

Package ‘glcdp’

September 27, 2026

Title Discover, Access, and Import Global Light Commons Data Packages

Version 1.1.0

Description Discovers Global Light Commons data packages through their registry, opens immutable passing revisions, and provides searchable inventories of package metadata. Selected metadata and measurement files can be downloaded or imported with metadata-defined columns, types, factor levels, date-time values, and time zones. 'Git Large File Storage' objects are resolved without requiring an external 'Git LFS' installation, and imported file groups can be explicitly collected into data suitable for personal light exposure analysis workflows. An included 'shiny' application supports interactive discovery, inspection, selection, preview, and reproducible handoff to 'R'.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports cli, digest, dplyr, httr2, jsonlite, lubridate, readr, tibble

Suggests bslib (>= 0.6.0), knitr, rmarkdown, shiny (>= 1.7.2),
testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL <https://tscnlab.github.io/glc-dp-r/>,
<https://github.com/tscnlab/glc-dp-r>

BugReports <https://github.com/tscnlab/glc-dp-r/issues>

Config/Needs/website pkgdown

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Johannes Zauner [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0003-2171-4566>>),
Salma M. Thalji [aut, cph] (ORCID:
<<https://orcid.org/0009-0002-7320-9289>>),
Manuel Spitschan [aut, cph] (ORCID:
<<https://orcid.org/0000-0002-8572-9268>>)

Maintainer Johannes Zauner <johannes.zauner@tum.de>
Repository CRAN
Date/Publication 2026-09-27 21:20:02 UTC

Contents

add_metadata	2
extract_metadata	3
glc_collect	5
glc_collection_plan	6
glc_collection_refine	13
glc_datasets	16
glc_download	16
glc_explore	17
glc_files	18
glc_metadata	19
glc_open	20
glc_packages	21
glc_read	21
glc_resources	23
glc_schema_versions	23
glc_search_metadata	24
glc_search_packages	25
glc_summary	25
glc_variables	26
Index	28

add_metadata	<i>Add metadata to imported data</i>
--------------	--------------------------------------

Description

Extracts requested metadata with `extract_metadata()` and joins it onto every matching observation in an imported dataset.

Usage

```
add_metadata(  
  dataset,  
  metadata,  
  fields,  
  by = "file_group_id",  
  resource = NULL,  
  overwrite = FALSE  
)
```

Arguments

dataset	A data frame containing imported observations.
metadata	A metadata data frame, a local CSV or TSV path, or a package opened with glc_open() .
fields	One or more exact, top-level metadata column names to select.
by	One common identifier column, or a named character mapping from the dataset column to the metadata column. The default is "file_group_id". Use "Id" for explicitly dataset-level extraction, or c(Id = "dataset_internal_id") for a differently named metadata key.
resource	An optional declared resource name when metadata is a glc_package. For file-group or dataset identifiers, omitting resource searches declared resources connected through the package's file-group, dataset, participant, study, and device relationships. Each requested field must resolve to exactly one connected resource. For other by mappings, exactly one declared resource must contain the metadata join column and a requested field.
overwrite	Replace existing dataset columns that have the same names as extracted metadata fields. The default is FALSE.

Value

`add_metadata()` returns the original dataset with the requested metadata columns added. Row order, row count, and dplyr grouping are preserved.

Examples

```
dataset <- tibble::tibble(
  file_group_id = c("DS1:1", "DS1:1", "DS2:1"),
  value = c(1, 2, 3)
)
metadata <- tibble::tibble(
  file_group_id = c("DS1:1", "DS2:1"),
  condition = c("control", "intervention")
)

add_metadata(dataset, metadata, fields = "condition")
```

extract_metadata

Extract metadata for imported data

Description

Selects requested metadata fields for the identifiers represented in an imported dataset. By default, the result contains one row per unique file group and can be joined back with [add_metadata\(\)](#).

Usage

```
extract_metadata(
  dataset,
  metadata,
  fields,
  by = "file_group_id",
  resource = NULL
)
```

Arguments

dataset	A data frame containing imported observations.
metadata	A metadata data frame, a local CSV or TSV path, or a package opened with <code>glc_open()</code> .
fields	One or more exact, top-level metadata column names to select.
by	One common identifier column, or a named character mapping from the dataset column to the metadata column. The default is "file_group_id". Use "Id" for explicitly dataset-level extraction, or <code>c(Id = "dataset_internal_id")</code> for a differently named metadata key.
resource	An optional declared resource name when metadata is a <code>glc_package</code> . For file-group or dataset identifiers, omitting resource searches declared resources connected through the package's file-group, dataset, participant, study, and device relationships. Each requested field must resolve to exactly one connected resource. For other by mappings, exactly one declared resource must contain the metadata join column and a requested field.

Details

Identifiers are compared as character values, while the identifier column in the returned table retains its original class. Metadata-only identifiers are ignored. If metadata is a `glc_package`, the default file-group link may assemble fields from the linked dataset, participant, study, and device records. Dataset-level fields therefore repeat across file groups. Missing participant, study, or device links are retained as missing values with a warning. Use `by = "Id"` for one row per dataset; device fields then error when a dataset is linked to multiple devices.

For a grouped dataset, the grouping columns are placed before the extraction key and the original dplyr grouping (including its `.drop` setting) is retained. Each extraction-key value must map to exactly one combination of grouping-column values.

If only some input identifiers match, unmatched identifiers are retained with missing metadata and a warning is issued. If only some fields exist, the missing fields are omitted and a warning is issued.

Custom metadata are best stored in a declared data-package resource, for example `data/metadata.csv`, rather than discovered from the working directory or neighboring files.

Value

`extract_metadata()` returns a tibble with one row per unique value of `by` in first-occurrence order. The dataset's dplyr grouping columns and grouping are retained, with the `by` column added when it is not already a grouping column. With the default `by`, this is one row per file group.

Examples

```

dataset <- tibble::tibble(
  Id = c("DS1", "DS1", "DS2"),
  file_group_id = c("DS1:1", "DS1:1", "DS2:1"),
  value = c(1, 2, 3)
) |>
  dplyr::group_by(Id)
metadata <- tibble::tibble(
  file_group_id = c("DS1:1", "DS2:1"),
  condition = c("control", "intervention")
)

extract_metadata(dataset, metadata, fields = "condition")

if (interactive()) {
  package <- glc_open("owner/repository")
  imported <- glc_read(package, dataset_id = "DS1") |>
    glc_collect()
  extract_metadata(
    imported,
    package,
    fields = c("participant_age", "study_title", "device_model")
  )
  extract_metadata(imported, package, "dataset_timezone", by = "Id")
}

```

glc_collect

*Collect compatible file groups***Description**

Explicitly combines file-group tibbles after checking their columns, types, factor contracts, time zones, modalities, roles, data states, and relationship consistency. Compatible unordered factor declarations are harmonized with the same deterministic union used by [glc_collection_plan\(\)](#). Conflicting value, label, description, or order mappings remain blocking. Distinct stable file groups in one dataset may reference different devices because device identity is resolved by file_group_id.

Usage

```
glc_collect(x, standardize = c("lightlogr", "none"))
```

Arguments

x	A collection returned by glc_read() .
standardize	Either "lightlogr" to add the conventional Id, file_group_id, participant_Id, Datetime, and file.name columns and remove internal .glc_* provenance columns, or "none" to retain source and provenance columns unchanged.

Details

Collections returned by current `glc_read()` carry fingerprinted raw factor declarations. `glc_collect()` first verifies those facts against each parsed factor, computes a safe active union, and recasts the factors before binding. Invalid, changed, or conflicting contracts use condition classes `glcdp_factor_contract_invalid`, `glcdp_factor_contract_tampered`, or `glcdp_factor_harmonization_conflict`. The latter also inherits from `glcdp_incompatible_collection`. Legacy `glc_data_collection` objects without a factor-contract payload retain strict exact factor-level comparison.

A repeated `file_group_id` must still have one consistent dataset, study, participant, and device relationship. Each dataset must retain consistent study and participant relationships. These runtime checks are not weakened by file-group-scoped device identity.

Value

A combined tibble. In LightLogR-standardized output, `Id` contains the dataset id, `file_group_id` identifies the source file group, `participant_Id` contains the participant id, an existing source `file.name` column is retained, and the result is grouped by `Id`.

Examples

```
iztech <- glc_open("tscnlab/melidos-iztech-glc-dataset")
collection <- glc_read(
  iztech,
  dataset_id = "MELIDOS_IZTECH_S001",
  file_group = "MELIDOS_IZTECH_S001:17",
  n_max = 10
)
glc_collect(collection)
```

<code>glc_collection_plan</code>	<i>Plan declaration-compatible collection units</i>
----------------------------------	---

Description

Build a deterministic plan from validated declarations and normalized core metadata. The plan separates structural compatibility sets from final collectable units, and it never reads or inspects measurement contents.

Usage

```
glc_collection_plan(
  x,
  terms = NULL,
  variable_scope = c("matched", "all", "selected"),
  variables = NULL,
  dataset_id = NULL,
  file_group = NULL,
  standardize = c("lightlogr", "none")
)
```

Arguments

x	A glc_package opened with <code>glc_open()</code> at an exact, verified revision. A remote package must be at the registry's latest passing revision. A local package must be manifest-backed, with the same exact revision and <code>registry_verified = true</code> .
terms	Optional exact canonical semantic-term identifiers. Labels and other display text are not identifiers. <code>terms</code> is required when <code>variable_scope = "matched"</code> .
variable_scope	Which declared source variables to plan: • "matched" selects every variable whose canonical term is one of <code>terms</code>; • "all" selects all declared variables in each candidate file group; • "selected" selects the exact source names supplied in <code>variables</code>.
variables	Exact declared source-variable names. This argument is required for <code>variable_scope = "selected"</code> and must otherwise be <code>NULL</code> .
dataset_id	Optional exact dataset identifiers restricting the candidate groups.
file_group	Optional stable file-group identifiers, such as "DS1:1", restricting the candidate groups. Numeric group indices are deliberately not accepted because they are not stable identifiers.
standardize	Expected collection output convention. "lightlogr" plans the columns produced by <code>glc_collect(standardize = "lightlogr")</code> ; "none" plans the unstandardized provenance columns. This choice changes expected output columns and unit identifiers, but not the declaration compatibility partition.

Details

`terms` always acts as the file-group discovery predicate. A candidate group must declare every requested term, while one term may be declared by more than one variable. With `variable_scope = "matched"`, all variables carrying any requested term are selected. With "all", all declared variables are selected after the optional term predicate is applied. With "selected", `terms` remains an optional, independent discovery predicate and every requested source name must be declared by a group. Input order does not affect the result; selected variables retain declaration order.

`dataset_id` and `file_group` are intersecting restrictions. Omitting a restriction and explicitly supplying every possible identifier select the same groups, but deliberately remain different requests and therefore may produce different unit identifiers.

Value

A serializable `glc_collection_plan` object using plan schema "glc-collection-plan" version "1.2.0", as described in **Return tables**.

Structural compatibility and final units

Included groups are first compared by selected source names and declaration order, declared types, time zone, ordered modalities, role, data state, and datetime contract. Collection-based datetime values are record-specific and are ignored when comparing otherwise identical collection-based contracts.

Unordered factor declarations can share a structural set when their raw values have compatible effective labels and descriptions and their declared order constraints form an acyclic graph. An absent label means the raw value; no text normalization or semantic inference is performed. The union order is deterministic and preserves every declared order constraint. Duplicate raw values, ambiguous labels, conflicting labels or non-missing descriptions, cyclic order constraints, and unsupported ordered factors remain blocking. Blocking families retain exact factor-contract partitions.

Each structural set has a `compatibility_id` beginning with "glcc_". It is a version 2 SHA-256 digest of length-prefixed canonical UTF-8 text containing the exact package identity, repository, source revision, package schema, and the full structural-family contract computed before `dataset_id` and `file_group` restrictions are applied. It does not contain current membership, restrictions, metadata facets, or `standardize`. The identifier therefore stays the same when an unchanged family is narrowed at the same package revision. Wearing position, device identity or location, participant characteristics, file description, site context, and other descriptive metadata do not split structural sets.

Final units enforce consistent stable file-group relationships and dataset study and participant relationships. Device identity is file-group-scoped, so distinct file groups in one dataset may reference different devices while remaining in one final unit. The active factor union is recomputed after restrictions. `glc_read()` carries the raw declaration contract and `glc_collect()` validates and applies the same union to actual factors.

A `unit_id` begins with "glcu_" and uses version 2 canonical identity. It hashes the package and revision, normalized request including restrictions and `standardize`, active structural contract, and sorted member file-group identifiers. Final unit ids are request- and membership-sensitive, while structural ids are restriction stable. Both use canonical UTF-8 text rather than R serialized-object bytes, and their values and table order are stable under input row reordering. Version 2 identifiers deliberately differ from earlier identifiers because factor equivalence and device allocation rules changed.

`compatibility_diagnostics` explains safe unions and blocking differences. Selected-variable diagnostics describe the current partition. Non-selected diagnostics disclose what would require harmonization or block a later expanded variable request without selecting those variables now. Re-plan an expanded request before reading or collecting additional variables.

Declaration-only assurance

The planner uses the validated descriptor and core metadata associated with `x`. One explicit planning call may load descriptor-declared resources named `study`, `participants`, `participant_characteristics`, `datasets`, `devices`, `device_datasheets`, and optional contributors at the exact source revision. For a remote package, loading an uncached core resource can make an HTTP request. The allowlist is exactly the value returned internally by `glc_core_resource_names()`.

The planner does not call `glc_read()`, `glc_collect()`, `glc_files()`, `glc_summary()`, or `glc_download()`. It never requests a measurement path, probes measurement availability, or inspects source rows. Known declaration-level constraints, including reserved source names that begin with `.glc_`, are applied before units are formed.

File sizes come only from an explicit supported byte declaration or an entry already present in the local manifest. The planner never downloads a file or probes a remote object to discover its size; unavailable sizes remain NA. A unit's `declared_bytes` is the sum of known sizes, and `declared_bytes_complete` records whether every file size is known.

Compatibility is an assurance about validated declarations, not downloaded values. Actual columns, parsed classes and factor values, datetime values, and output-column collisions can only be checked after reading. `glc_read()` remains authoritative for source-file validation. `glc_collect()` validates the preserved raw factor declarations, harmonizes only safe unions, and remains authoritative for final compatibility and standardization.

Missing values and relationship links

Source missingness is retained. Missing scalar metadata stays as a typed NA, and absent repeated metadata stays an empty vector or list. Literal source values such as "not applicable" remain literal values. The planner does not synthesize a description, instrument, contributor id, or site id.

Relationship status columns use "linked", "not_applicable", "unresolved", or "metadata_unavailable". "not_applicable" means that no link applies, such as a dataset not associated with a participant or a group without a device id. "unresolved" means that an applicable id is missing or does not resolve in loaded metadata. "metadata_unavailable" means that an id is present but its optional core resource was not declared.

Recommended interactive workflow

Build one plan as an explicit planning task. Present one reader-oriented option per row of `compatibility_sets`, then filter groups and the normalized metadata tables in memory by stable ids. Do not rebuild the plan for each participant, device, position, characteristic, or site filter. Finally pass the narrowed file-group ids to `glc_collection_refine()`. A caller should proceed to reading only when the refinement reports one final unit and `final_selection_required = FALSE`. Keep the parent plan because the lightweight refinement does not copy its metadata tables.

Return tables

The result is a plain, serializable list with class `glc_collection_plan` and these components:

- `plan_schema` and `plan_version`: the top-level schema id "glc-collection-plan" and its semantic version.
- `provenance`: `package_id`, `repository`, `source_type`, `exact_source_revision`, `package_schema_version`, `verification`, `latest_pass_commit`, `registry_generated_at`, `manifest_version`, `declaration_metadata_fingerprint`, `planner_schema`, and `planner_version`. It contains no package handle, token, cache path, or temporary path.
- `request`: `normalized_terms`, `fixed_term_match = "all"`, `term_identifier = "canonical"`, `labels_used_for_matching = FALSE`, `variable_scope`, `requested_variables`, `dataset_id`, `file_group`, `standardize`, and the sorted union `resolved_variables` from included groups.
- `assurance`: `basis`, `actual_data_status`, `final_validation`, measurement transfer and inspection flags, `core_metadata_transport`, interactive-filter and refinement network flags, and `byte_policy`.
- `preferred_unit_id`: the preferred unit, or `NA_character_` when no unit is collectable. Preference is deterministic: most datasets, then most file groups, then the lexically smallest unit id.
- `units`: one row per final unit. Columns are `unit_id`, `compatibility_id`, `preferred`, `dataset_count`, `file_group_count`, `variable_count`, `file_count`, `declared_bytes`, `known_file_count`,

unknown_file_count, declared_bytes_complete, harmonization_required, and list-columns harmonized_variables, diagnostic_ids, and file_group_ids.

- **groups:** one row per declared file group. Columns are status, unit_id, compatibility_id, dataset_id, integer declaration index file_group, stable file_group_id, study_id, participant_id, participant_associated, study_link_status, participant_link_status, device_id, device_link_status, datasheet_id, device_location, device_location_type, description, instructions, instrument_declared, dataset_timezone, dataset_latitude, dataset_longitude, format, timezone, list-column modalities, modality_other, modality_other_type, role, data_state, temporal_type, temporal_value, temporal_unit, header_row, list-column preprocessing, datetime_source, datetime_date, datetime_format, datetime_time, datetime_time_format, and list-columns selected_variables, reason_codes, and messages. Excluded groups have missing unit_id and compatibility_id values.
- **variables:** one row per selected variable in an included group. Columns are unit_id, dataset_id, file_group_id, position, name, label, description, unit, calibration, type, canonical term, term_name, primary, list-columns factor_values, factor_labels, and factor_descriptions, and selection_origin.
- **read_columns:** one row per selected or automatically required source column. Columns are unit_id, dataset_id, file_group_id, position, name, declared_type, origin, selected_for_output, automatic, and message. Datetime source columns needed only for parsing are automatic read columns, not requested output variables.
- **output_columns:** one row per expected post-collection column and unit. Columns are unit_id, position, name, source_declared_type, expected_type, origin, automatic, runtime_validation_required, collision_validation_required, and message. These expectations are still subject to [glc_collect\(\)](#) validation.
- **files:** one row per declared file in included or excluded groups. Columns are status, unit_id, dataset_id, file_group_id, position, declared_path, format, encoding, declared_bytes, and bytes_known.
- **compatibility:** one row per unit. Columns are unit_id, list-columns selected_names, declared_types, factor_values, factor_labels, and factor_descriptions, harmonization_required, list-columns harmonized_variables and diagnostic_ids, timezone, list-column modalities, role, data_state, datetime_source, datetime_signature, datetime_date, datetime_format, datetime_time, datetime_time_format, collection_values_ignored, relationship_rule, device_rule, standardize, and the fixed standardize_affects_partition = FALSE assurance.
- **extensions:** one row per group. Columns are dataset_id, file_group_id, and preserved, forward-compatible unknown declaration fields in the plain list-column metadata.
- **compatibility_sets:** one row per structural set. Columns are compatibility_id, dataset, file-group, variable, and file counts; byte summaries; harmonization_required; list-columns harmonized_variables and diagnostic_ids; final_unit_count; final_selection_required; list-columns constraint_codes, constraint_messages, file_group_ids, and final_unit_ids; the selected-name, declared-type, factor, timezone, modality, role, data-state, and datetime contract columns also present in compatibility; relationship_rule; device_rule; and the fixed standardize_affects_structure = FALSE assurance.
- **compatibility_diagnostics:** one row per stable diagnostic. Columns are diagnostic_id, selection_scope, classification, code, variable_name, applies_to_current_plan, list-column prospective_scopes, message, affected_group, structure, and unit counts, and

list-columns file_group_ids, compatibility_ids, unit_ids, union_values, union_labels, and union_descriptions.

- compatibility_diagnostic_groups: one row per affected diagnostic and file-group pair. Columns are diagnostic_id, dataset_id, stable file_group_id, current_status, compatibility_id, unit_id, variable_present, selected_by_request, declaration_position, declared_type, and factor value, label, and description list-columns.
- metadata: a normalized typed core-metadata snapshot described below.
- refinement_input: a compact, serializable input used and validated by `glc_collection_refine()`. It contains schema, version, canonicalization, digest_algorithm, compact provenance and request lists, a membership table, plain-list structural contracts, per-group declarations and relationship facts, and a SHA-256 fingerprint. Consumers should not modify or reconstruct it.

All tables are tibbles with stable columns, including when they have no rows. List-columns contain only plain serializable vectors and lists. `print()` shows a compact package, request, unit, diagnostic, harmonization, byte, preferred-unit, and assurance summary and returns the plan invisibly.

Normalized metadata tables

metadata has schema "glc-package-metadata", version "1.0.0", and the following stable tables. All identifier joins are explicit; there is no sites table because the supported source schema has no stable site id.

- resource_status: resource, declared, status, and record_count. Status is "not_declared", "loaded_empty", or "loaded".
- studies: study_id, schema_version, title, short_description, preregistration, registration, ethics, sample, intervention, setting, geographical_location, study_type, and list-columns funding_sources, keywords, and dataset_ids.
- study_groups: study_id, position, name, description, size, and list-columns inclusion, exclusion, and dataset_ids.
- study_contributors: study_id, position, full_name, list-column roles, email, orcid, institution_name, institution_city, and institution_country.
- contributors: contributor_id, deterministic row key position, full_name, list-column roles, email, orcid, institution_name, institution_city, and institution_country. A missing source id remains typed NA; no id is synthesized.
- datasets: dataset_id, schema_version, study_id, study_link_status, participant_id, participant_associated, participant_link_status, timezone, numeric latitude and longitude, file_group_count, file_count, and list-columns modalities, device_ids, and primary_variables.
- dataset_terms: dataset_id, position, canonical term, and label.
- participants: participant_id, numeric age, sex, and gender.
- participant_characteristics: participant_id, participant_link_status, characteristic_position, value_position, name, typed scalar list-column value, value_type, unit, and description.
- devices: device_id, schema_version, manufacturer, model, serial_number, calibration_date, firmware_version, datasheet_id, and datasheet_link_status.
- device_sensors: device_id, position, sensor_type, datasheet_id, and datasheet_link_status.

- **datasheets:** `datasheet_id`, `schema_version`, `datasheet_version`, `manufacturer`, `type`, `list-column` modalities, `modality_other`, `model`, `calibration_interval`, `calibration_method`, `calibration_accuracy`, `calibration_range`, `calibration_notes`, `typed list-column` `calibration_spectral_ser`, `calibration_linearity`, and `calibration_directional_response`.
- **datasheet_parameters:** `datasheet_id`, `position`, `name`, `typed scalar list-column` `value`, `value_type`, `unit`, and `description`.
- **datasheet_channels:** `datasheet_id`, `position`, `integer` `channel_number`, `name`, `description`, and `unit`.
- **instruments:** `dataset_id`, `file_group_id`, `instrument_type`, `instrument_name`, `collection_method`, `recorded_by`, and `software_name`.
- **file_group_variables:** all declared variables for every included group, independent of `variable_scope`. Columns are `dataset_id`, `file_group_id`, `declaration_position`, `selected_by_request`, `selected_for_output`, `selection_origin`, `name`, `label`, `description`, `unit`, `calibration`, `type`, `canonical term`, `term_name`, `primary`, and `factor_level_count`.
- **file_group_factor_levels:** `dataset_id`, `file_group_id`, `variable_position`, `variable_name`, `level_position`, `value`, `label`, and `description`.
- **extensions:** `resource`, `entity_type`, `entity_id`, `parent_id`, `position`, and plain list-column metadata for forward-compatible unknown fields. Standard fields never require parsing this column.

Exclusions and errors

Per-group declaration outcomes are returned rather than thrown. Stable reason codes are included, `scope_dataset`, `scope_file_group`, `reserved_provenance_column`, `term_missing`, `variable_missing`, `invalid_factor_contract`, `no_declared_files`, `unsupported_format`, `invalid_timezone`, and `incomplete_datetime`; each has a plain-language message.

Invalid argument types, empty values, duplicates, and inconsistent `variable_scope`/selector combinations error before planning. Programmatically useful condition subclasses include `glcdp_unknown_dataset`, `glcdp_unknown_file_group`, `glcdp_unknown_term`, `glcdp_unknown_variable`, `glcdp_collection_plan_revision`, `glcdp_collection_plan_provenance`, `glcdp_collection_plan_ambiguous_variable`, `glcdp_collection_plan_rel`, `glcdp_collection_plan_serialization`, and `glcdp_collection_plan_id`. Term labels that are not canonical ids are reported as unknown terms rather than matched ambiguously.

See Also

[glc_collection_refine\(\)](#) for fast in-memory narrowing, [glc_variables\(\)](#) for declared selectors, [glc_read\(\)](#) for runtime import validation, and [glc_collect\(\)](#) for authoritative final collection.

Examples

```
## Not run:
# Use an existing local, manifest-backed package directory.
# This pattern performs no network request and does not read measurements.
pkg <- glc_open("path/to/manifest-backed-package", quiet = TRUE)
plan <- glc_collection_plan(
  pkg,
  terms = "photopic illuminance",
```

```

    variable_scope = "matched"
  )
  plan$compatibility_sets[, c(
    "compatibility_id", "file_group_count", "final_unit_count",
    "final_selection_required"
  )]

# Filter plan$groups and plan$metadata in memory, then refine exact ids.
set_id <- plan$compatibility_sets$compatibility_id[[1L]]
selected_ids <- plan$compatibility_sets$file_group_ids[[1L]]
refined <- glc_collection_refine(plan, selected_ids, set_id)
refined

## End(Not run)

```

glc_collection_refine *Refine a collection plan from stable file-group identifiers*

Description

Recompute final collection units for an in-memory subset of one structural compatibility set. Refinement validates and reuses the compact facts stored in a [glc_collection_plan\(\)](#) result. It does not reopen the package, load metadata, access a network, or inspect measurement contents.

Usage

```
glc_collection_refine(plan, file_group, compatibility_id = NULL)
```

Arguments

plan	A validated <code>glc_collection_plan</code> object with supported plan and refinement-input schema versions. Keep this parent plan after refinement; the lightweight result refers to it by schema, version, and fingerprint and does not copy its normalized metadata tables.
file_group	A character vector of unique, non-missing stable <code>file_group_id</code> values. Every value must be an included member of plan and all values must belong to one structural compatibility set. Numeric declaration indices are not accepted.
compatibility_id	Optional single structural compatibility identifier. When NULL, the identifier is inferred only if all selected groups share exactly one set. When supplied, it must equal that set's identifier.

Details

`glc_collection_refine()` is the final, fast step after interactive metadata narrowing. Build [glc_collection_plan\(\)](#) once, choose one row of `plan$compatibility_sets`, and filter its member groups through the typed tables in `plan$groups` and `plan$metadata`. Pass only the resulting stable file-group ids to this function. Input order does not affect the result.

Refinement reapplies the stored structural and relationship rules, recomputes the active deterministic factor union, and creates request-sensitive final unit ids. Its units table is identical to a fresh `glc_collection_plan()` call with the parent's original term, variable, dataset, and standardization request and with `file_group` set to the refined ids. The restriction-stable `compatibility_id` is retained. No full variable or metadata tables are rebuilt.

Device identity remains linked by stable file-group id, so different devices in one dataset do not split otherwise compatible groups. A safe active factor union is reported through the unit's `harmonization_required` and `harmonized_variables` fields. `glc_read()` and `glc_collect()` still validate actual source values and apply the same union before binding.

Value

A lightweight `glc_collection_refinement` object described in **Return structure**.

Validation and conditions

Refinement supports plan schema "glc-collection-plan" version "1.2.0" and refinement-input schema "glc-collection-refinement-input" version "1.1.0". It verifies the compact input fingerprint and checks it against the parent plan's provenance, request, and group membership. The fingerprint covers the structural-family contract, each member's declared factor contract, and relationship facts needed for refinement. It does not cover the larger normalized metadata snapshot, so validation does not rehash the complete plan. Earlier plan versions do not contain these facts and must be planned again.

All refinement errors inherit from `glcdp_collection_refine_error`. More specific subclasses are:

- `glcdp_collection_refine_plan` for a value that is not a collection plan;
- `glcdp_collection_refine_version` for an unsupported schema or version;
- `glcdp_collection_refine_incomplete` and `glcdp_collection_refine_tampered` for missing or changed parent facts;
- `glcdp_collection_refine_file_group`, `glcdp_collection_refine_empty`, and `glcdp_collection_refine_duplicate` for invalid file-group selectors;
- `glcdp_collection_refine_unknown_group` and `glcdp_collection_refine_excluded_group` for groups outside the eligible parent membership;
- `glcdp_collection_refine_compatibility`, `glcdp_collection_refine_unknown_compatibility`, and `glcdp_collection_refine_compatibility_mismatch` for invalid structural identifiers;
- `glcdp_collection_refine_cross_structure` when selected groups span more than one structural set; and
- `glcdp_collection_refine_unresolved_identity` when required study, participant, or device identities cannot be resolved from the stored metadata facts.

Zero-access assurance

Refinement performs no file or network input/output. It does not call `glc_open()`, `glc_files()`, `glc_summary()`, `glc_read()`, `glc_collect()`, `glc_download()`, or metadata-loading and materialization functions. Its assurance records that the package was not reopened, remote availability was not probed, and measurement contents were neither transferred nor inspected. `glc_read()` and `glc_collect()` remain authoritative after refinement.

Return structure

The result is a plain serializable list with class `glc_collection_refinement`, schema `"glc-collection-refinement"`, and version `"1.1.0"`. It contains:

- `parent`: `plan_schema`, `plan_version`, and the validated refinement-input fingerprint linking this result to the retained parent plan;
- `provenance`: `package_id`, `repository`, `exact_source_revision`, and `package_schema_version`;
- `request`: the selected `compatibility_id`, sorted `file_group` ids, and the parent's compact original request;
- `assurance`: declaration basis plus package, network, availability-probe, measurement-transfer, measurement-inspection, and final-validation fields;
- `compatibility_id`, `final_selection_required`, and `preferred_unit_id`;
- `units`: the same stable final-unit columns documented for [glc_collection_plan\(\)](#), including active factor harmonization fields;
- `groups`: `status`, `unit_id`, `compatibility_id`, `dataset_id`, integer `file_group`, stable `file_group_id`, `study_id`, `participant_id`, `participant_associated`, `study_link_status`, `participant_link_status`, `device_id`, `device_link_status`, and list-columns `reason_codes` and `messages`; and
- `constraints`: `compatibility_id`, `code`, `message`, and `logical_resolved`. Its schema is stable even when it has no rows.

All tables and list-columns contain only plain serializable values. The result contains no package handle, environment, token, cache path, or temporary path. `print()` reports the structural id, final-unit and group counts, final-selection status, and zero-access assurance, then returns the result invisibly.

See Also

[glc_collection_plan\(\)](#) for the required parent plan, [glc_read\(\)](#) and [glc_collect\(\)](#) for run-time validation and collection.

Examples

```
## Not run:
# Use an existing local, manifest-backed package directory.
# This pattern performs no network request and does not read measurements.
pkg <- glc_open("path/to/manifest-backed-package", quiet = TRUE)
plan <- glc_collection_plan(
  pkg,
  terms = "photopic illuminance",
  variable_scope = "matched"
)

# In an application, filter these ids with plan$groups and plan$metadata.
set <- plan$compatibility_sets[1L, ]
selected_ids <- set$file_group_ids[[1L]]
refined <- glc_collection_refine(
  plan,
```

```
      selected_ids,
      compatibility_id = set$compatibility_id[[1L]]
    )
    refined

## End(Not run)
```

glc_datasets	<i>Inventory datasets</i>
--------------	---------------------------

Description

Inventory datasets

Usage

```
glc_datasets(x, dataset_id = NULL)
```

Arguments

- x A package opened with [glc_open\(\)](#).
- dataset_id Optional dataset id or ids.

Value

A tibble with one row per dataset.

Examples

```
iztech <- glc_open("tscnlab/melidos-iztech-glc-dataset")
glc_datasets(iztech)
```

glc_download	<i>Download package metadata or data</i>
--------------	--

Description

Downloads only descriptor-declared and dataset-referenced content, preserves repository-relative paths, and writes a reproducibility manifest.

Usage

```
glc_download(
  x,
  dest_dir,
  include = c("metadata", "data", "all"),
  dataset_id = NULL,
  file_group = NULL,
  resources = NULL,
  files = NULL,
  overwrite = FALSE
)
```

Arguments

<code>x</code>	A package opened with <code>glc_open()</code> .
<code>dest_dir</code>	Destination directory.
<code>include</code>	One of "metadata", "data", or "all". Metadata is the safe default.
<code>dataset_id</code>	Optional dataset id selection for data downloads.
<code>file_group</code>	Optional file-group selection.
<code>resources</code>	Optional descriptor resource names.
<code>files</code>	Optional exact paths, declared paths, or basenames.
<code>overwrite</code>	Whether existing files may be replaced.

Value

A tibble recording downloaded paths, storage, size, and hashes.

Examples

```
iztech <- glc_open("tscnlab/melidos-iztech-glc-dataset")
destination <- tempfile("glcdp-metadata-")
glc_download(iztech, destination)
```

glc_explore

Explore Global Light Commons data packages

Description

Launches a local Shiny application for browsing the GLC registry, opening the immutable latest passing revision of a package, reviewing its contents, and filtering participants, devices, datasets, file groups, semantic terms, and source variables. The app can preview the resulting selection and export an annotated, reproducible R script without uploading package data to another service.

Usage

```
glc_explore(
  registry = NULL,
  launch.browser = getOption("shiny.launch.browser", interactive()),
  ...
)
```

Arguments

registry	Optional registry JSON URL or local path. Defaults to the official registry or the value of option <code>glcdp.registry_url</code> .
launch.browser	Whether to open the application in a browser, or a function that Shiny calls with the application URL. The default respects IDE viewer functions supplied through <code>shiny.launch.browser</code> .
...	Additional arguments passed to <code>shiny::runApp()</code> .

Value

Called for its side effect of running a Shiny application.

See Also

The [Shiny app workflow](#).

Examples

```
glc_explore()
```

glc_files

Inventory declared data files

Description

Inventory declared data files

Usage

```
glc_files(
  x,
  dataset_id = NULL,
  file_group = NULL,
  role = NULL,
  modality = NULL,
  available = NULL
)
```

Arguments

x	A package opened with <code>glc_open()</code> .
dataset_id	Optional dataset id or ids.
file_group	Optional group index or stable dataset:group id.
role	Optional file-group role.
modality	Optional modality.
available	Optional logical filter for file availability.

Value

A tibble with one row per concrete declared file, including the file-specific encoding declared by its file group.

Examples

```
iztech <- glc_open("tscnlab/melidos-iztech-glc-dataset")
glc_files(iztech, dataset_id = "MELIDOS_IZTECH_S001")
```

glc_metadata	<i>Load metadata resources</i>
--------------	--------------------------------

Description

Loads core metadata by default. Additional descriptor resources can be requested explicitly by name.

Usage

```
glc_metadata(x, resources = NULL)
```

Arguments

x	A package opened with <code>glc_open()</code> .
resources	Optional resource names. The default selects declared core metadata resources.

Value

A named list with one element per requested resource. Tabular resources are returned as tibbles; directory resources contain named sub-lists.

Examples

```
iztech <- glc_open("tscnlab/melidos-iztech-glc-dataset")
glc_metadata(iztech, resources = "study")
```

`glc_open`*Open a Global Light Commons data package*

Description

Opens a local package or resolves a GitHub-hosted package at an immutable commit. Registered packages default to their latest passing revision.

Usage

```
glc_open(  
  source,  
  ref = "latest_pass",  
  token = NULL,  
  cache_dir = NULL,  
  registry = NULL,  
  quiet = FALSE  
)
```

Arguments

<code>source</code>	Registry id, owner/repository, GitHub URL, local package directory, or local datapackage.json path.
<code>ref</code>	Remote revision: "latest_pass", "current", or an exact 40-character commit SHA.
<code>token</code>	Optional GitHub token. When omitted, GITHUB_PAT and then GITHUB_TOKEN are consulted.
<code>cache_dir</code>	Optional explicit persistent cache directory. The default uses session-temporary storage for remote reads.
<code>registry</code>	Optional registry object, URL, or local JSON path.
<code>quiet</code>	Suppress informational messages. Warnings remain visible.

Value

A `glc_package` handle.

Examples

```
package <- glc_open("tscnlab/melidos-iztech-glc-dataset")  
package
```

glc_packages	<i>List registered Global Light Commons data packages</i>
--------------	---

Description

Downloads and flattens the Global Light Commons registry. Both passing and non-passing current revisions are retained.

Usage

```
glc_packages(registry = glc_default_registry(), refresh = FALSE)
```

Arguments

registry	Registry JSON URL or path. Defaults to the official registry or the value of option <code>glcdp.registry_url</code> .
refresh	Whether to bypass the in-session registry cache.

Value

A `glc_registry` tibble with one row per registered repository.

Examples

```
packages <- glc_packages()
glc_search_packages("iztech", packages)
```

glc_read	<i>Read metadata-described dataset files</i>
----------	--

Description

Read metadata-described dataset files

Usage

```
glc_read(
  x,
  dataset_id,
  file_group = NULL,
  files = NULL,
  variables = NULL,
  terms = NULL,
  primary_only = FALSE,
  n_max = Inf,
```

```

    problems = c("error", "warn"),
    progress = interactive()
  )

```

Arguments

<code>x</code>	A package opened with <code>glc_open()</code> .
<code>dataset_id</code>	Dataset id or ids. Use "all" explicitly to read every dataset.
<code>file_group</code>	Optional group index or stable id.
<code>files</code>	Optional declared paths, resolved paths, or basenames.
<code>variables</code>	Optional source variable names.
<code>terms</code>	Optional semantic variable terms.
<code>primary_only</code>	Select only declared primary variables.
<code>n_max</code>	Maximum rows read from each file.
<code>problems</code>	Whether metadata mismatches should be errors or warnings.
<code>progress</code>	Show progress while files are imported. Defaults to TRUE in interactive sessions and FALSE otherwise.

Details

When variable filters are used, source columns required to construct datetimes are used internally but omitted unless selected by the filters. Declared files that are absent from a local package subset are skipped. When a local package contains fewer datasets or files than declared, `glc_read()` reports the discrepancy and reads the available files.

Each returned file-group row includes a plain serializable `factor_contract` payload for the selected variables. It preserves raw factor values, effective labels, descriptions, unordered status, schema version, and a SHA-256 fingerprint. `glc_collect()` validates this payload against the parsed data before applying any safe factor-level union. Invalid declarations error with class `glcdp_factor_contract_invalid`. Changed payloads or parsed factors are rejected later with class `glcdp_factor_contract_tampered`.

Value

A `glc_data_collection` tibble with one data list-column per file group and one serializable `factor_contract` list-column. The contract contains no package handle, token, cache path, or temporary path.

Examples

```

iztech <- glc_open("tscnlab/melidos-iztech-glc-dataset")
glc_read(
  iztech,
  dataset_id = "MELIDOS_IZTECH_S001",
  file_group = "MELIDOS_IZTECH_S001:17",
  n_max = 10
)

```

glc_resources	<i>Inventory data-package resources</i>
---------------	---

Description

Inventory data-package resources

Usage

```
glc_resources(x)
```

Arguments

x A package opened with `glc_open()`.

Value

A tibble with one row per declared resource path.

Examples

```
iztech <- glc_open("tscnlab/melidos-iztech-glc-dataset")
glc_resources(iztech)
```

glc_schema_versions	<i>Report supported GLC schema versions</i>
---------------------	---

Description

Report supported GLC schema versions

Usage

```
glc_schema_versions()
```

Value

A tibble describing support status for each schema version.

Examples

```
glc_schema_versions()
```

glc_search_metadata *Search metadata values or field paths*

Description

Search metadata values or field paths

Usage

```
glc_search_metadata(
  x,
  query,
  resources = NULL,
  fields = NULL,
  fixed = TRUE,
  ignore_case = TRUE,
  search_in = c("values", "fields", "both")
)
```

Arguments

x	A package opened with <code>glc_open()</code> .
query	Text or regular expression to search for.
resources	Optional metadata resource names.
fields	Optional exact field names or complete field paths to include.
fixed	Treat query as fixed text rather than a regular expression.
ignore_case	Ignore letter case while matching.
search_in	Where to match query: scalar metadata "values", complete field paths "fields", or "both". The default is "values".

Details

Field searches return the same leaf-level rows as value searches. A field path that contains multiple scalar values therefore produces one row per value. The `fields` argument can be combined with any `search_in` mode to restrict which field paths are searched.

Value

A tibble of matching scalar metadata values and their field paths.

Examples

```
iztech <- glc_open("tscnlab/melidos-iztech-glc-dataset")
glc_search_metadata(iztech, "Izmir", resources = "study")
```

glc_search_packages	<i>Search registered data packages</i>
---------------------	--

Description

Search registered data packages

Usage

```
glc_search_packages(  
  query = NULL,  
  packages = glc_packages(),  
  status = NULL,  
  has_pass = NULL  
)
```

Arguments

query	Optional fixed, case-insensitive text searched in package ids and repository names.
packages	A registry returned by glc_packages() .
status	Optional current validation status or statuses.
has_pass	Optional logical value selecting packages with or without a recorded passing revision.

Value

A filtered glc_registry tibble.

Examples

```
packages <- glc_packages()  
glc_search_packages("iztech", packages)  
glc_search_packages(packages = packages, status = "pass")
```

glc_summary	<i>Summarize a Global Light Commons data package</i>
-------------	--

Description

Summarize a Global Light Commons data package

Usage

```
glc_summary(x)
```

Arguments

x A package opened with `glc_open()`.

Value

A one-row `glc_summary` tibble. For local packages, declared and locally available dataset, file-group, and file counts are reported separately.

Examples

```
iztech <- glc_open("tscnlab/melidos-iztech-glc-dataset")
glc_summary(iztech)
```

glc_variables	<i>Inventory and search declared variables</i>
---------------	--

Description

Inventory and search declared variables

Usage

```
glc_variables(
  x,
  dataset_id = NULL,
  file_group = NULL,
  term = NULL,
  primary = NULL
)
```

Arguments

x A package opened with `glc_open()`.
dataset_id Optional dataset id or ids.
file_group Optional group index or stable id.
term Optional semantic term or terms.
primary Optional logical filter for primary variables.

Value

A tibble with one row per declared variable, including its declared type and factor values, labels, and descriptions.

Examples

```
iztech <- glc_open("tscnlab/melidos-iztech-glc-dataset")
glc_variables(
  iztech,
  file_group = "MELIDOS_IZTECH_S001:17",
  primary = TRUE
)
```

Index

`add_metadata`, [2](#)
`add_metadata()`, [3](#)

`extract_metadata`, [3](#)
`extract_metadata()`, [2](#)

`glc_collect`, [5](#)
`glc_collect()`, [8–10](#), [12](#), [14](#), [15](#), [22](#)
`glc_collection_plan`, [6](#)
`glc_collection_plan()`, [5](#), [13–15](#)
`glc_collection_refine`, [13](#)
`glc_collection_refine()`, [9](#), [11](#), [12](#)
`glc_datasets`, [16](#)
`glc_download`, [16](#)
`glc_download()`, [8](#), [14](#)
`glc_explore`, [17](#)
`glc_files`, [18](#)
`glc_files()`, [8](#), [14](#)
`glc_metadata`, [19](#)
`glc_open`, [20](#)
`glc_open()`, [3](#), [4](#), [7](#), [14](#), [16](#), [17](#), [19](#), [22–24](#), [26](#)
`glc_packages`, [21](#)
`glc_packages()`, [25](#)
`glc_read`, [21](#)
`glc_read()`, [5](#), [6](#), [8](#), [9](#), [12](#), [14](#), [15](#)
`glc_resources`, [23](#)
`glc_schema_versions`, [23](#)
`glc_search_metadata`, [24](#)
`glc_search_packages`, [25](#)
`glc_summary`, [25](#)
`glc_summary()`, [8](#), [14](#)
`glc_variables`, [26](#)
`glc_variables()`, [12](#)

`shiny::runApp()`, [18](#)