Package: tabularberg (via r-universe)

November 16, 2026

Title Icechunk Transactional Array Repositories

Version 0.0.0.9000

Copyright The Rust crates compiled into the package belong to their authors; see inst/AUTHORS.

Description Bindings to the 'Icechunk' Rust library: storage, repositories, branches, tags, snapshots, sessions, commits, conflict resolution, and virtual chunk references with explicit authorisation of the locations they point to. Sessions can be handed to other native code as bytes. Chunk references of any snapshot can be read as a table.

License Apache License (>= 2)

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

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

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

SystemRequirements Cargo (Rust's package manager), rustc >= 1.95

Imports DBI, jsonlite, methods, utils

Suggests testthat (>= 3.0.0), RSQLite, duckdb, nanoarrow, dbplyr, dplyr, arrow, nanoparquet, openssl, pizzarr, R6, blob, bit64, curl, withr, knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/rextendr/version 0.5.0

Config/roxygen2/version 8.1.1

Config/pak/sysreqs libclang-dev

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

Date/Publication 2026-11-16 11:05:24 UTC

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

RemoteRef HEAD

RemoteSha b3497ff1756855ad672a610211c2783c703a0737

Contents

ic_byterefs_compare . . . 2
ic_byterefs_containers . . . 3
ic_byterefs_read . . . 4
ic_chunk_refs . . . 5
ic_commit . . . 6
ic_compiled_features . . . 7
ic_credentials . . . 8
ic_example_oisst . . . 9
ic_example_repo . . . 10
ic_manifest_concurrency . . . 11
ic_pizzarr_store . . . 12
ic_refs . . . 13
ic_refs_arrow . . . 14
ic_repo_containers . . . 15
ic_repo_from_byterefs . . . 16
ic_repo_open . . . 17
ic_session_bytes . . . 18
ic_session_readonly . . . 20
ic_set_virtual_ref . . . 21
ic_storage . . . 22
ic_store . . . 24
ic_to_byterefs . . . 25
ic_write_byterefs . . . 26
Icechunk . . . 27

Index **32**

`ic_byterefs_compare` *Compare two byterefs stores*

Description

The conformance check for a round trip: array descriptions must agree (dims, shape, chunks, dtype, fill, codec, attributes, coordinate values) and every array's global refs rows must agree on index, presence, resolved location, offset, size and inline bytes. Source paths are compared after each side's own remap for `target`.

Usage

```
ic_byterefs_compare(x, y, target = NULL)
```

Arguments

x, y	byterefs stores (see ic_byterefs_read()).
target	remap target used to resolve paths on both sides.

Value

a data frame of checks: array, check, ok, detail.

Examples

```
ex <- ic_example_repo()
repo <- ic_repo_open(ex$path, authorize = ex$authorize)
now <- ic_to_byterefs(ic_session_readonly(repo, branch = "main"))
then <- ic_to_byterefs(ic_session_readonly(repo, tag = "v1"))
cmp <- ic_byterefs_compare(then, now)
cmp[!cmp$ok, ]
```

---

ic_byterefs_containers

Virtual chunk containers needed by a byterefs store

Description

One container per scheme and host (http/https), bucket (s3, gcs), or directory (file), as the locations resolve under target.

Usage

```
ic_byterefs_containers(
  br,
  target = NULL,
  region = Sys.getenv("AWS_REGION", "us-east-1")
)
```

Arguments

br	a byterefs store (see ic_byterefs_read()).
target	remap target; default meta.default_target, else "public", else no remapping.
region	S3 region for s3 containers.

Value

a list of ic_container() objects; the prefixes attribute holds the URL prefixes, suitable for authorize.

Examples

```
ex <- ic_example_repo()
br <- ic_to_byterefs(ic_session_readonly(ic_repo_open(ex$path, authorize = ex$authorize), branch = "main"))
ic_byterefs_containers(br)
```

`ic_byterefs_read` *Read a byterefs store*

Description

Reads a byterefs table, either the SQLite working store (`store.sqlite`, or an open DBI connection) or the Parquet "flat" export (a directory holding `arrays.json`, `sources.parquet` and `refs_<array>.parquet`). `byterefs/0.1` and its predecessor `mdim-refs/0.2` are accepted.

Usage

```
ic_byterefs_read(x)
```

Arguments

`x`	path to a `.sqlite` file, a flat-export directory, a DBI connection, or a `byterefs` object (returned as is).

Value

a `byterefs` object: a list with `format_version`, `meta` (named list of strings), `sources`, `arrays` and `remap` data frames, and `refs(array)`, a function returning that array's global refs rows (`dim_0..`, `present`, `path`, `offset`, `size`, `info`, `data`) joined to the source's `etag`, `mtime` and `source_size`. Read from SQLite, `refs()` queries the file lazily, so the connection stays open: call `$close()` when done (functions given a file path close it themselves).

Examples

```
ex <- ic_example_repo()
s <- ic_session_readonly(ic_repo_open(ex$path, authorize = ex$authorize), branch = "main")
# write the store's references as a byterefs SQLite file, read it back
if (requireNamespace("RSQLite", quietly = TRUE)) {
  db <- tempfile(fileext = ".sqlite")
  ic_to_byterefs(s, dsn = db)
  br <- ic_byterefs_read(db)
  print(br)
  print(br$arrays[, c("array", "kind", "shape", "chunks", "dtype")])
  print(head(br$refs("from_bin")))
  br$close()
}
```

ic_chunk_refs *Nodes and chunk references*

Description

ic_chunk_refs() returns one row per stored chunk of a snapshot (or of one array): its Zarr key and where its bytes are. kind is "native" (bytes in the repository, location is chunks/<id> relative to the repository root), "virtual" (bytes in another object, location is its URL) or "inline" (bytes in the manifest). Chunks never written are absent (fill value). This is the table a reader needs to fetch chunks itself, and to check a repository against the reference table it was built from.

Usage

```
ic_chunk_refs(session, path = NULL, inline = FALSE)

ic_nodes(session)

ic_virtual_locations(session)
```

Arguments

session	an icechunk_session.
path	an array path ("/temp"), or NULL for all arrays.
inline	include inline chunk bytes as a list column.

Value

a data frame: path, coord (comma-separated 0-based chunk index), key, kind, location, offset, length, etag, last_modified (and inline).

Examples

```
ex <- ic_example_repo()
s <- ic_session_readonly(ic_repo_open(ex$path, authorize = ex$authorize), branch = "main")
ic_nodes(s)
ic_chunk_refs(s, "/from_bin")[, c("coord", "kind", "location", "offset", "length")]
ic_chunk_refs(s, inline = TRUE)[, c("path", "coord", "kind", "length")]
ic_virtual_locations(s)
```

`ic_commit` *Commit, amend, flush, discard, rebase*

Description

Commit, amend, flush, discard, rebase

Usage

```
ic_commit(session, message, metadata = NULL, allow_empty = FALSE)

ic_amend(session, message, metadata = NULL)

ic_flush(session, message, metadata = NULL)

ic_discard(session)

ic_rebase(
  session,
  solver = c("detect", "basic"),
  on_chunk = c("ours", "theirs", "fail")
)

ic_status(session)
```

Arguments

`session`	a writable `icechunk_session`.
`message`	commit message.
`metadata`	a named list stored with the snapshot.
`allow_empty`	commit even with no changes.
`solver`	`"detect"` fails on any conflict; `"basic"` resolves chunk conflicts by `on_chunk`.
`on_chunk`	for `solver = "basic"`: `"ours"`, `"theirs"` or `"fail"`.

Details

A commit that loses a race raises a condition of class `icechunk_conflict`; call `ic_rebase()` and commit again.

Value

new snapshot id (commit/amend/flush).

Examples

```
ex <- ic_example_repo()
repo <- ic_repo_open(ex$path, authorize = ex$authorize)
s <- ic_session_writable(repo, "main")
ic_set(s, "zarr.json", charToRaw('{"zarr_format":3,"node_type":"group","attributes":{"v":2}}'))
ic_status(s)
id <- ic_commit(s, "attributes, take one", metadata = list(author = "example"))
ic_session_info(s)$read_only     # a committed session is done

# replace that commit instead of adding another
s <- ic_session_writable(repo, "main")
ic_set(s, "zarr.json", charToRaw('{"zarr_format":3,"node_type":"group","attributes":{"v":3}}'))
ic_amend(s, "attributes, take two")
ic_ancestry(repo, branch = "main")$message[1:2]

# two writers: the second commit conflicts, rebase onto the new head
a <- ic_session_writable(repo, "main")
b <- ic_session_writable(repo, "main")
ic_set(a, "small/c/0/0", writeBin(as.double(1:6), raw(), size = 8, endian = "little"))
ic_set(b, "small/c/1/1", writeBin(as.double(7:12), raw(), size = 8, endian = "little"))
ic_commit(a, "writer a")
res <- tryCatch(ic_commit(b, "writer b"), icechunk_conflict = function(e) e)
class(res)
ic_rebase(b)
ic_commit(b, "writer b, rebased")
```

---

ic_compiled_features *Build information*

Description

Build information

Usage

```
ic_compiled_features()

ic_icechunk_version()
```

Value

`ic_compiled_features()` the Cargo features in this build; `ic_icechunk_version()` the icechunk crate version.

Examples

```
ic_compiled_features()
ic_icechunk_version()
```

`ic_credentials` *Credentials and virtual chunk containers*

Description

Virtual chunk references point outside the repository. A repository declares *containers* (URL prefixes plus how to reach them), and the reader must *authorise* each prefix it is willing to read, with credentials. Nothing is authorised by default: this is the "where" control.

Usage

```
ic_cred_s3(
  kind = c("from_env", "anonymous", "static"),
  access_key_id = NULL,
  secret_access_key = NULL,
  session_token = NULL
)

ic_cred_http()

ic_cred_local()

ic_cred_gcs(kind = c("from_env", "anonymous"))

ic_cred_azure(kind = c("from_env", "anonymous"))

ic_container(
  url_prefix,
  type = c("s3", "s3_compatible", "http", "local", "gcs", "in_memory"),
  region = NULL,
  endpoint_url = NULL,
  anonymous = FALSE,
  allow_http = FALSE,
  force_path_style = FALSE,
  path = NULL
)
```

Arguments

`kind`	for S3: `"from_env"`, `"anonymous"` or `"static"`; for GCS and Azure: `"from_env"` or `"anonymous"`.
`access_key_id`, `secret_access_key`, `session_token`	static S3 keys.
`url_prefix`	the prefix chunk locations must start with, e.g. `"s3://bucket/path/"`, `"https://host/path/"`, `"file:///data/"`.
`type`	store type for the container.

region, endpoint_url, anonymous, allow_http, force_path_style
: S3 options for type = "s3" or "s3_compatible".

path
: for type = "local".

Value

a credential spec (a list), for authorize in ic_repo_open().

Examples

```
# a virtual chunk container: where chunk bytes may come from
ic_container("https://noaa-cdr-sea-surface-temp-optimum-interpolation-pds.s3.amazonaws.com/", "http")
ic_container("s3://my-bucket/data/", "s3", region = "us-west-2")
ic_container("file:///data/netcdf/", "local", path = "/data/netcdf")
# credentials for storage
ic_cred_s3("anonymous")
## Not run:
ic_cred_s3("static", access_key_id = "AKIA...", secret_access_key = "...")

## End(Not run)
```

---

ic_example_oisst *A small real repository: ten days of NOAA OISST*

Description

Unpacks one of two Icechunk repositories shipped with the package: ten days of NOAA OISST v2.1 daily sea surface temperature (September 1981), from zarr-icechunk-assure. The two are the same repository built twice, with the virtual references written for NOAA's https:// archive or its s3:// bucket. Metadata, coordinates and history read offline; chunk values are fetched from NOAA when read, so they need the network. Each has two tags: as-virtualized (the time array lacks its units attribute, a defect kept on purpose) and cf-repaired (the units added by a metadata-only commit).

Usage

```
ic_example_oisst(cell = c("https", "s3"), path = tempfile("oisst-"))
```

Arguments

cell
: "https" or "s3": how the references were written.

path
: directory to unpack into (must not exist).

Value

A list with path (the repository), authorize (a named list: the container's URL prefix and an anonymous credential, ready for ic_repo_open()), probe (the contract from probe.json: shape, codecs, prefixes and one probe pixel) and cell.

See Also

ic_example_repo() for a synthetic repository; the files are in system.file("extdata", "oisst-sample", package = "tabularberg").

Examples

```
ox <- ic_example_oisst("https")
repo <- ic_repo_open(ox$path)
ic_tags(repo)

# the repair between the two tags touched metadata only
ic_diff(repo, from = list(tag = "as-virtualized"), to = list(tag = "cf-repaired"))

# coordinates are stored in the repository; data chunks point at NOAA
s <- ic_session_readonly(repo, tag = "cf-repaired")
ic_chunk_refs(s, "/sst")[1:2, c("key", "location", "length")]

# reading a data chunk needs the container authorised (and the network)
s <- ic_session_readonly(ic_repo_open(ox$path, authorize = ox$authorize), tag = "cf-repaired")
length(ic_get(s, "sst/c/0/0/0/0"))
```

---

ic_example_repo *A small example repository*

Description

Builds a self-contained Icechunk repository in a temporary directory, for examples and experiments. Nothing is downloaded. It holds:

Usage

```
ic_example_repo(path = tempfile("icechunk-example-"))
```

Arguments

path	directory to create (must not exist).

Details

- /from_bin: float32, 64 x 48, chunks 16 x 48. Four virtual references into a raw binary file grid.bin, one per 16-row block.
- /from_nc: float32, 64 x 48, one chunk. A virtual reference into grid.nc at offset 2048 (a 2048-byte header, then the same values).
- /small: float64, 4 x 6, chunks 2 x 3, values 1:24 in C order, written through the session in a second commit. Its chunks are small enough (48 bytes) that Icechunk keeps them inline in the manifest.

The values of the virtual arrays are `(0:3071) / 4`, little-endian float32 in C order. Both source files live in a directory next to the repository, declared as a virtual chunk container. Like any container, it must be authorised before its chunks are read: pass `$authorize` to `ic_repo_open()` or `DBI::dbConnect()`. The first commit is tagged `v1`.

Value

A list with `path` (the repository), `authorize` (the container's URL prefix, `file://...`), `source` (the directory holding `grid.bin` and `grid.nc`) and `values` (the virtual arrays' values).

Examples

```
ex <- ic_example_repo()
repo <- ic_repo_open(ex$path, authorize = ex$authorize)
repo
ic_nodes(ic_session_readonly(repo, branch = "main"))
```

---

ic_manifest_concurrency

Concurrent manifest fetches

Description

Reading an array's chunk references (`ic_chunk_refs()`, `ic_to_byterefs()`, `ic_refs_arrow()`, the DBI `refs_<array>` tables) needs every manifest that covers it. With split manifests on remote storage, fetching them one after another dominates; tabularberg first loads them into the session's cache this many at a time.

Usage

```
ic_manifest_concurrency(n = 32L)
```

Arguments

`n`	manifests fetched at once (1 = sequential).

Value

the previous value (invisibly).

Examples

```
old <- ic_manifest_concurrency(8L)
old
ic_manifest_concurrency(old)
```

`ic_pizzarr_store` *A pizzarr store over an Icechunk session*

Description

Returns an R6 store that pizzarr's pure-R code path accepts (it inherits `pizzarr::Store`), so pizzarr without its Rust backend, as released on CRAN, can read and write an Icechunk repository. Every key read or written goes through the session: reads see the session's snapshot plus its uncommitted changes, and writes stay in the session until `ic_commit()`.

Usage

```
ic_pizzarr_store(session)
```

Arguments

`session`	an `icechunk_session` (read-only, or writable to write).

Details

The store reports no store identifier, so pizzarr always uses its R code path with it (for pizzarr's zarrs backend, use pizzarr's own `IcechunkStore`, which takes the session as bytes).

Limits come from pizzarr's R decoder, not from Icechunk: Zarr v3 arrays whose codecs it does not implement cannot be read this way (`sharding_indexed`; and, in pizzarr 0.2.x as released, the netCDF-4 `numcodecs.shuffle` and `numcodecs.zlib`, which the mdsumner/pizzarr fork adds). `ic_get()` still returns their raw chunk bytes.

Value

an R6 object inheriting `pizzarr::Store`.

Examples

```
if (requireNamespace("pizzarr", quietly = TRUE) && requireNamespace("R6", quietly = TRUE)) {
  ex <- ic_example_repo()
 s <- ic_session_readonly(ic_repo_open(ex$path, authorize = ex$authorize), branch = "main")
  st <- ic_pizzarr_store(s)
  print(st)
  z <- pizzarr::zarr_open(st, mode = "r", path = "small")
  print(z$get_item("...")$data)
  # the virtual array, decoded from the source file's bytes
  v <- pizzarr::zarr_open(st, mode = "r", path = "from_nc")$get_item("...")$data
  all.equal(as.vector(t(v)), ex$values)
}
```

ic_refs *Branches, tags, ancestry and diffs*

Description

Branches, tags, ancestry and diffs

Usage

```
ic_branches(repo)

ic_tags(repo)

ic_ancestry(repo, branch = NULL, tag = NULL, snapshot = NULL)

ic_branch_create(repo, name, snapshot_id)

ic_branch_reset(repo, name, snapshot_id, from = NULL)

ic_branch_delete(repo, name)

ic_tag_create(repo, name, snapshot_id)

ic_tag_delete(repo, name)

ic_lookup(repo, branch = "main", tag = NULL)

ic_diff(repo, from, to)
```

Arguments

repo	an icechunk_repo.
branch, tag, snapshot	select one version (default branch "main").
name	branch or tag name.
snapshot_id	a snapshot id.
from	for ic_branch_reset(): optional expected current tip (the reset fails if the branch has moved); for ic_diff(): the starting version, as for to.
to	for ic_diff(): a list like list(branch = "main"), list(tag = "v1") or list(snapshot = id), or a snapshot id.

Value

data frames for listings; snapshot ids as character.

Examples

```
ex <- ic_example_repo()
repo <- ic_repo_open(ex$path, authorize = ex$authorize)
ic_branches(repo)
ic_tags(repo)
ic_ancestry(repo, branch = "main")[, c("snapshot_id", "message")]

# a branch from the tagged snapshot, and what changed since
v1 <- ic_lookup(repo, tag = "v1")
ic_branch_create(repo, "dev", v1)
ic_diff(repo, from = list(tag = "v1"), to = list(branch = "main"))
ic_branch_delete(repo, "dev")

# a real repository: a metadata repair recorded as a commit, no references rewritten
oisst <- ic_repo_open(ic_example_oisst()$path)
ic_ancestry(oisst, tag = "cf-repaired")[, c("snapshot_id", "message")]
ic_diff(oisst, from = list(tag = "as-virtualized"), to = list(tag = "cf-repaired"))
```

`ic_refs_arrow` *Chunk references of one array as an Arrow stream*

Description

The byterefs refs view of an array (`dim_0..`, `present`, `path`, `offset`, `size`, `info`, `data`), built in Rust as Arrow record batches and handed over as a `nanoarrow_array_stream`, without R vectors in between. `path` is dictionary-encoded (one entry per source), `offset` and `size` are int64, as the byterefs spec has them. Paths have the repository's remap rules undone (see `ic_write_byterefs()`); native chunks (bytes held by the repository) have no path, and their bytes in `data` when `native = TRUE`, as `ic_to_byterefs()` writes them.

Usage

```
ic_refs_arrow(session, path, chunks = NULL, batch_rows = 65536L, native = TRUE)
```

Arguments

`session`	an `icechunk_session`.
`path`	array path.
`chunks`	optional slab, `list(dim_0 = c(lo, hi), ...)` of chunk indices (hi exclusive); only the manifests covering it are read.
`batch_rows`	rows per record batch.
`native`	fetch native chunks' bytes into `data` (concurrently, see `ic_manifest_concurrency()`); `FALSE` leaves them null.

Details

The stream is lazy: it reads the session as it is when called (later writes to the session are not seen), and builds each batch as the consumer asks for it, with one batch of read-ahead, so a large array's refs never sit in memory whole. Errors met while streaming (a manifest that cannot be fetched) are raised by the consumer.

Consume with `nanoarrow::convert_array_stream()`, `arrow::as_arrow_table()`, `arrow::write_dataset()` or `duckdb::duckdb_register_arrow()`.

Value

a `nanoarrow_array_stream`.

Examples

```
ex <- ic_example_repo()
s <- ic_session_readonly(ic_repo_open(ex$path, authorize = ex$authorize), branch = "main")
if (requireNamespace("nanoarrow", quietly = TRUE)) {
  st <- ic_refs_arrow(s, "from_bin", batch_rows = 2L)
  print(st$get_schema())
  # batches are built as they are pulled
 while (!is.null(b <- st$get_next())) print(nanoarrow::convert_array(b)[, c("dim_0", "offset", "size")])
  # one slab, with chunk bytes for inline/native chunks
 print(nanoarrow::convert_array_stream(ic_refs_arrow(s, "small", chunks = list(dim_0 = 1))))
}
```

`ic_repo_containers` *Virtual chunk containers of a repository*

Description

Virtual chunk containers of a repository

Usage

```
ic_repo_containers(repo)
```

Arguments

`repo`	an `icechunk_repo`.

Value

a data frame: `prefix` (URL prefix of each declared container) and `authorized` (whether this repository object may read from it).

Examples

```
ex <- ic_example_repo()
repo <- ic_repo_open(ex$path)
ic_repo_containers(repo)           # declared, not yet authorised
repo <- ic_repo_open(ex$path, authorize = "file://")
ic_repo_containers(repo)           # a broader prefix covers it
```

`ic_repo_from_byterefs` *Create an Icechunk repository from a byterefs store*

Description

Creates the repository with one virtual chunk container per source prefix (`ic_byterefs_containers()`), authorises them, writes the store with `ic_write_byterefs()` and commits.

Usage

```
ic_repo_from_byterefs(
  br,
  storage,
  target = NULL,
  message = NULL,
  authorize = NULL,
  manifest_split = NULL,
  ...
)
```

Arguments

`br`	a byterefs store (see `ic_byterefs_read()`).
`storage`	an ic_storage object, or a local path, for the new repository.
`target`	remap target.
`message`	commit message.
`authorize`	credentials for the containers; default inferred from each prefix's scheme (see `ic_repo_open()`).
`manifest_split`	manifest splitting, see `ic_repo_create()`; e.g. `c(time = 1)` for a daily series, so a day's refs are one manifest.
`...`	passed to `ic_write_byterefs()`.

Value

the `icechunk_repo` (invisibly), with attributes `snapshot` and `summary`.

Examples

```
ex <- ic_example_repo()
s <- ic_session_readonly(ic_repo_open(ex$path, authorize = ex$authorize), branch = "main")
br <- ic_to_byterefs(s)

# a new repository from the references: same chunks, same bytes
dst <- tempfile("copy-")
repo2 <- ic_repo_from_byterefs(br, dst, authorize = ex$authorize, message = "a copy")
s2 <- ic_session_readonly(repo2, branch = "main")
identical(ic_get(s2, "from_bin/c/2/0"), ic_get(s, "from_bin/c/2/0"))
ic_byterefs_compare(br, ic_to_byterefs(s2))

# with one manifest per chunk row of dimension 0
dst3 <- tempfile("split-")
ic_repo_from_byterefs(br, dst3, authorize = ex$authorize, manifest_split = c("0" = 1))
length(list.files(file.path(dst3, "manifests")))
```

---

ic_repo_open *Open or create an Icechunk repository*

Description

Open or create an Icechunk repository

Usage

```
ic_repo_open(storage, containers = NULL, authorize = NULL)

ic_repo_create(
  storage,
  containers = NULL,
  authorize = NULL,
  spec_version = NULL,
  manifest_split = NULL
)

ic_repo_exists(storage)

ic_repo_save_config(repo)

ic_repo_config(repo)
```

Arguments

storage	an ic_storage object, or a local path.
containers	virtual chunk containers (ic_container()) to layer on the persisted config (use ic_repo_save_config() to persist).

authorize	virtual chunk prefixes this session may read: NULL (none), a character vector of prefixes (credentials inferred from the scheme: s3 from env, http(s) anonymous, file local), or a named list of prefix = ic_cred_s3() etc. Icechunk authorises whole virtual chunk containers (see ic_repo_containers()): a prefix that is a container's prefix, or broader than one (e.g. "https://"), authorises every declared container it covers, with those credentials; a prefix inside a container is an error naming the container; a prefix that covers no container is a warning listing the declared ones.
spec_version	format version for new repositories (1 or 2; NULL for the library default).
manifest_split	manifest splitting for new repositories: NULL (one manifest per array), a named integer vector of chunks per manifest along each dimension, by axis index or dimension name (e.g. c(time = 1) or c("0" = 30); "*" for any dimension), applied to every array, or a list of rules list(path = <regex or NULL>, dims = <as before>) tried in order. Split manifests let a read of a slab (see ic_to_byterefs(), the DBI refs_<array> tables) fetch only the manifests that cover it. Persist with ic_repo_save_config().
repo	an icechunk_repo.

Value

an icechunk_repo.

Examples

```
# create a repository, write a group, commit
path <- tempfile("ic-repo-open-")
repo <- ic_repo_create(path)
s <- ic_session_writable(repo, "main")
ic_set(s, "zarr.json", charToRaw('{"zarr_format":3,"node_type":"group","attributes":{}}'))
ic_commit(s, "an empty group")
ic_repo_exists(path)

# open it again; a repository with virtual chunk containers needs them
# authorised before their chunks are read
ex <- ic_example_repo()
repo <- ic_repo_open(ex$path, authorize = ex$authorize)
repo
ic_repo_config(repo)$virtual_chunk_containers[[1]]$url_prefix
unlink(path, recursive = TRUE)
```

---

ic_session_bytes *Hand a session to other native code*

Description

ic_session_bytes() serialises a session (Icechunk's own Session::as_bytes) behind a small header: "ICR1", a u16 little-endian length, the icechunk crate version, then the payload. Any consumer built against the same icechunk version can rebuild the session (e.g. pizzarr's zarrs backend) and read or write at native speed.

Usage

```
ic_session_bytes(session)

ic_session_from_bytes(bytes)

ic_session_fork(session)

ic_session_merge(session, bytes)
```

Arguments

session	an icechunk_session.
bytes	a raw vector from ic_session_bytes().

Details

For writes, hand out a ic_session_fork() of a writable session, take the consumer's bytes back with ic_session_merge(), then ic_commit() the base session. Commit authority stays here.

The bytes carry credentials. They hold the repository storage (bucket, prefix, endpoint, and its credentials) and the virtual chunk authorisations the repository was opened with. Static keys given to ic_storage_s3() or to ic_cred_s3()("static", ...) are in the bytes in plain text (verified for icechunk 2.3.1: S3ObjectStoreBackend and S3Storage serialise their credentials field, and so does the virtual chunk resolver). "from_env" and "anonymous" credentials carry no secret: the consumer resolves them from its own environment. Prefer those when bytes leave the process, and never log or persist bytes made with static keys.

Value

raw vector; a session; or nothing.

Examples

```
ex <- ic_example_repo()
repo <- ic_repo_open(ex$path, authorize = ex$authorize)
s <- ic_session_writable(repo, "main")

# hand a session to another process (or package) as bytes ...
b <- ic_session_bytes(s)
b
s2 <- ic_session_from_bytes(b)
ic_set(s2, "small/c/0/0", writeBin(as.double(6:1), raw(), size = 8, endian = "little"))

# ... and merge its changes back before committing
ic_session_merge(s, ic_session_bytes(s2))
ic_status(s)
ic_commit(s, "written elsewhere")
```

`ic_session_readonly` *Sessions*

Description

A session is a view of one snapshot. Read-only sessions are pinned to a branch tip, tag or snapshot id; writable sessions start from a branch tip and accumulate changes until `ic_commit()`.

Usage

```
ic_session_readonly(repo, branch = NULL, tag = NULL, snapshot = NULL)

ic_session_writable(repo, branch = "main")

ic_session_info(session)
```

Arguments

`repo`	an `icechunk_repo`.
`branch, tag, snapshot`	select one version (default branch "main").
`session`	an `icechunk_session`.

Value

an `icechunk_session`.

Examples

```
ex <- ic_example_repo()
repo <- ic_repo_open(ex$path, authorize = ex$authorize)

# read the branch head, a tag, or a snapshot
s <- ic_session_readonly(repo, branch = "main")
ic_session_info(s)
then <- ic_session_readonly(repo, tag = "v1")
ic_nodes(then)$path

# a writable session collects changes until ic_commit()
w <- ic_session_writable(repo, "main")
ic_set(w, "zarr.json", charToRaw('{"zarr_format":3,"node_type":"group","attributes":{"edited":true}}'))
ic_session_info(w)$has_uncommitted_changes
ic_discard(w)

# one array's metadata at two tags of a real repository
oisst <- ic_repo_open(ic_example_oisst()$path)
units <- function(tag) {
  meta <- ic_get(ic_session_readonly(oisst, tag = tag), "time/zarr.json")
```

```
  jsonlite::fromJSON(rawToChar(meta))$attributes$units
}
units("as-virtualized")
units("cf-repaired")
```

`ic_set_virtual_ref` *Write virtual chunk references*

Description

Write virtual chunk references

Usage

```
ic_set_virtual_ref(
  session,
  key,
  location,
  offset,
  length,
  etag = NULL,
  last_modified = NULL,
  validate = TRUE
)

ic_set_virtual_refs(session, path, refs, validate = TRUE)
```

Arguments

`session`	a writable `icechunk_session`.
`key`	Zarr chunk key.
`location`	URL of the object holding the bytes.
`offset`, `length`	byte range in that object.
`etag`, `last_modified`	optional checksum: reads fail if the object has changed.
`validate`	check the location falls within a declared container.
`path`	array path.
`refs`	data frame of 0-based chunk indices and byte ranges. Indices are either `dim_0, dim_1, ...` integer columns (the byterefs layout) or a `coord` column (list of integer vectors, or comma-separated strings). Byte ranges are `location` (or `path`), `offset` and `length` (or `size`). Optional `etag` and `last_modified` (POSIXct or seconds) set a per-chunk checksum; etag wins when both are present.

Value

number of refs that failed validation (invisibly).

Examples

```
ex <- ic_example_repo()
repo <- ic_repo_open(ex$path, authorize = ex$authorize)
s <- ic_session_writable(repo, "main")
grid <- paste0(ex$authorize, "grid.bin")

# point from_bin's last chunk at the first block of grid.bin instead
ic_set_virtual_ref(s, "from_bin/c/3/0", grid, offset = 0, length = 3072)

# or many at once, one row per chunk (dim_* are chunk indices)
ic_set_virtual_refs(s, "/from_bin", data.frame(
  dim_0 = 0:1, dim_1 = 0L, location = grid, offset = c(3072, 0), length = 3072))
ic_chunk_refs(s, "/from_bin")[, c("coord", "offset")]
ic_commit(s, "rearranged from_bin")
```

---

ic_storage *Icechunk storage*

Description

Where the repository lives. Storage objects hold no open connections and are cheap; credentials given here are never printed.

Usage

```
ic_storage_local(path)

ic_storage_memory()

ic_storage_http(base_url)

ic_storage_s3(
  bucket,
  prefix = NULL,
  region = NULL,
  endpoint_url = NULL,
  anonymous = FALSE,
  allow_http = FALSE,
  force_path_style = !is.null(endpoint_url),
  requester_pays = FALSE,
  credentials = NULL,
  backend = c("object_store", "aws")
)

ic_storage_gcs(bucket, prefix = NULL, anonymous = FALSE)

ic_storage_azure(account, container, prefix = NULL)
```

Arguments

path	directory of the repository (local).
base_url	URL of the repository root (read-only HTTP).
bucket, prefix	bucket and key prefix of the repository.
region, endpoint_url	S3 region and endpoint (for S3-compatible services such as Pawsey Acacia or data.source.coop).
anonymous	unsigned requests.
allow_http	allow http:// endpoints.
force_path_style	path-style addressing (most S3-compatible services need this).
requester_pays	requester-pays bucket.
credentials	an ic_cred_s3() spec, or NULL for icechunk's default (from the environment).
backend	"object_store" (default) or "aws" (the aws-sdk backend, only in the full build).
account, container	Azure account and container.

Value

an icechunk_storage object.

Examples

```
st <- ic_storage_local(file.path(tempdir(), "a-repo"))
st
ic_storage_memory()
ic_storage_http("https://data.source.coop/ausantarctic/ausantarctic-scratch/oisst/oisst-v2.1-daily/icechunk")
## Not run:
# cloud object stores (the "cloud" build tier)
ic_storage_s3("my-bucket", prefix = "repo", region = "us-west-2",
              credentials = ic_cred_s3("from_env"))
ic_storage_s3("bucket", prefix = "repo", endpoint_url = "https://projects.pawsey.org.au",
              anonymous = TRUE)
ic_storage_gcs("my-bucket", prefix = "repo", anonymous = TRUE)
ic_storage_azure("account", "container", prefix = "repo")

## End(Not run)
```

ic_store *Zarr key access*

Description

Raw bytes by Zarr key (`"zarr.json"`, `"temp/zarr.json"`, `"temp/c/0/0/0"`). Decoding is a Zarr reader's job (pizzarr, GDAL).

Usage

```
ic_get(session, key, offset = NULL, length = NULL)

ic_get_many(session, keys)

ic_set(session, key, value)

ic_exists(session, key)

ic_delete(session, key)

ic_list(session, prefix = "", recursive = TRUE)
```

Arguments

`session`	an `icechunk_session`.
`key`, `keys`	Zarr keys.
`offset`, `length`	optional byte range.
`value`	a raw vector.
`prefix`	key prefix; must be a group or array path (icechunk refuses partial prefixes such as `"x/c"`); `recursive = FALSE` lists one level.
`recursive`	list all keys below `prefix`.

Value

`ic_get()` a raw vector or `NULL` if absent (an absent chunk means fill value); `ic_get_many()` a list of those.

Examples

```
ex <- ic_example_repo()
s <- ic_session_readonly(ic_repo_open(ex$path, authorize = ex$authorize), branch = "main")
ic_list(s, "small")
ic_exists(s, "small/zarr.json")
jsonlite::fromJSON(rawToChar(ic_get(s, "small/zarr.json")))$shape

# a chunk's bytes: the virtual ones come from the source file
```

```
x <- ic_get(s, "from_bin/c/1/0")
head(readBin(x, "double", 3072, size = 4, endian = "little"))
# a byte range of a chunk
readBin(ic_get(s, "from_nc/c/0/0", offset = 0, length = 8), "double", 2, size = 4)
lengths(ic_get_many(s, c("small/c/0/0", "small/c/1/1", "small/c/9/9")))

# writes go to a writable session
w <- ic_session_writable(ic_repo_open(ex$path), "main")
ic_set(w, "small/c/0/0", writeBin(as.double(101:106), raw(), size = 8, endian = "little"))
ic_delete(w, "small/c/1/1")
ic_status(w)
```

`ic_to_byterefs` *Project an Icechunk session to a byterefs store*

Description

The inverse of `ic_write_byterefs()`, and the "chunk references as a table" view of any Icechunk snapshot. Each array becomes an `arrays` row (dtype, fill, codec and attributes from its `zarr.json`); virtual chunks become refs rows with their source path, offset and size; inline and native chunks become rows with a `data` blob (native chunk bytes are fetched only if `native = TRUE`). Distinct locations become `sources`, with `etag` / `mtime` from the chunk checksums.

Usage

```
ic_to_byterefs(session, dsn = NULL, native = TRUE)
```

Arguments

`session`	an `icechunk_session`.
`dsn`	optional path of a SQLite file to write (requires DBI and RSQLite); it must not exist.
`native`	fetch native chunk bytes into `data` (else those rows have `present = 1` and no location; a byterefs reader cannot use them).

Details

What Icechunk does not hold is reconstructed or left empty: source `size` is `NA`, and `placements` are synthesised so the global refs view is the identity (one placement per source, covering its chunks). If the repository was written by `ic_write_byterefs()`, the root group's `byterefs` entry restores the `meta`, the remap rules and the verbatim source paths.

Value

a `byterefs` object (invisibly if `dsn` is given).

Examples

```
ex <- ic_example_repo()
s <- ic_session_readonly(ic_repo_open(ex$path, authorize = ex$authorize), branch = "main")
br <- ic_to_byterefs(s)
br$arrays[, c("array", "kind", "storage", "shape", "chunks")]
br$refs("from_bin")[, c("dim_0", "dim_1", "path", "offset", "size")]
br$sources
```

`ic_write_byterefs` *Write a byterefs store into an Icechunk session*

Description

Every present chunk becomes a virtual reference (location, offset, length) to the source bytes, with the source's `etag` (or `mtime`) as the checksum, so Icechunk refuses chunks whose source has changed. (Icechunk checks "unmodified since"; for `file://` locations the recorded time is `mtime` + 1, because local filesystems keep sub-second mtimes and byterefs records whole seconds.) Chunks with an inline `data` blob are written as native chunks, absent chunks are left unwritten (they read as the fill value), and coordinate arrays (`affine`, `inline`) are materialised as single native chunks. The root group's attributes hold `meta`, plus a `byterefs` entry with the remap rules and target, so `ic_to_byterefs()` can recover the verbatim source paths.

Usage

```
ic_write_byterefs(
  session,
  br,
  target = NULL,
  checksums = TRUE,
  validate = TRUE
)
```

Arguments

`session`	a writable `icechunk_session`.
`br`	a byterefs store (see `ic_byterefs_read()`).
`target`	remap target (see `ic_byterefs_containers()`).
`checksums`	record `etag` / `mtime` checksums on virtual refs.
`validate`	check every location falls within a declared container.

Details

Arrays become Zarr v3: dtype and byte order from `dtype`, `bytes` then `numcodecs.shuffle` then `zstd` / `numcodecs.zlib` / `gzip` from `codec`, with `imagecodecs_delta` (TIFF predictor 2) ahead of `bytes`.

Value

a data frame, one row per array: `array`, `refs` (virtual), `native` (inline or coordinate chunks), `absent`.

Examples

```
ex <- ic_example_repo()
repo <- ic_repo_open(ex$path, authorize = ex$authorize)
br <- ic_to_byterefs(ic_session_readonly(repo, branch = "main"))

# edit the table, then write it into a session of another repository
i <- which(br$arrays$array == "from_bin")
br$arrays$attrs[[i]] <- list(units = "K")
dst <- ic_repo_create(tempfile("dst-"), containers = ic_byterefs_containers(br),
                      authorize = ex$authorize)
w <- ic_session_writable(dst, "main")
ic_write_byterefs(w, br)
ic_commit(w, "from byterefs")
jsonlite::fromJSON(rawToChar(ic_get(w, "from_bin/zarr.json")))$attributes
```

Icechunk *Icechunk DBI driver*

Description

An Icechunk repository seen through DBI. Tables are the byterefs projection of the snapshot the connection is pinned to (see `ic_to_byterefs()`): `meta`, `arrays`, `sources` and one `refs_<array>` per data array, plus Icechunk's own `branches`, `tags`, `ancestry` and `nodes`. SQL (`DBI::dbGetQuery()`) runs on an in-memory SQLite copy of the tables a statement names: an in-memory SQLite copy (RSQLite), or with `sql = "duckdb"` a DuckDB registration that does not copy.

Usage

```
Icechunk()

## S4 method for signature 'IcechunkDriver'
dbConnect(
  drv,
  storage,
  branch = NULL,
  tag = NULL,
  snapshot = NULL,
  authorize = NULL,
  containers = NULL,
  create = FALSE,
  target = NULL,
  sql = c("sqlite", "duckdb"),
  pushdown = TRUE,
```

```
  arrow = NA,
  ...
)

## S4 method for signature 'IcechunkConnection'
dbListTables(conn, ...)

## S4 method for signature 'IcechunkConnection,character'
dbExistsTable(conn, name, ...)

## S4 method for signature 'IcechunkConnection,character'
dbReadTable(conn, name, ..., chunks = NULL)

## S4 method for signature 'IcechunkConnection,character'
dbListFields(conn, name, ...)

## S4 method for signature 'IcechunkConnection,character'
dbGetQuery(conn, statement, ...)

## S4 method for signature 'IcechunkConnection,character'
dbSendQuery(conn, statement, ...)

## S4 method for signature 'IcechunkResult'
dbFetch(res, n = -1, ...)

## S4 method for signature 'IcechunkResult'
dbClearResult(res, ...)

## S4 method for signature 'IcechunkResult'
dbHasCompleted(res, ...)

## S4 method for signature 'IcechunkResult'
dbGetStatement(res, ...)

## S4 method for signature 'IcechunkResult'
dbIsValid(dbObj, ...)

## S4 method for signature 'IcechunkConnection,character'
dbExecute(conn, statement, ...)

## S4 method for signature 'IcechunkConnection'
dbBegin(conn, ...)

## S4 method for signature 'IcechunkConnection'
dbCommit(
  conn,
  message = "DBI commit",
  metadata = NULL,
```

```
  rebase = FALSE,
  solver = c("detect", "basic"),
  on_chunk = c("ours", "theirs", "fail"),
  tries = 5L,
  ...
)

## S4 method for signature 'IcechunkConnection'
dbRollback(conn, ...)

## S4 method for signature 'IcechunkConnection'
dbAppendTable(conn, name, value, ..., checksums = TRUE, row.names = NULL)

## S4 method for signature 'IcechunkConnection,character,data.frame'
dbWriteTable(conn, name, value, ..., overwrite = FALSE, append = FALSE)

## S4 method for signature 'IcechunkConnection'
dbDisconnect(conn, ...)

## S4 method for signature 'IcechunkConnection'
dbIsValid(dbObj, ...)

## S4 method for signature 'IcechunkDriver'
dbIsValid(dbObj, ...)

## S4 method for signature 'IcechunkConnection'
dbGetInfo(dbObj, ...)

## S4 method for signature 'IcechunkDriver'
dbGetInfo(dbObj, ...)

## S4 method for signature 'IcechunkConnection'
show(object)

## S4 method for signature 'IcechunkDriver'
show(object)

## S4 method for signature 'IcechunkDriver'
dbDataType(dbObj, obj, ...)
```

Arguments

drv	an IcechunkDriver.
storage	an ic_storage object or a local path.
branch, tag, snapshot	what the connection is pinned to (one of them; default branch "main"). Only a branch connection can write.
authorize, containers	as for ic_repo_open().

create	create the repository if it does not exist.
target	remap target used to resolve path on writes (default: the one recorded by ic_write_byterefs(), if any).
sql	SQL engine for DBI::dbGetQuery(): "sqlite" (default) copies each table a statement names into an in-memory SQLite; "duckdb" registers them with an in-memory DuckDB without copying, faster on large refs tables. In DuckDB offset and array are reserved words, so quote those columns ("offset", "array"); SQL written that way runs on both engines.
pushdown	for a statement with a plain conjunctive WHERE (no OR, NOT, CASE, subquery or set operation) on dim_k (=, <, <=, >, >=, BETWEEN, IN), read only that slab of chunks of each refs_<array> table it names (and, with split manifests, only the manifests covering it) instead of the whole table. In joins, a predicate counts for a table when qualified by its name or alias, and ranges carry across a.dim_i = b.dim_j equalities, so a filter on one side narrows the other; self-joins and NATURAL / USING joins are read whole. The SQL still applies the full predicate, so results do not change.
arrow	with sql = "duckdb", hand refs_<array> tables to DuckDB as Arrow record batches built in Rust (ic_refs_arrow(), native chunks' bytes included) rather than R vectors: less memory, faster first query, same results. NA (default): when nanoarrow and arrow are installed.
...	unused.
conn	an IcechunkConnection.
name	table name.
chunks	for dbReadTable() on refs_<array>: a slab of chunk indices, list(dim_0 = c(lo, hi), ...) (hi exclusive; a single number is one index). Only the manifests covering it are read when the repository splits manifests (see manifest_split in ic_repo_create()).
statement	SQL (read-only: SELECT over the tables it names).
res	an IcechunkResult.
n	rows to fetch (-1 for all).
dbObj	an Icechunk DBI object.
message	commit message.
metadata	commit metadata (named list).
rebase	if another writer moved the branch since DBI::dbBegin(), rebase onto the new head and retry, up to tries times. With solver = "detect" this succeeds only when the two change sets touch different chunks and metadata; solver = "basic" also resolves chunk conflicts, keeping on_chunk ("ours" or "theirs"). An unresolvable conflict is signalled as icechunk_conflict and leaves the transaction open (rebased as far as it got), for DBI::dbRollback() or another try.
solver, on_chunk	as for ic_rebase().
tries	maximum commit attempts with rebase = TRUE.

value	a data frame. For refs_<array>: dim_0.., and either path (+ offset, size) for a virtual ref or data (raw) for an inline chunk; optional present (0 rows are skipped), etag, mtime. For arrays: byterefs arrays rows (JSON text columns), creating each array.
checksums	record etag / mtime checksums on virtual refs. Rows without them (columns absent or NA) take the checksum of the same path in the sources table, so refs to a known source are checked like the rest; a path new to the repository gets none unless supplied.
row.names	ignored (required by the DBI generic).
overwrite, append	only appending is supported.
object	an Icechunk DBI object.
obj	an R object.

Details

Writes go through a writable session on the connection's branch: DBI::dbBegin() opens it, DBI::dbAppendTable() on refs_<array> sets virtual (or inline, from data) chunk refs, on arrays creates arrays, DBI::dbCommit() commits (pass message), DBI::dbRollback() discards. Without dbBegin() each append commits on its own.

Value

Icechunk(): an IcechunkDriver.

Examples

```
if (requireNamespace("RSQLite", quietly = TRUE)) {
  library(DBI)
  ex <- ic_example_repo()
  con <- dbConnect(Icechunk(), ex$path, authorize = ex$authorize)
  print(con)
  print(dbListTables(con))
  print(dbGetQuery(con, 'SELECT "array", kind, shape, chunks, dtype FROM arrays'))
  # chunk references: one row per chunk; WHERE on dim_k reads only that slab
 print(dbGetQuery(con, 'SELECT dim_0, path, "offset", size FROM refs_from_bin WHERE dim_0 >= 2'))
  print(dbReadTable(con, "ancestry")[, c("snapshot_id", "message")])

  # write: append references in a transaction, which becomes one commit
  dbBegin(con)
  dbAppendTable(con, "refs_from_bin", data.frame(
   dim_0 = 3L, dim_1 = 0L, path = file.path(ex$source, "grid.bin"), offset = 0, size = 3072))
  dbCommit(con, "point the last block at the first")
  print(dbGetQuery(con, 'SELECT dim_0, "offset" FROM refs_from_bin'))

  # an earlier state, read-only
  old <- dbConnect(Icechunk(), ex$path, tag = "v1", authorize = ex$authorize)
  print(dbListTables(old))
  dbDisconnect(old); dbDisconnect(con)
}
```

Index

dbAppendTable,IcechunkConnection-method (Icechunk), 27
dbBegin,IcechunkConnection-method (Icechunk), 27
dbClearResult,IcechunkResult-method (Icechunk), 27
dbCommit,IcechunkConnection-method (Icechunk), 27
dbConnect,IcechunkDriver-method (Icechunk), 27
dbDataType,IcechunkDriver-method (Icechunk), 27
dbDisconnect,IcechunkConnection-method (Icechunk), 27
dbExecute,IcechunkConnection,character-method (Icechunk), 27
dbExistsTable,IcechunkConnection,character-method (Icechunk), 27
dbFetch,IcechunkResult-method (Icechunk), 27
dbGetInfo,IcechunkConnection-method (Icechunk), 27
dbGetInfo,IcechunkDriver-method (Icechunk), 27
dbGetQuery,IcechunkConnection,character-method (Icechunk), 27
dbGetStatement,IcechunkResult-method (Icechunk), 27
dbHasCompleted,IcechunkResult-method (Icechunk), 27
DBI::dbAppendTable(), *31*
DBI::dbBegin(), *30, 31*
DBI::dbCommit(), *31*
DBI::dbConnect(), *11*
DBI::dbGetQuery(), *27*, *30*
DBI::dbRollback(), *30, 31*
dbIsValid,IcechunkConnection-method (Icechunk), 27
dbIsValid,IcechunkDriver-method (Icechunk), 27
dbIsValid,IcechunkResult-method (Icechunk), 27
dbListFields,IcechunkConnection,character-method (Icechunk), 27
dbListTables,IcechunkConnection-method (Icechunk), 27
dbReadTable,IcechunkConnection,character-method (Icechunk), 27
dbRollback,IcechunkConnection-method (Icechunk), 27
dbSendQuery,IcechunkConnection,character-method (Icechunk), 27
dbWriteTable,IcechunkConnection,character,data.frame-metho (Icechunk), 27

ic_amend *(*ic_commit*)*, 6
ic_ancestry *(*ic_refs*)*, 13
ic_branch_create *(*ic_refs*)*, 13
ic_branch_delete *(*ic_refs*)*, 13
ic_branch_reset *(*ic_refs*)*, 13
ic_branches *(*ic_refs*)*, 13
ic_byterefs_compare, 2
ic_byterefs_containers, 3
ic_byterefs_containers(), *16*, *26*
ic_byterefs_read, 4
ic_byterefs_read(), *3*, *16*, *26*
ic_chunk_refs, 5
ic_chunk_refs(), *11*
ic_commit, 6
ic_commit(), *12*, *19, 20*
ic_compiled_features, 7
ic_container *(*ic_credentials*)*, 8
ic_container(), *3*, *17*
ic_cred_azure *(*ic_credentials*)*, 8
ic_cred_gcs *(*ic_credentials*)*, 8
ic_cred_http *(*ic_credentials*)*, 8
ic_cred_local *(*ic_credentials*)*, 8
ic_cred_s3 *(*ic_credentials*)*, 8
ic_cred_s3(), *18, 19*, *23*

ic_credentials, 8
ic_delete (ic_store), 24
ic_diff (ic_refs), 13
ic_discard (ic_commit), 6
ic_example_oisst, 9
ic_example_repo, 10
ic_example_repo(), *10*
ic_exists (ic_store), 24
ic_flush (ic_commit), 6
ic_get (ic_store), 24
ic_get(), *12*
ic_get_many (ic_store), 24
ic_icechunk_version
(ic_compiled_features), 7
ic_list (ic_store), 24
ic_lookup (ic_refs), 13
ic_manifest_concurrency, 11
ic_manifest_concurrency(), *14*
ic_nodes (ic_chunk_refs), 5
ic_pizzarr_store, 12
ic_rebase (ic_commit), 6
ic_rebase(), *30*
ic_refs, 13
ic_refs_arrow, 14
ic_refs_arrow(), *11*, *30*
ic_repo_config (ic_repo_open), 17
ic_repo_containers, 15
ic_repo_containers(), *18*
ic_repo_create (ic_repo_open), 17
ic_repo_create(), *16*, *30*
ic_repo_exists (ic_repo_open), 17
ic_repo_from_byterefs, 16
ic_repo_open, 17
ic_repo_open(), *9*, *11*, *16*, *29*
ic_repo_save_config (ic_repo_open), 17
ic_repo_save_config(), *17, 18*
ic_session_bytes, 18
ic_session_fork (ic_session_bytes), 18
ic_session_fork(), *19*
ic_session_from_bytes
(ic_session_bytes), 18
ic_session_info (ic_session_readonly),
20
ic_session_merge (ic_session_bytes), 18
ic_session_readonly, 20
ic_session_writable
(ic_session_readonly), 20
ic_set (ic_store), 24
ic_set_virtual_ref, 21
ic_set_virtual_refs
(ic_set_virtual_ref), 21
ic_status (ic_commit), 6
ic_storage, *16, 17*, 22, *29*
ic_storage_azure (ic_storage), 22
ic_storage_gcs (ic_storage), 22
ic_storage_http (ic_storage), 22
ic_storage_local (ic_storage), 22
ic_storage_memory (ic_storage), 22
ic_storage_s3 (ic_storage), 22
ic_storage_s3(), *19*
ic_store, 24
ic_tag_create (ic_refs), 13
ic_tag_delete (ic_refs), 13
ic_tags (ic_refs), 13
ic_to_byterefs, 25
ic_to_byterefs(), *11*, *14*, *18*, *26, 27*
ic_virtual_locations (ic_chunk_refs), 5
ic_write_byterefs, 26
ic_write_byterefs(), *14*, *16*, *25*, *30*
Icechunk, 27
IcechunkConnection-class (Icechunk), 27
IcechunkDriver-class (Icechunk), 27
IcechunkResult-class (Icechunk), 27

show,IcechunkConnection-method
(Icechunk), 27
show,IcechunkDriver-method (Icechunk),
27