	Optional name for this data array (character, length 0 or 1).

Examples

```
# 2D array with implicit spatial coords
v <- Variable(
  dims = c("lat", "lon"),
  data = matrix(rnorm(180 * 360), 180, 360),
  attrs = list(units = "K")
)
da <- DataArray(
  variable = v,
  coords = list(
    lat = ImplicitCoord(dimension = "lat", n = 180L, offset = -89.5, step = 1.0),
    lon = ImplicitCoord(dimension = "lon", n = 360L, offset = 0.5, step = 1.0)
  ),
  name = "temperature"
)
da
```

Dataset *Dataset: a collection of aligned Variables on shared dimensions*

Description

A Dataset holds multiple data variables that share a common set of dimensions and coordinates. The alignment constraint: any two variables that share a dimension name must agree on its size.

Usage

```
Dataset(data_vars = list(), coords = list(), attrs = list(), .backend = NULL)
```

Arguments

data_vars	Named list of Variable objects.
coords	Named list of coordinate objects (ImplicitCoord or ExplicitCoord).
attrs	Named list of global metadata.
.backend	Backend reader (internal, created by open_dataset()).

Examples

```
lat <- ImplicitCoord(dimension = "lat", n = 180L, offset = -89.5, step = 1.0)
lon <- ImplicitCoord(dimension = "lon", n = 360L, offset = 0.5, step = 1.0)

ds <- Dataset(
  data_vars = list(
    temperature = Variable(
      dims = c("lat", "lon"),
      data = matrix(rnorm(180*360), 180, 360),
      attrs = list(units = "K")
    ),
    pressure = Variable(
      dims = c("lat", "lon"),
      data = matrix(rnorm(180*360), 180, 360),
      attrs = list(units = "Pa")
    )
  ),
  coords = list(lat = lat, lon = lon),
  attrs = list(title = "Example dataset")
)
ds
```

---

ds_dims *List dimension names and sizes across a Dataset*

Description

List dimension names and sizes across a Dataset

Usage

```
ds_dims(ds)
```

Arguments

ds	A Dataset

Value

Named integer vector of all dimensions

`ExplicitCoord` *Create an explicit (irregular) coordinate*

Description

Create an explicit (irregular) coordinate

Usage

```
ExplicitCoord(dimension = character(0), values = NULL)
```

Arguments

`dimension`	Dimension name (character, length 1)
`values`	Vector of coordinate values

Examples

```
time <- ExplicitCoord(
  dimension = "time",
  values = as.Date("2020-01-01") + 0:364
)
coord_length(time)  # 365
```

`ImplicitCoord` *Create an implicit (regular-grid) coordinate*

Description

Create an implicit (regular-grid) coordinate

Usage

```
ImplicitCoord(
  dimension = character(0),
  n = integer(0),
  offset = numeric(0),
  step = numeric(0)
)
```

Arguments

dimension	Dimension name (character, length 1)
n	Number of elements (integer, length 1)
offset	Start value (numeric, length 1)
step	Step size (numeric, length 1)

Examples

```
lat <- ImplicitCoord(dimension = "lat", n = 180L, offset = -89.5, step = 1.0)
coord_values(lat)  # -89.5, -88.5, ..., 89.5
coord_lookup(lat, 0)  # integer index of latitude 0
```

---

indexes *Indexes: the contract between labels and positions*

Description

An index maps coordinate labels to integer positions along one or more dimensions. It is the object that `sel()` asks "which positions?", that `isel()` asks "what are you after this subset?", and that arithmetic asks "are these two objects on the same grid?". The design follows xarray's Index API (`sel`, `isel`, `equals`, `create_variables`), kept small so that other packages can implement new kinds of index.

Usage

```
Index()

index_dims(x)

index_sizes(x)

index_coords(x)

index_sel(x, labels)

index_isel(x, positions)

index_drop(x, dims)

index_equals(x, y)
```

Arguments

x, y	Index objects
labels	Named list of label values, keyed by dimension name
positions	Named list of 1-based integer positions, keyed by dimension name
dims	Character vector of dimension names

Details

Every entry in a DataArray or Dataset `coords` list is an `Index`. ImplicitCoord and ExplicitCoord are one-dimensional indexes; AffineIndex owns two dimensions at once through a geotransform.

An index implements these generics:

- `index_dims(x)`: the dimension names it covers.
- `index_sizes(x)`: named integer, the length of each of those dims.
- `index_coords(x)`: named list of coordinate values (one per coordinate it owns; a vector for 1D coordinates, a matrix for 2D).
- `index_sel(x, labels)`: `labels` is a named list of label values keyed by dimension; returns a named list of 1-based integer positions, one element per dimension that was selected on.
- `index_isel(x, positions)`: `positions` is a named list of 1-based integer positions keyed by dimension (dims not named are kept whole; a single position drops that dim); returns a list of indexes for the result (empty when every dim of the index was dropped).
- `index_drop(x, dims)`: list of indexes left when `dims` are removed entirely (by a reduction).
- `index_equals(x, y)`: `TRUE` when the two indexes describe the same labels.

Examples

```
lat <- ImplicitCoord(dimension = "lat", n = 180L, offset = -89.5, step = 1)
index_dims(lat)
index_sel(lat, list(lat = c(-10, 10)))
index_equals(lat, ExplicitCoord(dimension = "lat", values = seq(-89.5, 89.5)))
```

---

indexing *Coordinate-based indexing*

Description

`sel()` selects data by coordinate value (label-based). `isel()` selects data by integer index (position-based).

Usage

```
isel(.data, ...)

sel(.data, ...)
```

Arguments

`.data`	A Variable, DataArray, or Dataset
`...`	Named arguments specifying the selection. For `sel()`, values are coordinate values. For `isel()`, values are integer indices (1-based). Use a vector for ranges: `sel(da, lat = c(-30, 30))` selects the range. Use a scalar for a single position: `isel(da, time = 1)` picks one index and drops that dimension.

Details

Both work on Variables, DataArrays, and Datasets, and return objects of the same type with dimensions correctly updated.

Value

Same type as input, with dimensions sliced or dropped.

Examples

```
v <- Variable(
  dims = c("lat", "lon"),
  data = matrix(1:12, 3, 4)
)
da <- DataArray(
  variable = v,
  coords = list(
    lat = ImplicitCoord(dimension = "lat", n = 3L, offset = -10, step = 10),
    lon = ImplicitCoord(dimension = "lon", n = 4L, offset = 100, step = 10)
  )
)

# Select by coordinate value
sel(da, lat = 0)             # single latitude, drops lat dim
sel(da, lat = c(-10, 0))     # range of latitudes

# Select by integer index
isel(da, lon = 1:2)          # first two longitude columns
isel(da, lat = 2, lon = 3)   # single cell
```

---

lazy-data *Lazy data in a Variable*

Description

A Variable's `data` can be a lazy chunked array from the altarr package: an ordinary double, integer or logical vector with a `dim` attribute that reads its values chunk by chunk only when they are asked for. Variables read by `open_dataset()` hold such arrays, and any altarr array can be used directly:

Details

```
v <- Variable(dims = c("lon", "lat", "time"), data = altarr::altarr(...))
```

Every Variable, DataArray and Dataset method accepts lazy data:

- `dim()`, `shape()`, printing and coordinate work read nothing.
- `isel()` and `sel()` read nothing: they return a Variable whose data is a lazy view of the selection, still backed by the same chunks.

- Reductions (nd_mean() and friends) stream the array in blocks aligned to its chunks, one planned read per block, and never hold more than one block plus the result in memory.
- Arithmetic, as.array(), as.data.frame() and collect() read the (selected) values into memory with one planned read.
- nd_plan() says what such a read would cost before it happens: the chunks it touches and, with the zaro engine, the requests, bytes and hosts.

The block size for reductions is getOption("ndr.block_values") values (default 2^22); the chunk cache and fetch batching are altarr's options (see ?altarr::altarr_contract).

---

names.ndr::Dataset *Variable names in a Dataset*

Description

Returns all available variable names, including lazy (not-yet-loaded) variables from the backend.

Usage

```
## S3 method for class '`ndr::Dataset`'
names(x)
```

Arguments

x	A Dataset

Value

Character vector of variable names

---

nd_plan *What reading lazy data will cost*

Description

Works out what a read of lazy data (from open_dataset(), after any sel() or isel()) would touch, without reading it: the selection's size, the source chunks it needs, and with the zaro engine the requests, bytes, hosts and transports, from zaro::zaro_plan(). Call it before collect() on something large or remote.

Usage

```
nd_plan(x, ..., max_chunks = getOption("ndr.plan_max_chunks", 1e+05))
```

Arguments

x	a Variable, DataArray or Dataset.
...	unused.
max_chunks	the most chunks to plan byte references for: above this, only the counts are given (a whole large array can have millions of chunks). Default `getOption("ndr.plan_max_chunks", 1e5)`.

Details

With the GDAL engine the chunks are GDAL's block size and the requests are not estimated (they depend on the driver). In-memory data has nothing to read.

Value

An `nd_plan` (printed as a summary). For a Variable or DataArray, a list with `var`, `engine`, `lazy`, `selection` (the selected sizes of the source dims), `values`, `chunks` (touched), `chunks_total`, `chunk_shape`, `chunk_values` (values in the touched chunks), and with the zaro engine `requests`, `bytes` (known), `unknown_size`, `hosts`, `transport` and `refs` (the `zaro::zaro_plan()` table). For a Dataset, a data frame with one row per data variable.

Examples

```
## Not run:
ds <- open_dataset("https://host/store.zarr", engine = "zaro")
ds$temp |> sel(time = "2023-01-01") |> nd_plan()
nd_plan(ds)

## End(Not run)
```

---

ndim *Number of dimensions and shape of a Variable*

Description

`ndim()` is the number of dimensions; `shape()` is the named integer vector of dimension lengths.

Usage

```
ndim(x)

shape(x)
```

Arguments

x	A Variable or DataArray

Value

`ndim()`: an integer. `shape()`: a named integer vector.

open_dataset *Open a dataset from a file or URL*

Description

Read a multidimensional data source into an ndr Dataset, through one of two engines:

Usage

```
open_dataset(dsn, vars = NULL, ..., group = NULL, engine = NULL)
```

Arguments

dsn	Data source name. A file path, URL, or GDAL connection string (e.g. 'ZARR:"/vsicurl/https://examp
vars	Character vector of variable names to include. Default NULL includes all data variables. Use character() for schema + coords only. All variables are loaded lazily on first access via $.
...	Passed to zaro::zaro() for the "zaro" engine (e.g. anonymous = TRUE, region =); otherwise unused.
group	Path of the group to open in a nested store, e.g. "measurements/reflectance/r10m". Default NULL, the root group (or the group named in dsn after .zarr/, see Details).
engine	"gdal" or "zaro". The default picks "zaro" for zaro objects and for s3://, gs://, virtualizarr://, byterefs://, mdimrefs://, flatrefs:// and reference+json:///refere sources, and for local or http(s):// paths ending in .zarr (or with a group after it) when zaro is installed; else "gdal" (including ZARR: connection strings and /vsi paths).

Details

- "gdal": any source GDAL's multidim API can read (NetCDF, HDF5, Zarr v2/v3, kerchunk-parquet virtual stores, VRT multidim), from local paths, /vsicurl/, /vsis3/ and other GDAL virtual filesystems. Needs the GDAL7 package (remotes::install_github("rgdal-dev/GDAL7"), GDAL >= 3.10).
- "zaro": Zarr V2/V3 and virtual Zarr (kerchunk, VirtualiZarr Parquet, mdim-refs) read by the zaro package (remotes::install_github("hypertidy/zaro")), with no GDAL in the path. Takes anything zaro::zaro() opens, such as s3:// and gs:// URIs, virtualizarr://, byterefs:// (or mdimrefs://) and flatrefs:// (a byterefs flat refs store, read by row group) sources, or a store or zaro::zaro_dataset() object.

Reading data needs the altarr package (remotes::install_github("hypertidy/altarr")).

Lazy loading:

open_dataset() reads only coordinates and metadata. Accessing a data variable via ds$var_name returns a DataArray whose data is a lazy chunked array (see lazy-data): still no array data is read. sel() and isel() stay lazy, reductions stream the array chunk by chunk, and arithmetic or

collect() read the selected values. This allows opening large datasets (e.g. 12TB BRAN2023) without reading any array data. Use vars to limit which variables are available. Lazy reads need the altarr package (remotes::install_github("hypertidy/altarr")).

Variable classification:

Arrays are classified as coordinates or data variables based on CF conventions: a 1D array whose name matches its dimension name is treated as a coordinate. Every other array with one or more dimensions is a data variable, including 1D arrays along another dimension (such as a mean per band) and bounds arrays (e.g. time_bnds). Scalar arrays are skipped.

Numeric and logical data variables are lazy. String data variables can't be (lazy arrays are numeric), so they are read at open when they have at most getOption("ndr.eager_string_max", 1e6) elements, and skipped with a message when larger. String coordinates are read like any other. The GDAL engine reads strings with GDAL7 from rgdal-dev/GDAL7#15; with an older GDAL7 it skips string arrays with a message (string coordinates too, leaving that dimension without coordinate values).

Coordinate types:

Regular spatial grids (equal spacing within floating-point tolerance) are stored as ImplicitCoord (offset + step, no data allocation). Irregular grids and time coordinates are stored as ExplicitCoord.

CF time decoding:

Time dimensions (GDAL type "TEMPORAL") are automatically decoded from their CF units (e.g. "days since 1800-01-01") to R Date or POSIXct values using cf_decode_time().

The zaro engine:

Consolidated metadata is one request for the whole store. Data variables are CF-decoded as they are read: fill and missing values read as NA, and scale_factor/add_offset are applied. Those attributes move from the Variable's attrs to its encoding. Arrays without dimension names get names from zaro (see zaro::zaro_dataset()).

Arrays a store can't decode:

A byterefs store (byterefs://) names the arrays whose codecs zaro does not know, rather than let them be misread (byterefs decision 0007). zaro warns about them at open; the Dataset leaves them out of its coordinates and variables, lists them with the reason when printed, and asking for one (ds$v, vars = "v") is an error giving the reason.

Georeferenced grids:

A georeferenced grid becomes one AffineIndex, by the same rules in both engines:

- GeoZarr (spatial:dimensions, spatial:transform, spatial:registration, optional spatial:shape, on the group or the data variable): the transform defines the grid, so the x/y coordinate arrays are not read. The CRS comes from proj:code, proj:wkt2 or proj:projjson, else the CF grid mapping.
- CF: a data variable whose grid_mapping variable has crs_wkt (or spatial_ref), and whose first two dims (R order) have regular coordinates.
- GDAL engine only, before the CF route: a data variable with a CRS from GDAL whose HORIZONTAL_X / HORIZONTAL_Y dimensions (as netCDF declares them) have regular coordinates.

Groups:

A nested store (e.g. an EOPF Sentinel-2 product or other GeoZarr) keeps its arrays in groups below the root. Open one group with `group =`, or name it in the path: `"https://host/product.zarr/measurements/r10m"` is read as `dsn = "https://host/product.zarr"`, `group = "measurements/r10m"`, so the store's consolidated metadata (at its root) is used. The group's attributes become the Dataset's. List a store's groups with `zaro::zaro_groups()`.

Dimension ordering:

Arrays are stored in R's column-major (Fortran) order, matching GDAL7's `read_mdarray()$gis$dim` convention. Dimension names follow the same order. For a NetCDF variable with dimensions (time, lat, lon), the R array has `dim = c(nlon, nlat, ntime)` and `dims = c("lon", "lat", "time")`.

Value

A Dataset with coordinates, global attributes, and lazy data variables whose values are read only when used.

Examples

```
## Not run:
# Open lazily - no data read yet
ds <- open_dataset("sst.mnmean.nc")
ds  # shows variables with [not loaded]

# Selecting is lazy too; collect() reads just the selected values
ds$sst |> sel(time = as.Date("2020-06-15"), lat = c(-60, -30)) |> collect()

# Reductions stream the array chunk by chunk
ds$sst |> sel(lat = c(-60, -30)) |> nd_mean("time", na.rm = TRUE)

# Scope to specific variables (still lazy)
ds <- open_dataset("sst.mnmean.nc", vars = "sst")

# Remote kerchunk-parquet - only sst schema, 12TB never touched
dsn <- 'ZARR:"/vsicurl/https://example.com/store.parq"'
ds <- open_dataset(dsn, vars = "temp")
ds$temp  # a lazy DataArray: nothing read yet

# One group of a nested (sharded GeoZarr) store
eopf <- paste0("https://s3.explorer.eopf.copernicus.eu/esa-zarr-sentinel-explorer-fra/",
               "tests-output/sentinel-2-l2a/",
               "S2C_MSIL2A_20260120T084301_N0511_R064_T36UXA_20260120T122910.zarr")
r10m <- open_dataset(eopf, group = "measurements/reflectance/r10m")
r10m$b04 |> isel(x = 5001:6000, y = 5001:6000) |> collect()

## End(Not run)
```

open_raster *Open a raster band as a lazy DataArray*

Description

Opens one band of a classic (2D) GDAL raster, such as a GeoTIFF or COG, as a DataArray with lazy altarr data and an AffineIndex built from the dataset's geotransform and CRS. Nothing but metadata is read until values are asked for.

Usage

```
open_raster(dsn, band = 1L, dims = c("x", "y"))
```

Arguments

dsn	Data source name (file path, URL, or GDAL DSN string)
band	Band number (1-based)
dims	Names for the column and row dimensions

Value

A DataArray with dims `dims` (columns first, as GDAL7 returns them)

Examples

```
## Not run:
r <- open_raster("/vsicurl/https://example.com/dem.tif")
r  # coords: (x, y) affine ...
r |> sel(x = c(140, 150), y = c(-45, -40)) |> collect()

## End(Not run)
```

ops *Arithmetic operations on Variables and DataArrays*

Description

All arithmetic (+, -, *, /, etc.) and comparison (==, <, etc.) operators are supported. Operations broadcast by dimension name: dimensions present in one operand but not the other are automatically expanded.

Examples

```
# Broadcasting: 3D temperature * 2D land mask
temp <- Variable(
  dims = c("time", "lat", "lon"),
  data = array(rnorm(10 * 3 * 4), dim = c(10, 3, 4))
)
mask <- Variable(
  dims = c("lat", "lon"),
  data = matrix(c(1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0), 3, 4)
)

result <- temp * mask
shape(result)  # time=10, lat=3, lon=4

# Scalar operations
temp + 273.15
temp * 2
```

`print-methods` *Print methods for ndr objects*

Description

Displays dimension names, shapes, coordinate info, and attributes without loading or touching the underlying data.

`reductions` *Reductions along named dimensions*

Description

Summarise a Variable or DataArray by applying a function along one or more named dimensions. The reduced dimensions are dropped from the result.

Usage

```
nd_mean(x, dims, na.rm = FALSE)

nd_sum(x, dims, na.rm = FALSE)

nd_min(x, dims, na.rm = FALSE)

nd_max(x, dims, na.rm = FALSE)
```

Arguments

x	A Variable or DataArray
dims	Character vector of dimension names to reduce over
na.rm	Logical, whether to remove NAs

Value

Same type as input, with reduced dimensions dropped

Examples

```
temp <- Variable(
  dims = c("time", "lat", "lon"),
  data = array(rnorm(10 * 3 * 4), c(10, 3, 4))
)

# Time mean
nd_mean(temp, "time")  # shape: lat=3, lon=4

# Spatial mean (reduce lat and lon)
nd_mean(temp, c("lat", "lon"))  # shape: time=10

# Global mean (reduce everything)
nd_mean(temp, c("time", "lat", "lon"))  # scalar
```

---

set_affine_index *Combine two regular coordinates into an AffineIndex*

Description

Replaces the two ImplicitCoords for dims with one AffineIndex, so the grid and its CRS are carried, selected and compared as one thing.

Usage

```
set_affine_index(x, dims = c("x", "y"), crs = character())
```

Arguments

x	A DataArray or Dataset
dims	Character, length 2: the column (x) dim then the row (y) dim.
crs	Optional CRS string to attach.

Value

x with its coords updated

Examples

```
v <- Variable(dims = c("lon", "lat"), data = matrix(1:12, 4, 3))
da <- DataArray(variable = v, coords = list(
  lon = ImplicitCoord(dimension = "lon", n = 4L, offset = 100.5, step = 1),
  lat = ImplicitCoord(dimension = "lat", n = 3L, offset = -40.5, step = -1)
))
set_affine_index(da, c("lon", "lat"), crs = "EPSG:4326")
```

`Variable` *Variable: a named-dimension array*

Description

The fundamental building block. A Variable is an N-dimensional array that knows the names of its dimensions. It does NOT know about coordinates - that's DataArray's job.

Usage

```
Variable(
  dims = character(0),
  data = NULL,
  attrs = list(),
  encoding = list(),
  source = NULL
)
```

Arguments

`dims`	Character vector of dimension names.
`data`	An R array (or matrix, or vector with dim attribute), or a lazy chunked array from the altarr package (see lazy-data).
`attrs`	Named list of arbitrary metadata.
`encoding`	Named list of on-disk encoding info (scale_factor, etc.).
`source`	Where lazy data comes from, or `NULL`: set by `open_dataset()` and carried through `isel()` and `sel()`, so that `nd_plan()` can say what reading it will cost. Not for setting by hand.

Examples

```
# A 3D temperature field
temp_data <- array(rnorm(365 * 180 * 360), dim = c(365, 180, 360))
v <- Variable(
  dims = c("time", "lat", "lon"),
  data = temp_data,
  attrs = list(units = "K", long_name = "Temperature")
)
v
```

```
# Scalar variable
s <- Variable(dims = character(), data = array(42))
s
```

Index

$.ndr::Dataset, 2

AffineIndex, 3, *6*, *15*, *20*, *22*, *24*
as.data.frame.ndr::DataArray, 4
as.list.ndr::Dataset, 5
as_geotransform, 5
as_variable, 6

broadcasting, 6

cf_decode_time, 7
cf_decode_time(), *20*
collect, 7
collect(), *17*, *20*
coord_dim, 8
coord_length, 8
coord_lookup, 9
coord_slice, 9
coord_values, 10
coordinates, 10

DataArray, 11, *19*
Dataset, 11, *21*
ds_dims, 12

ExplicitCoord, 13, *15*, *20*

ImplicitCoord, 13, *15*, *20*, *24*
Index, *10*
Index *(*indexes*)*, 14
index_coords *(*indexes*)*, 14
index_dims *(*indexes*)*, 14
index_drop *(*indexes*)*, 14
index_equals *(*indexes*)*, 14
index_isel *(*indexes*)*, 14
index_sel *(*indexes*)*, 14
index_sizes *(*indexes*)*, 14
indexes, *11*, 14
indexing, 15
isel *(*indexing*)*, 15
isel(), *17*, *25*

lazy-data, *8*, 16, *19*, *25*

names.ndr::Dataset, 17
nd_max *(*reductions*)*, 23
nd_mean *(*reductions*)*, 23
nd_mean(), *17*
nd_min *(*reductions*)*, 23
nd_plan, 17
nd_plan(), *17*, *25*
nd_sum *(*reductions*)*, 23
ndim, 18

open_dataset, 19
open_dataset(), *12*, *16*, *17*, *25*
open_raster, 22
ops, 22

print-methods, 23

reductions, 23

sel *(*indexing*)*, 15
sel(), *17*, *25*
set_affine_index, 24
shape *(*ndim*)*, 18

Variable, 25

zaro::zaro(), *19*
zaro::zaro_dataset(), *19*, *20*
zaro::zaro_groups(), *21*
zaro::zaro_plan(), *17*, *18*