Package 'getaca'

September 1, 2026

Title Reproducible External Data Dependencies

Version 0.1.6

Language en-GB

Description Declares, retrieves, verifies, tracks and actively manages external data dependencies too large or too fast-moving to ship inside a package. Resources are identified by package, name and version, pinned to a Secure Hash Algorithm (SHA-256) checksum, and resolved through an explicit policy so that the same installed package always resolves the same bytes. A registry served from a remote host may be signed with Ed25519 and verified against a key the declaring package ships, so the declaration and the key that vouches for it arrive by different routes. Hashing follows National Institute of Standards and Technology (2015) ``Secure Hash Standard" <doi:10.6028/NIST.FIPS.180-4>; signing follows Bernstein, Duif, Lange, Schwabe and Yang (2012) ``High-Speed High-Security Signatures" <doi:10.1007/s13389-012-0027-1> and Josefsson and Liusvaara (2017) ``Edwards-Curve Digital Signature Algorithm (EdDSA)" <doi:10.17487/RFC8032>. Designed for reproducible offline use and graceful behaviour during package checks.

License MIT + file LICENSE

URL https://gillescolling.com/getaca/, https://github.com/gcol33/getaca

BugReports https://github.com/gcol33/getaca/issues

Encoding UTF-8

Depends R (>= 4.0.0)

Imports curl (>= 5.0.0), stats, tools, utils

Suggests jsonlite, knitr, rmarkdown, testthat (>= 3.0.0), withr, yaml

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Gilles Colling [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-3070-6066>),
Daniel J. Bernstein [ctb] (TweetNaCl, from which src/ed25519.c derives),
Bernard van Gastel [ctb] (TweetNaCl, from which src/ed25519.c derives),
Wesley Janssen [ctb] (TweetNaCl, from which src/ed25519.c derives),
Tanja Lange [ctb] (TweetNaCl, from which src/ed25519.c derives),
Peter Schwabe [ctb] (TweetNaCl, from which src/ed25519.c derives),
Sjaak Smetsers [ctb] (TweetNaCl, from which src/ed25519.c derives)

Maintainer Gilles Colling <gilles.colling051@gmail.com>

Repository CRAN

Date/Publication 2026-09-01 14:00:02 UTC

Contents

`as_registry` *Convert an authoring format into a registry*

Description

Accepts the list shape a YAML or JSON registry parses into. Requires the `yaml` or `jsonlite` package only when reading those formats; neither is a hard dependency.

Usage

```
as_registry(x, package, ...)
```

Arguments

`x`	A list, or a path to a `.yml`, `.yaml` or `.json` file.
`package`	Declaring package name.
`...`	Passed to `registry()`.

Details

Multi-part records are expressible, since a `part()` is data. A `combiner()` is not, so a record combined by anything other than the default has to be declared with `resource()` in R, where the function it names exists.

Value

A `getaca_registry`.

`combiner` *Declare how parts are combined*

Description

A combiner turns the verified `part()`s of a resource, in declaration order, into the single artefact the resource names. Concatenation is the default and needs no declaration; a combiner is what a delta format calls for, since applying a patch is knowledge about a file format and getaca has none.

Usage

```
combiner(id, fn)
```

Arguments

`id`	Short stable identifier for this transformation, for example `"bsdiff"`.
`fn`	A function `(parts, output)`, where `parts` is a character vector of verified local paths in declaration order and `output` is the file to write. The return value is ignored.

Details

The result is held to the resource's own SHA-256 like any other bytes, so a combiner cannot produce something other than what the declaration promises. That is also why the manifest records a combiner by `id` alone: the checksum says the result is right, and the identifier only says what to run.

Value

An object of class `getaca_combiner`.

See Also

`part()`

Examples

```
combiner("bsdiff", function(parts, output) {
  # apply parts[-1] to parts[1], writing the result to output
  file.copy(parts[1], output)
})
```

`getaca` *Get a declared external resource*

Description

The single retrieval verb. Packages declare resources; getaca retrieves them. Returns an ordinary local path, always: getaca never reads data and knows nothing about file formats.

Usage

```
getaca(
  name,
  package = NULL,
  registry = NULL,
  version = NULL,
  policy = NULL,
  verify = FALSE,
  processed = TRUE,
  quiet = FALSE
)
```

Arguments

name	Resource name as declared by package.
package	Declaring package. The resource identity is package / name / version, so two packages declaring the same name never collide.
registry	A registry() object, for standalone use without a declaring package.
version	Explicit version, bypassing channel resolution. Use this to hold an analysis to one release.
policy	Resolution policy for this call. Defaults to getaca_policy().
verify	Force a full re-hash of the cached copy before returning it. Ordinary access performs a cheap size check and re-hashes on a schedule.
processed	Apply the declared processor(), when there is one, and return the processed path. FALSE returns the raw artefact.
quiet	Report nothing for this call, whatever getaca_progress() is set to.

Details

The returned path is guaranteed to be a complete file, verified against the declared SHA-256, at the requested version, in a cache slot getaca owns and tracks.

Value

A local file or directory path.

See Also

getaca_available() to test without downloading, getaca_info() for provenance, getaca_clean() for cache management.

Examples

```
# Resources are declared by the packages that need them. This one is a zip
# its host would not take whole, uploaded in two pieces and unpacked on
# arrival:
unzipper <- processor("unzip", function(input, output_dir) {
  utils::unzip(input, exdir = output_dir)
  output_dir
})

atlas <- resource("atlas", "1.0",
                  sha256 = strrep("c", 64),
                  size = 1572864,
                  file = "atlas.zip",
                  license = "CC-BY-4.0",
                  parts = list(
                    part("https://example.org/atlas-1.0.zip.001",
                         sha256 = strrep("a", 64), size = 1048576),
                    part("https://example.org/atlas-1.0.zip.002",
                         sha256 = strrep("b", 64), size = 524288)
```

```
                  ),
                  processor = unzipper)
atlas

reg <- registry("demo", list(atlas))

## Not run:
# Each piece is fetched and verified on its own, the two are concatenated,
# and the zip is held to the resource's own sha256 before the processor
# sees it. The returned path is the unpacked directory:
getaca("atlas", registry = reg)

# The raw zip, without unpacking:
getaca("atlas", registry = reg, processed = FALSE)

## End(Not run)
```

`getaca-auth` *Declaring a credential without holding one*

Description

Some versioned scientific files are served only to a registered account. A declaration can say which credential a host requires without ever carrying one: `bearer()` and `basic()` name environment variables, and `auth_host()` binds a scheme to the host it applies to.

Usage

```
bearer(variable)

basic(user, password)

auth_host(host, scheme, register = NULL)
```

Arguments

`variable`	Name of the environment variable holding the token. The variable name is the declaration; its value never enters the registry.
`user`	Name of the environment variable holding the user name.
`password`	Name of the environment variable holding the password.
`host`	Host the credential applies to, as it appears in the URL, for example `"data.example.org"`. Matched exactly, without wildcards.
`scheme`	A `bearer()` or `basic()` declaration.
`register`	Optional URL where a user obtains a credential. Reported when one is missing or refused, and part of the manifest so that a signed registry covers it.

Details

A credential belongs to a host rather than to a file. One record may list a mirror behind a token beside a public one, and several records routinely share a credential, so the declaration sits on the `registry()` and is matched by host. Parts are matched by the same rule, since a part's URLs are URLs.

Value

`bearer()` and `basic()` return a `getaca_auth_scheme`; `auth_host()` returns a `getaca_auth_host`.

What getaca will not do

The credential is read from the environment at the moment of the request and is never stored, never written to the cache, never recorded in provenance and never printed. It is sent as an `Authorization` header and to nothing but the declared host: libcurl withholds that header from a redirect to a different host, which is what a query-string token could not offer and is why one is not accepted here.

Hosts are matched exactly and there is no wildcard, so a declaration can never widen the set of hosts that receive a credential.

Examples

```
bearer("EXAMPLE_TOKEN")
basic("EXAMPLE_USER", "EXAMPLE_PASSWORD")
auth_host("data.example.org", bearer("EXAMPLE_TOKEN"),
          register = "https://data.example.org/register")
```

`getaca-checks` *Behave during checks, examples and tests*

Description

CRAN policy requires that a package using Internet resources "fail gracefully with an informative message if the resource is not available or has changed (and not give a check warning nor error)". These three helpers answer three different questions, so a package can satisfy that in every context without writing its own availability logic.

Usage

```
getaca_available(
  name,
  package = NULL,
  registry = NULL,
  version = NULL,
  processed = TRUE
)
```

```
getaca_optional(
  name,
  package = NULL,
  registry = NULL,
  version = NULL,
  processed = TRUE,
  quiet = FALSE
)

getaca_skip_if_unavailable(
  name,
  package = NULL,
  registry = NULL,
  version = NULL,
  processed = TRUE
)
```

Arguments

name	Resource name as declared by package.
package	Declaring package. The resource identity is package / name / version, so two packages declaring the same name never collide.
registry	A registry() object, for standalone use without a declaring package.
version	Explicit version, bypassing channel resolution. Use this to hold an analysis to one release.
processed	Apply the declared processor(), when there is one, and return the processed path. FALSE returns the raw artefact.
quiet	Report nothing for this call, whatever getaca_progress() is set to.

Details

getaca_available() Is this resource usable right now, without touching the network? Returns a logical.

getaca_optional() Give me the path if you have it. Returns NULL with a message otherwise, and never errors. For examples and vignettes.

getaca_skip_if_unavailable() Skip this test, naming the missing dependency and how to prefetch it. For testthat.

Value

getaca_available() returns TRUE when the resource is cached and passes its cheap integrity check.

getaca_optional() returns a path, or NULL when the resource is unavailable.

getaca_skip_if_unavailable() returns NULL invisibly, or signals a testthat skip.

Examples

```
getaca_available("nothing-here", package = "getaca")
```

`getaca-gc` *Active cache management*

Description

CRAN policy permits a package to use `tools::R_user_dir()` "provided that by default sizes are kept as small as possible and the contents are actively managed (including removing outdated material)". Actively managed means the package has a retention policy, not that it ships a function users might discover. getaca therefore collects conservatively after every successful retrieval, and `getaca_clean()` exists for the deliberate case.

Usage

```
getaca_clean(
  name = NULL,
  package = NULL,
  what = c("broken", "temp", "superseded", "lru", "unreferenced"),
  dry_run = FALSE
)
```

Arguments

`name`	Restrict to one resource name.
`package`	Restrict to one declaring package. `NULL` means all.
`what`	Which sweeps to run. Any of `"broken"`, `"temp"`, `"superseded"`, `"lru"`, `"unreferenced"`.
`dry_run`	Report what would be removed without removing it.

Details

Removal order, cheapest and safest first:

1. broken and incomplete material
2. abandoned temporary transfers
3. superseded unpinned versions, once past the retention period
4. least recently used unpinned resources, only when over the size ceiling
5. bytes in the store that no declaration references any more

"Superseded" and "not recently used" are different states and age on different clocks, so an expensive resource is not deleted merely for being old.

A version slot holds a name for bytes the store owns, so the sweeps above remove names. Bytes go once the last name for them does, which is why the store sweep runs last.

Never removed: pinned entries, the version the bundled registry currently names, and anything under an active lock.

Value

A data frame of affected entries, invisibly when acting.

Examples

```
getaca_clean(dry_run = TRUE)
```

`getaca-progress` *How a transfer reports itself*

Description

getaca drives its own transfer loop, so what a download reports is a decision the package makes rather than one libcurl makes for it. A reporter receives an event for every transfer that begins, for the bytes that arrive while it runs, and for how it ended, and renders them however it likes.

Details

The events carry what getaca knows and a transfer library cannot: the resource identity, which part of a series is moving, the size the registry declares for it, and how much of it was already on disk when the attempt resumed. That is why the declared total is available before the first byte arrives, and why a series reports `(part 2 of 3)` rather than three unrelated downloads.

Events

Each event is a list with a `type` and the fields for that type. A handler switches on `type` and ignores what it does not use, so a later release adding an event type leaves an existing reporter working.

`"begin"` A transfer attempt starts. `id` is the resource, which `format()` renders including the part label where there is one. `url` is the mirror. `total` is the declared size in bytes, or `NA` when the declaration gives none. `offset` is what was already on disk, which is non-zero when an interrupted transfer resumes.

`"bytes"` Bytes have arrived. `bytes` is the cumulative total for this attempt including `offset`, so it can be compared against `total` directly. Fires often; a reporter that draws is expected to throttle.

`"end"` The attempt finished. `status` is `"ok"` or `"failed"`, `reason` is the failure reason or `NA`, and `bytes` is what arrived.

A reporter never affects the outcome of a retrieval. An error raised inside one is caught, reported once as a warning, and the reporter is switched off for the rest of the session's call rather than being allowed to fail a download that is otherwise fine.

See Also

`getaca_progress()` to choose one, `reporter()` to write one.

getaca_cache_dir *Where getaca stores things*

Description

Uses `tools::R_user_dir("getaca", "cache")` as CRAN policy requires, unless overridden by the `getaca.cache` option or the `GETACA_CACHE` environment variable. Setting `GETACA_CACHE` is the supported way to point a CI job or a check run at a pre-seeded cache.

Usage

```
getaca_cache_dir()
```

Value

A directory path. Not created as a side effect of asking.

Examples

```
getaca_cache_dir()
```

getaca_catalogue *What is declared, and what is cached*

Description

Reports both halves. Declared resources appear whether or not they have ever been downloaded, so "what does this package need, and what do I already have" is one table rather than two. Cached copies of versions that are no longer declared appear as well, since those are what `getaca_clean()` reclaims.

Usage

```
getaca_catalogue(package = NULL, registry = NULL)
```

Arguments

`package`	Restrict to one declaring package. `NULL` reports every installed package that ships a registry, together with any package holding cached resources.
`registry`	A `registry()` object, for standalone use without an installed declaring package.

Value

A data frame, one row per resource version. `parts` is how many pieces the artefact is composed from, and `0` where it is served whole, so what a version update costs to fetch is visible. `current` marks the version a bare request for that name resolves to, so a channel head is visible rather than implied. `declared` is `TRUE` when the registry in force names that version, `FALSE` when it does not, and `NA` when no registry could be read for the package; `current` is `NA` in that same case. `cached` says whether a local copy is recorded; the provenance columns are `NA` for declared resources that are not cached. `doi` is what the artefact is cited as, where the declaration states one. `link` says how the slot reaches its bytes, so two packages sharing one copy in the store are visible as such.

Examples

```
reg <- registry("demo", list(
  resource("example", "1.0",
           urls = "https://example.org/example-1.0.csv",
           sha256 = strrep("c", 64))
))
getaca_catalogue(registry = reg)
```

`getaca_credentials` *Which credentials a package expects*

Description

Reports the environment variables a package's declaration reads, and whether each is set in this session. Answers "do I have what I need" before a fetch rather than during one, without touching the network.

Usage

```
getaca_credentials(package = NULL, registry = NULL)
```

Arguments

`package`	Declaring package. Ignored when `registry` is supplied.
`registry`	A `registry()` object, for standalone use.

Details

Values are never read or shown. `set` says only that the variable holds something.

Value

A data frame with one row per declared variable: `package`, `host`, `scheme`, `variable`, `set` and `register`. Empty when nothing is declared.

See Also

getaca-auth

Examples

```
reg <- registry("demo",
  auth = list(auth_host("data.example.org", bearer("EXAMPLE_TOKEN"))),
  resources = list(
    resource("example", "1.0",
             urls = "https://data.example.org/example-1.0.csv",
             sha256 = strrep("c", 64))
  ))
getaca_credentials(registry = reg)
```

---

getaca_info *Provenance for a resource*

Description

Answers, for a cached resource: which package declared it, which registry state and which policy resolved it, the exact version, declared and observed checksums, which mirror served it, when it was fetched and when it was last fully verified, its license and DOI, any processor applied, which getaca retrieved it, and the local path. Suitable for a reproducibility appendix or a bug report.

Usage

```
getaca_info(
  name,
  package = NULL,
  registry = NULL,
  version = NULL,
  processed = TRUE
)
```

Arguments

name	Resource name as declared by package.
package	Declaring package. The resource identity is package / name / version, so two packages declaring the same name never collide.
registry	A registry() object, for standalone use without a declaring package.
version	Explicit version, bypassing channel resolution. Use this to hold an analysis to one release.
processed	Apply the declared processor(), when there is one, and return the processed path. FALSE returns the raw artefact.

Details

The registry state appears as a `registry_digest()`, so the declaration that resolved the resource can be identified exactly rather than by a number someone kept in step by hand.

Value

A `getaca_entry`, or `NULL` when the resource is not cached.

`getaca_keep` *Keep a resource from being collected*

Description

Keep a resource from being collected

Usage

```
getaca_keep(
  name,
  package = NULL,
  registry = NULL,
  version = NULL,
  processed = TRUE,
  pinned = TRUE
)
```

Arguments

`name`	Resource name as declared by `package`.
`package`	Declaring package. The resource identity is `package` / `name` / `version`, so two packages declaring the same name never collide.
`registry`	A `registry()` object, for standalone use without a declaring package.
`version`	Explicit version, bypassing channel resolution. Use this to hold an analysis to one release.
`processed`	Apply the declared `processor()`, when there is one, and return the processed path. `FALSE` returns the raw artefact.
`pinned`	Set `FALSE` to release the pin.

Value

The updated entry, invisibly.

getaca_pin *Freeze current resolution into a pin file*

Description

Records, for each named package, the registry state currently in effect. Under the "pinned" policy those records are what resolution uses, so an analysis keeps resolving the versions it was written against.

Usage

```
getaca_pin(packages, path = pin_file())
```

Arguments

packages	Character vector of package names.
path	Where to write the pin file.

Value

path, invisibly.

getaca_policy *Resolution policy and settings*

Description

getaca_policy() reports or sets the resolution policy for the current session. The policy decides which registry state a name resolves through, and is recorded in provenance so a result can always be traced back to it.

Usage

```
getaca_policy(policy = NULL)
```

Arguments

policy	One of "bundled", "current", "pinned", "offline", or NULL to query without setting.

Value

When querying, the policy in effect. When setting, the previous value of the getaca.policy option invisibly, NULL if it was unset.

Policies

`"bundled"` Always use the registry shipped with the declaring package. The same installed package resolves the same bytes forever. This is the default.

`"current"` Consult the author-controlled remote registry, falling back to bundled when it cannot be reached. Lets an author repair a dead mirror or publish a new version without a CRAN release.

`"pinned"` Resolve through a frozen local snapshot, so an analysis keeps resolving what it resolved on the day it was written.

`"offline"` Never touch the network. Cached and bundled information only.

During `R CMD check` resolution always collapses to `"offline"`, whatever is set here.

Setting the policy sets the `getaca.policy` option and nothing else, and returns what that option held before, so a caller that has to change it can put it back:

```
old <- getaca_policy("offline")
on.exit(options(getaca.policy = old), add = TRUE)
```

Examples

```
getaca_policy()

# Setting it is reversible, because the previous value comes back.
old <- getaca_policy("offline")
getaca_policy()
options(getaca.policy = old)
```

`getaca_prefetch` *Warm the cache ahead of time*

Description

Downloads and verifies without returning anything, so a connected machine can prepare a cache that a check run, a CI job or an offline session will then find already populated.

Usage

```
getaca_prefetch(names = NULL, package = NULL, registry = NULL, quiet = FALSE)
```

Arguments

`names`	Resource names. `NULL` prefetches everything the package declares.
`package`	Declaring package. The resource identity is `package` / `name` / `version`, so two packages declaring the same name never collide.
`registry`	A `registry()` object, for standalone use without a declaring package.
`quiet`	Report nothing for this call, whatever `getaca_progress()` is set to.

Value

A character vector of paths, invisibly.

`getaca_progress` *Choose how transfers report progress*

Description

Reports or sets the progress style for the current session. The default, `"auto"`, draws a bar when the session is interactive and reports nothing when it is not, which keeps a log or a CI transcript clean without asking.

Usage

```
getaca_progress(progress = NULL)
```

Arguments

`progress`	A style name, a `reporter()`, or `NULL` to query without setting.

Details

`quiet = TRUE` on an individual `getaca()` call overrides whatever is set here, so one silent call never needs the session changed and put back.

Value

When querying, the reporter in effect. When setting, the previous value of the `getaca.progress` option invisibly, `NULL` if it was unset.

Styles

`"auto"` A bar when interactive, nothing otherwise. The default.

`"bar"` A single line that redraws in place, with the share transferred, the rate and an estimate of what is left. Falls back to bytes and a rate where the registry declares no size.

`"line"` One line when a transfer starts and one when it ends. What a CI log or a `sink()`ed script wants, where a redrawing bar leaves thousands of fragments.

`"none"` Nothing at all.

Setting the style sets the `getaca.progress` option and nothing else, and returns what that option held before, so a caller that has to change it can put it back:

```
old <- getaca_progress("none")
on.exit(options(getaca.progress = old), add = TRUE)
```

See Also

`reporter()` to write your own, and getaca-progress for the events one receives.

Examples

```
getaca_progress()

# A reporter of your own: one line per completed transfer, and nothing
# while it runs.
logger <- reporter("log", function(event) {
  if (identical(event$type, "end") && identical(event$status, "ok")) {
    message(format(event$id), " retrieved (", event$bytes, " bytes)")
  }
})
logger

# Choosing one is reversible, because the previous setting comes back.
old <- getaca_progress(logger)
getaca_progress()
options(getaca.progress = old)
```

---

getaca_refresh *Forget cached registry state*

Description

Registries are read once per session: the one a package ships is cached after the first `system.file()` lookup, and a remote registry is cached after the first fetch. Call this after installing a new version of a declaring package, or to make the `"current"` policy consult the remote again within the same session.

Usage

```
getaca_refresh()
```

Details

Cached *resources* are untouched. This forgets declarations, not data.

Value

`NULL`, invisibly.

Examples

```
getaca_refresh()
```

part *Declare one part of a resource*

Description

A part names bytes that are a piece of an artefact rather than the artefact: a chunk of a file split for a host with an upload limit, or a base release and a delta issued against it. Parts are retrieved and verified individually and then combined, in declaration order, into the file `resource()` names.

Usage

```
part(urls, sha256, size = NA_real_)
```

Arguments

`urls`	Character vector of download locations for this part, tried in order. All must be `https://`.
`sha256`	Lowercase hex SHA-256 of this part as served.
`size`	Expected size in bytes, or `NA`. Used to detect a truncated transfer before hashing.

Details

Each part carries its own checksum and is stored under it, so a base shared by every version of a resource is transferred once and kept once however many versions declare it. Publishing a new version then costs its consumers the delta rather than the whole file.

Parts describe how the bytes arrive. A declaration may re-split an artefact, add mirrors for a piece or drop one, the same way it may repair a mirror list, because what a version means is fixed by the resource's own `sha256` and checked against it after composition.

Value

An object of class `getaca_part`.

See Also

`resource()`, and `combiner()` for parts that are not simply concatenated.

Examples

```
part("https://example.org/backbone-base.bin", sha256 = strrep("c", 64),
     size = 1048576)
```

processor *Declare a post-download processor*

Description

A processor turns one verified path into another path: unpacking an archive, or preparing a package-specific layout. It carries an `id` so the processed result gets its own cache slot and its own provenance, rather than being confused with the raw artefact it came from.

Usage

```
processor(id, fn)
```

Arguments

`id`	Short stable identifier for this transformation, for example `"unzip"` or `"unzip-v2"`. Changing the transformation means changing the id, which invalidates previously processed copies.
`fn`	A function `(input, output_dir)` returning a path inside `output_dir`.

Details

getaca knows nothing about file formats. It never reads data.

Value

An object of class `getaca_processor`.

Examples

```
unzipper <- processor("unzip", function(input, output_dir) {
  utils::unzip(input, exdir = output_dir)
  output_dir
})
```

registry *Declare a package's external resources*

Description

A registry is one package's declaration of what it needs. It carries no download logic: getaca is the single engine, and every package supplies only its own list of resource records.

Usage

```
registry(
  package,
  resources,
  remote = NULL,
  policy = c("bundled", "current", "pinned", "offline"),
  current = NULL,
  keys = NULL,
  auth = NULL
)
```

Arguments

package	Name of the declaring package. Becomes part of every resource identity and scopes the cache, so two packages declaring the same resource name never collide.
resources	A list of `resource()` records, or a single record.
remote	Optional URL of an author-controlled registry file. Consulted only under the `"current"` policy. It may repair or add mirrors and may introduce new versions. It may never change the bytes a published version refers to.
policy	Default resolution policy for this package. One of `"bundled"`, `"current"`, `"pinned"`, `"offline"`. See `getaca_policy()`.
current	Named character vector giving the channel head: the version a bare request for each resource name resolves to, as `c(backbone = "2026-09")`. Required for any name declaring more than one version, and optional for the rest, since a name with one version has only one answer.
keys	Public keys, from `registry_keygen()`, that may sign this package's remote registry. Declaring any of them makes a signature mandatory under the `"current"` policy: an unsigned or unverifiable remote registry is then refused rather than used. The keys trusted are the ones in the registry the *package ships*, which reaches a user by a different route than the remote does, and that is what a signature rests on. See getaca-signing.
auth	Optional list of `auth_host()` declarations, naming the environment variables a host requires before it will serve a resource. Read from the registry the *package ships*, never from a remote one, for the reason `keys` is: a declaration arriving over the network must not be able to say where a credential is sent. See getaca-auth.

Details

Ship the result at `inst/getaca/registry.rds` via `registry_write()`. getaca discovers it with `system.file()`, so no registration call and no load hook are required.

Value

An object of class `getaca_registry`.

Identity

A registry state is identified by `registry_digest()`, derived from the declaration itself, and recorded in the provenance of every resource it resolves. There is no revision number to keep in step: a digest cannot be typed wrong, and two states that differ cannot claim to be the same one. `registry_write()` stamps `created`, which is what orders two states in time, and a bundled registry additionally has the version of the package that ships it.

Channel heads

A registry declares records; a channel points at one of them. When a resource name carries several versions, which of them `getaca("name")` returns is a decision, so the registry states it in `current` rather than leaving it to declaration order. A registry that declares two versions of a name without naming a head is refused, which is what stops a version appended in the wrong place from silently moving every user backwards.

Examples

```
registry(
  package = "yourpkg",
  resources = list(
    resource("reference-data", "2.1",
             urls = "https://example.org/ref-2.1.zip",
             sha256 = strrep("b", 64))
  )
)

# Two versions on offer, one of them the channel head:
registry(
  package = "yourpkg",
  current = c("reference-data" = "2.1"),
  resources = list(
    resource("reference-data", "2.0",
             urls = "https://example.org/ref-2.0.zip",
             sha256 = strrep("a", 64)),
    resource("reference-data", "2.1",
             urls = "https://example.org/ref-2.1.zip",
             sha256 = strrep("b", 64))
  )
)
```

`registry_digest` *Content identity of a registry*

Description

The digest of a registry's `registry_manifest()`. This is what identifies a declaration state: it is derived from the declaration rather than asserted alongside it, so it cannot be typed wrong, cannot go stale, and cannot claim that two different states are the same one. Provenance records it for every retrieved resource, so a cached file can always be traced to the exact declaration that resolved it.

Usage

```
registry_digest(registry)
```

Arguments

registry	A registry() object.

Details

The value is self-describing, as in "sha256:3f9ac2...", so the algorithm can change later without changing the shape of anything that stores one.

A digest says whether two registries are the same. It does not say which is newer; created answers that. It also carries no authenticity: a digest travelling inside the file it describes is rewritten by anyone who rewrites the file. What stops a remote registry from redefining published bytes is the per-resource checksum comparison in resolve_resource(), which names the offending resource rather than reporting that something, somewhere, moved.

Value

A single string: an algorithm name, a colon, and lowercase hex.

See Also

registry_manifest() for the exact bytes hashed.

Examples

```
reg <- registry("demo", list(
  resource("example", "1.0",
           urls = "https://example.org/example-1.0.csv",
           sha256 = strrep("c", 64))
))
registry_digest(reg)
```

---

registry_draft *Draft a registry from where the data is*

Description

Takes locations, returns a registry() with every checksum filled in. What it saves is the part of authoring that cannot be done by hand: a SHA-256 has to be computed from the bytes, which means retrieving them.

Usage

```
registry_draft(
  x,
  package,
  version = NULL,
  source = "auto",
  local = NULL,
  sha256 = NULL,
  keep = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

x	Locations. A character vector, where each element is one location, or a list, where each element is a character vector of mirrors for one resource. Names, if any, name the resources.
package	Declaring package name.
version	Version label for every resource drafted. Optional where the archive supplies one, required for a plain URL.
source	Which handler to use: "auto", or one of "zenodo", "figshare", "dataverse", "url" to override the detection.
local	Paths to copies already on this machine, hashed in place instead of retrieving. One per location: named after the locations they belong to, or one for each in order. A location holding several files cannot take one.
sha256	Checksums to declare as given, retrieving nothing. Named or positional on the same terms as local, and combinable with it, in which case the local copy is hashed and held to the checksum.
keep	Keep the retrieved bytes in the cache, so that a later getaca() call for the drafted resource finds them already there instead of transferring them a second time. Without it a retrieved file is hashed as it arrives and never written down. A location answered from sha256 = alone transfers nothing, so there is nothing for this to keep.
quiet	Suppress transfer progress.
...	Passed to registry(), for remote, policy, keys and auth.

Details

A location is a plain URL, or an identifier for a data archive that holds several files. Which one it is, and which archive, is read off the string, so one call covers both:

```
registry_draft("10.5281/zenodo.17844561", package = "yourpkg")
registry_draft(c(backbone = "https://example.org/backbone.parquet"),
               package = "yourpkg", version = "2026.1")
```

Every file is hashed from its own bytes. Checksums an archive reports are not used: they are md5 at all three archives supported here, and they arrive from the host that serves the bytes, so they say nothing the transfer itself has not already said.

This is an authoring tool. Nothing in the retrieval path calls it, and a drafted registry names ordinary `https://` locations, so an archive is consulted when the registry is written and never when a user fetches.

Value

A `getaca_registry`.

Where the bytes come from

A checksum can only come from the bytes, but they need not be transferred to get one, and where they are they need not be written down.

retrieved The default. The file is fetched once and hashed as it arrives, so nothing is written and a location of any size costs no disk. `keep = TRUE` writes it to the cache instead, where a later `getaca()` call for the drafted resource finds it already there.

`local =` A copy already on this machine, which is the usual case for a file you have just published: it is hashed where it lies and nothing is transferred. The record still names the location, since that is where a user will fetch from.

`sha256 =` A checksum you already hold from somewhere that is not the serving host. Taken as declared, and nothing is retrieved at all.

Giving both `local =` and `sha256 =` for one location hashes the local copy and holds it to the checksum, which is how a published file is confirmed to be the one that was uploaded.

Archives

Zenodo `"10.5281/zenodo.17844561"` or a `https://zenodo.org/records/...` URL. Version defaults to the record id, which Zenodo mints afresh for each version.

figshare `"10.6084/m9.figshare.14763051.v1"` or a `https://figshare.com/articles/...` URL. A DOI without a `.vN` suffix resolves to whatever figshare currently calls latest, and the version it served is what the draft records.

Dataverse A dataset DOI, or a `https://<host>/dataset.xhtml?persistentId=...` URL. Instances are self-hosted under their own DOI prefixes, so a bare DOI is resolved through `doi.org` to find which host to ask. The other two are recognised from the string alone and cost no such lookup.

What to edit afterwards

A draft is a starting point. Resources are named after their files, which is rarely the name you want a user to type, and `description` is left empty. Both are ordinary arguments of `resource()`; edit the call, or edit the returned registry, and write it with `registry_write()`.

See Also

`registry_write()` to ship it, `registry_sign()` to sign it.

Examples

```
## Not run:
reg <- registry_draft("10.5281/zenodo.17844561", package = "yourpkg")
registry_write(reg, "inst/getaca/registry.rds")

# The file you just uploaded, hashed from the copy you uploaded it from.
registry_draft(c(backbone = "https://example.org/backbone.parquet"),
               package = "yourpkg", version = "2026.1",
               local = c(backbone = "~/data/backbone.parquet"))

## End(Not run)
```

`registry_for` *Find the registry a package ships*

Description

Find the registry a package ships

Usage

```
registry_for(package)
```

Arguments

`package`	Package name.

Value

A `getaca_registry`, or `NULL` when the package ships none.

`registry_keygen` *Create a signing key*

Description

Generates an Ed25519 key pair, writes the secret half to `path`, and returns the public half in the form `registry()` takes. The seed comes from the operating system's cryptographic random source, not from R's generator, whose stream is reproducible by design.

Usage

```
registry_keygen(path, seed = NULL)
```

Arguments

path	Where to write the secret key.
seed	Optional 32 raw bytes to derive the key from, for tests that need a fixed key. Omit for a real key.

Details

The secret file is the one thing in this package that must not be published. It is written with owner-only permissions where the platform has them, and belongs outside the package source tree so that no build can sweep it up.

Value

The public key, as `"ed25519:<hex>"`.

See Also

`registry_sign()`

Examples

```
file <- tempfile()
public <- registry_keygen(file)
substr(public, 1, 16)
unlink(file)
```

---

registry_manifest *The canonical form a registry hashes to*

Description

Renders the declaration as a sorted, escaped, line-oriented text form and returns its lines. `registry_digest()` hashes exactly these bytes, so a digest is never a black box: two registries that disagree can be diffed on the text that produced the disagreement.

Usage

```
registry_manifest(registry)
```

Arguments

registry	A `registry()` object.

Details

Hashing the R object directly is not an option. A `resource()` may carry a `processor()`, which holds a closure, and a closure digests differently across machines and builds because its environment, bytecode and source references travel with it. A registry declaring a processor would appear to change identity on every machine. The manifest reduces a processor to its `id`, which is what the declaration actually promises.

Value

A character vector of lines, classed `getaca_manifest`.

Format

The first line names the format version, which moves independently of `schema_version` because the two change for different reasons: a new field in the stored form need not change how existing fields are rendered.

```
getaca-manifest 1
package yourpkg
remote https://yourpkg.example.org/registry.rds
key ed25519:9f8a...
auth data.example.org bearer EXAMPLE_TOKEN
  register https://data.example.org/register
current backbone 2026-09
resource backbone 2026-06
  sha256 3f9ac2...
  size 1048576
  license CC-BY-4.0
  doi 10.5281/zenodo.123
  url https://zenodo.org/record/123/backbone-2026-06.parquet
  url https://mirror.example.org/backbone-2026-06.parquet
  upstream release 2026-06
  processor unzip-v2
resource backbone 2026-09
  sha256 b104e7...
  file backbone.parquet
  part 91cc0d... 1048576
    url https://zenodo.org/record/456/backbone-base.bin
  part 4e77a1... 20481
    url https://zenodo.org/record/456/backbone-2026-09.bin
  combiner concat
```

Resources are sorted by `name@version` in the C locale, so the same declaration renders identically wherever it is read. URLs keep declaration order, which is load-bearing: mirrors are tried in the order given. Parts keep it for a stronger reason, since the order is the one they are combined in. Signing keys are sorted, since which one signs is a fact about the signature rather than about the declaration. Absent and `NA` fields are omitted rather than rendered as empty, since a key that is present carries a value by construction.

Rendering an absent field as nothing is what let signing keys join the manifest without a new format version, and then `file`, `part`, `combiner`, `auth` and `doi` after them. A registry declaring none of them renders exactly the bytes it always did, so every digest recorded before they existed still identifies the state that produced it.

What is left out

`created` and `policy` are not part of the declaration. `created` says when a state was written and `policy` supplies a default; neither changes which bytes a name resolves to, and including either would mean an unchanged registry took on a new identity for writing it out again. `description` is prose, so a typo fix would invalidate a digest already recorded in provenance.

See Also

`registry_digest()`

Examples

```
reg <- registry("demo", list(
  resource("example", "1.0",
           urls = "https://example.org/example-1.0.csv",
           sha256 = strrep("c", 64), license = "CC0-1.0")
))
registry_manifest(reg)
```

---

registry_sign *Sign a registry file*

Description

Signs the declaration in a written registry, producing a detached signature beside it. Sign after `registry_write()`: writing is what stamps `created`, and the signature binds that stamp.

Usage

```
registry_sign(path, key, expires = Sys.time() + SIGNATURE_DAYS * 86400)
```

Arguments

`path`	Path to a registry file written by `registry_write()`.
`key`	Path to a secret key from `registry_keygen()`.
`expires`	When the signature stops being accepted. Re-sign an unchanged registry to extend it. `NA` signs without an expiry, which leaves nothing bounding how long a stale declaration is served.

Details

Publish the `.sig` alongside the registry it describes. getaca fetches it from the registry's own URL with `.sig` appended.

Value

The signature path, invisibly.

See Also

registry_keygen(), registry_verify()

`registry_verify` *Verify a signed registry file*

Description

Checks a registry against the detached signature beside it. Returns `TRUE` or raises `getaca_error_signature` naming what failed.

Usage

```
registry_verify(path, keys = NULL, now = Sys.time())
```

Arguments

`path`	Path to a registry file.
`keys`	Public keys to accept, as returned by `registry_keygen()`. Defaults to the keys the registry declares.
`now`	Time to judge expiry against.

Details

This is the check resolution performs on a fetched remote registry, exposed so an author can run it on their own output before publishing. In resolution the trusted keys come from the bundled registry; here they default to the keys the file itself declares, which answers whether a registry is internally consistent rather than whether it is authentic.

Value

`TRUE`, invisibly.

See Also

registry_sign()

registry_write *Read and write registry files*

Description

The stored form is an R object serialised with `saveRDS()`. YAML and JSON are supported as optional authoring formats only; they never define the internal model and never become a hard dependency.

Usage

```
registry_write(registry, path, created = Sys.time())

registry_read(path)
```

Arguments

`registry`	A `registry()` object.
`path`	File path. For `registry_write()`, the conventional location inside a package source tree is `inst/getaca/registry.rds`.
`created`	Publication time recorded in the file. Pass a fixed value to keep a build byte-reproducible.

Details

Writing stamps `created`, since publishing a state is what dates it. `created` is what orders two states in time, which a content digest cannot do; `registry_digest()` says whether two states are the same, and `created` says which came first. It is deliberately outside the digest, so writing an unchanged registry out again leaves its identity alone.

Value

`registry_write()` returns `path` invisibly. `registry_read()` returns a `getaca_registry`.

reporter *Write a progress reporter*

Description

A reporter turns transfer events into whatever you want a transfer to look like: a bar, a log line, a row in a database, an update to a Shiny session. It carries an `id` so that what is reporting is visible when one is set.

Usage

```
reporter(id, fn)
```

Arguments

id	Short stable identifier, for example "bar" or "shiny".
fn	A function of one argument, the event. Its return value is ignored.

Details

See getaca-progress for the events and their fields.

Value

An object of class getaca_reporter.

See Also

getaca_progress()

Examples

```
# Only the totals, once each transfer is done.
reporter("totals", function(event) {
  if (identical(event$type, "end")) {
    cat(format(event$id), event$status, event$bytes, "\n")
  }
})
```

---

resolve_resource *Resolve a name to an immutable resource record*

Description

Separates the two things that must not be conflated: an immutable resource record, and the mutable channel that maps a logical name onto one. This function walks the channel; everything downstream deals only in records.

Usage

```
resolve_resource(
  name,
  package = NULL,
  registry = NULL,
  policy = NULL,
  version = NULL
)
```

Arguments

name	Resource name.
package	Declaring package. Ignored when registry is supplied.
registry	A registry() object, for standalone use.
policy	Resolution policy, defaulting to getaca_policy().
version	Optional explicit version, bypassing channel resolution.

Value

A list with id, record, policy, source, digest, created and auth. policy is the one actually in force, after the argument, the session setting, the registry default and the check clamp have been resolved. digest identifies the registry state that answered, and created says when that state was published, or NA for a declaration that was built in the session rather than read from a file. auth is the credential declaration, taken from the registry the package ships.

---

resource *Declare an immutable resource record*

Description

A resource record names one concrete artefact: exact bytes, at one or more locations, under one version label. Once published, a record is immutable. If a publisher reissues the same nominal file with different bytes that is an upstream mutation, not a routine update, and getaca reports it as such.

Usage

```
resource(
  name,
  version,
  urls = NULL,
  sha256,
  size = NA_real_,
  license = NA_character_,
  description = NA_character_,
  doi = NULL,
  upstream = NULL,
  processor = NULL,
  parts = NULL,
  combiner = NULL,
  file = NULL
)
```

Arguments

name	Resource name. Must be usable as a directory name.
version	Version label for these exact bytes, for example "2026.1".
urls	Character vector of download locations, tried in order. All must be https://. Give this or parts.
sha256	Lowercase hex SHA-256 of the artefact. For a record with parts, of the artefact they compose rather than of any one of them.
size	Expected size in bytes, or NA. Used to detect truncated transfers before hashing.
license	License identifier for the data, for example "CC-BY-4.0".
description	One-line human description.
doi	Optional DOI for these bytes, for example "10.5281/zenodo.1234567". A https://doi.org/ or doi: prefix is accepted and stripped. This is what the artefact is cited as, and it travels into provenance; it never routes anything, and the locations to fetch from stay in urls.
upstream	Optional named list identifying what these bytes were built from, when the artefact is derived rather than an original release. A prepared database records both its own build identity and the upstream release it was made from, and both travel into provenance.
processor	Optional processor() applied after verification.
parts	Optional list of part() records, in the order they are combined. Give this or urls.
combiner	Optional combiner() turning the parts into the artefact. The default concatenates them, which is what a file split for a host with a size limit needs.
file	Name the artefact is cached under. Defaults to the file name in the first URL, and is required alongside parts, where the URLs name the pieces rather than the result.

Value

An object of class getaca_resource.

Whole files and parts

A record names either locations for the whole file, in urls, or the ordered series it is composed from, in parts. sha256 describes the artefact either way, so what a version means does not depend on how it arrives. See part().

Examples

```
resource(
  name = "backbone",
  version = "2026.1",
  urls = "https://example.org/backbone-2026.1.parquet",
  sha256 = strrep("a", 64),
  size = 1048576,
```

```
  license = "CC-BY-4.0"
)

# The same artefact, published as a base and the delta issued against it:
resource(
  name = "backbone",
  version = "2026.2",
  sha256 = strrep("b", 64),
  file = "backbone.parquet",
  parts = list(
    part("https://example.org/backbone-base.bin", sha256 = strrep("c", 64)),
    part("https://example.org/backbone-2026.2.bin", sha256 = strrep("d", 64))
  )
)
```

`resource_id` *Identify a resource*

Description

A resource is identified by the triple package / name / version, never by name alone. Callers rarely build these by hand; the registry supplies the package and the resolution policy supplies the version.

Usage

```
resource_id(package, name, version)
```

Arguments

`package`	Declaring package name.
`name`	Resource name, as declared in the registry.
`version`	Resource version string.

Value

An object of class `getaca_id`.

Examples

```
resource_id("yourpkg", "backbone", "2026.1")
```

unpack *Unpack an archive or a compressed file*

Description

A stock `processor()` for the transformation almost every declaration of an archive wants: extract it, once, into its own cache slot. `getaca()` then returns the unpacked directory, and `processed = FALSE` still returns the archive it was built from.

Usage

```
unpack(format = c("auto", "zip", "tar", "gzip", "bzip2", "xz"), members = NULL)
```

Arguments

`format`	One of `"auto"`, `"zip"`, `"tar"`, `"gzip"`, `"bzip2"` or `"xz"`. `"tar"` covers the compressed tarballs; `"gzip"`, `"bzip2"` and `"xz"` are for a single compressed file.
`members`	Optional character vector of paths inside the archive to extract, instead of all of it. A name that ends a directory extracts everything under it. A name matching nothing in the archive is an error. Not applicable to a single compressed file.

Details

`format = "auto"` reads the format from the cached file's name: `.zip`; `.tar`, `.tar.gz`, `.tgz`, `.tar.bz2`, `.tbz2`, `.tar.xz` and `.txz`; and the single-file compressions `.gz`, `.bz2` and `.xz`. Name the format instead for an archive whose file name does not carry one.

A compressed single file is written under its own name with the compression extension removed, so `backbone-2026-06.csv.gz` unpacks to `backbone-2026-06.csv`. Archives keep the layout they were packed with.

The id encodes the settings, because the id is what the cache slot and the registry manifest are keyed on: `unpack()` is `"unpack"`, `unpack("zip")` is `"unpack-zip"`, and naming `members` appends a digest of them. Two records asking for different subsets therefore cannot land in one slot.

Value

An object of class `getaca_processor`, for `resource(processor = )`.

See Also

`processor()` to write your own, `resource()` to attach one.

Examples

```
unpack()
unpack("zip")$id
unpack("zip", members = "backbone/names.csv")$id

resource("backbone", "2026-06",
         urls      = "https://example.org/backbone-2026-06.zip",
         sha256    = strrep("9f", 32),
         processor = unpack())
```

Index