Package 'Rega'

May 15, 2026

Title R Interface to European Genome-Phenome Archive

Version 1.0.0

Description The European Genome-phenome Archive (EGA) provides long-term storage and controlled sharing of personally identifiable genetic data. The Rega package offers a streamlined and extensible R interface to the EGA API, facilitating the programmatic upload of metadata. GEO-like Excel submission template is provided as a default method of organizing submission metadata.

License Artistic-2.0

URL https://github.com/ivanek/Rega

BugReports https://github.com/ivanek/Rega/issues

Depends R (>= 4.6)

Imports askpass, httr2, jsonlite, jsonvalidate, keyring, readxl, rlang, stringr, tibble, tidyr, validate, yaml

Suggests BiocStyle, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

biocViews Software, Infrastructure, ThirdPartyClient

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

git_url https://git.bioconductor.org/packages/Rega

git_branch RELEASE_3_23

git_last_commit 5a60572

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-05-14

Author Igor Cervenka [aut, cre] (ORCID: <https://orcid.org/0000-0002-9438-5161>), Athimed El Taher [aut] (ORCID: <https://orcid.org/0000-0003-2424-8476>), Robert Ivanek [aut] (ORCID: <https://orcid.org/0000-0002-8403-056X>)

Maintainer Igor Cervenka <igor.cervenka@unibas.ch>

Contents

.add_headers . . . 4
.add_json_validation . . . 4
.add_paths . . . 5
.add_queries . . . 6
.add_request_body . . . 6
.analyses_extra_validator . . . 7
.basic_validator . . . 8
.datasets_extra_validator . . . 9
.dataset_analyses_validator . . . 10
.EGA_TOKEN_URL . . . 11
.get_ega_password . . . 11
.get_ega_username . . . 12
.get_operation_params . . . 12
.get_rega_key . . . 13
.has_analyses . . . 14
.is_client . . . 14
.is_scalar . . . 15
.operation_params_to_args . . . 16
.runs_extra_validator . . . 16
.studies_extra_validator . . . 17
.submission_validator . . . 18
.summarise_validation . . . 19
.validate_character_scalar . . . 19
.validate_logical_scalar . . . 20
add_required_str . . . 21
aliases_formatter . . . 22
api_function_factory . . . 22
api_name_to_label . . . 23
column_table_formatter . . . 24
create_client . . . 25
default_parser . . . 25
default_validator . . . 26
delete_submission . . . 27
delete_submission_contents . . . 28
DELIM_CONVERTERS . . . 29
ega_oauth . . . 30
ega_token . . . 31
extract_api . . . 32
extract_operation_definitions . . . 32
extract_resource_name . . . 33
fetch_files . . . 34
file_formatter . . . 35
filter_id_fields . . . 35
finalise_submission . . . 36
first_row_to_colnames . . . 37
fold_column . . . 38

format_chromosomes . . . 39
get_chr_group . . . 40
get_entry_by_title . . . 40
get_enum . . . 41
get_formatter . . . 42
get_formatter_params . . . 43
get_operation_schema . . . 44
get_or_post . . . 44
get_properties . . . 46
get_schemas . . . 47
get_sentence_number . . . 47
get_submission . . . 48
get_word_number . . . 49
has_linked_sheets . . . 49
is_accession . . . 50
is_provisional . . . 51
is_valid_http_method . . . 51
label_to_api_name . . . 52
link_sheet . . . 52
lut_add . . . 53
merge_linked_sheet . . . 54
multi_lut_add . . . 54
na_to_empty_list . . . 55
new_submission . . . 56
parse_ega_body . . . 57
parse_enum . . . 58
parse_json_body . . . 59
parse_text_body . . . 60
process_chromosomes . . . 61
process_delimited_column . . . 62
rollback_submission . . . 63
row_table_formatter . . . 63
samples_in_db . . . 64
save_log . . . 65
step_msg . . . 66
submit_table . . . 66
try_step . . . 67
unbox_list . . . 68
unbox_row . . . 68
use_submission . . . 69
validate_schema . . . 70
validation_to_msg . . . 71
workflow_error_handler . . . 71

Index **73**

.add_headers *Generate Header Expressions for an API Request*

Description

This function creates expressions to add headers to an API request, including content type, authorization, and any additional headers specified in the parameters.

Usage

```
.add_headers(header_params, operation, api, token = NULL)
```

Arguments

header_params	A character vector of header parameter names to include in the request. Header parameters are created .get_operation_params function.
operation	List. The operation definition, which may include security details.
api	List. The API definition, which may include global security details and other metadata.

Value

An expression to add headers to an API request using req_headers().

Examples

```
api <- extract_api()
opdefs <- extract_operation_definitions(api)
params <- Rega:::.get_operation_params(opdefs[["get__files"]])
# No header parameters in operation, `Content-Type` added by default
Rega:::.add_headers(params$header, opdefs[["get__files"]], api)
```

.add_json_validation *Add JSON Schema Validation to API Operation*

Description

Generates validation expressions for an API operation based on its JSON schema. If a schema is present, the function returns expressions to validate the request body and raise an error if validation fails.

Usage

```
.add_json_validation(op)
```

Arguments

op	A list representing the API operation, which may contain a request body and schema.

Value

A list of expressions for JSON schema validation, or an empty list if no schema is found.

Examples

```
## Not run:
op <- list(requestBody = TRUE, schema = list(type = "object"))
Rega:::.add_json_validation(op)

## End(Not run)
```

.add_paths *Generate URL Parameter Replacement Expressions for an API Request*

Description

This function creates a list of expressions to replace placeholders in a URL with corresponding parameter values.

Usage

```
.add_paths(path_params)
```

Arguments

path_params	A character vector of names of the parameters to replace in the URL. Each name should correspond to a placeholder in the URL in the format {param}. Path parameters are created .get_operation_params function.

Value

A list of expressions. Each expression replaces a {param} placeholder in the URL with the value of the corresponding parameter.

Examples

```
# Generate replacement expressions for path parameters
Rega:::.add_paths(c("id", "type"))
```

.add_queries *Generate Query Expressions for an API Request*

Description

This function creates an expression to add query parameters to an API request.

Usage

```
.add_queries(query_params)
```

Arguments

query_params	A character vector of query parameter names to include in the request. Query parameters are created .get_operation_params function.

Value

An expression to add query parameters to an API request using req_url_query().

Examples

```
Rega:::.add_queries("queries")

opdefs <- extract_operation_definitions(extract_api())
params <- Rega:::.get_operation_params(opdefs[["get__files"]])
Rega:::.add_queries(params$query)
```

.add_request_body *Add Request Body to API Request*

Description

Adds a JSON request body to an API request if required. If the request requires a body, an expression is returned to include it; otherwise, an empty list is returned.

Usage

```
.add_request_body(has_body)
```

Arguments

has_body	A logical value indicating whether the request requires a body.

Value

An expression to add the JSON request body if has_body is TRUE, otherwise an empty list.

Examples

```
Rega:::.add_request_body(TRUE)
Rega:::.add_request_body(FALSE)
```

.analyses_extra_validator

Validate Analyses Metadata

Description

Validates the metadata by checking the following:

- analyses titles are unique
- analyses descriptions are unique
- experiments for analyses are present in aliases
- samples for analyses are present in aliases
- files for analyses are specified
- files for analyses are unique

Usage

```
.analyses_extra_validator(meta, aliases)
```

Arguments

meta	A list containing parsed EGA metadata, where elements correspond to individual sheets.
aliases	A list of aliases, reference values used for validation.

Value

A validation object containing the results of the check or NULL if analyses sheet was not present in metadata

Examples

```
meta <- list(analyses = data.frame(
    title = c("TitleA", "TitleB", NA),
    description = c("DescriptionA", "DescriptionB", "DescriptionA"),
    samples = I(
        list(c("Sample1", NA), c("Sample2", "Sample3"), c("Sample3"))
    ),
    experiments = I(list(NA, c("ExpA", "ExpB"), c("ExpD"))),
    files = I(list(c("File1", "File2"), c("File3"), NA))
))

aliases <- list(
    samples = c("Sample1", "Sample3"),
    experiments = c("ExpA", "ExpB", "ExpC")
)

validate::summary(Rega:::.analyses_extra_validator(meta, aliases))
```

.basic_validator *Create a Basic Validator for Data Columns*

Description

Generates a validator to check multiple conditions on a specified data column. Following validations are performed:

- are all entries specified (non-NA)
- are all entries unique
- are all entries present in aliases
- are all aliases present in the sheet

Usage

```
.basic_validator(meta, aliases, column, code_list_entry)
```

Arguments

`meta`	A list containing parsed EGA metadata, where elements correspond sheets, which contain columns to be validated.
`aliases`	A named list of reference values used for validation.
`column`	A string specifying the name of the column to validate.
`code_list_entry`	A string specifying the alias entry in `aliases` and sheet name in metadata to compare against.

Value

A validation object containing the results of the check or NULL if sheet specified by `ode_list_entry` was not present in metadata

Examples

```
meta <- list(sample_codes = data.frame(sample_ids = c("A", "B", "C")))
aliases <- list(sample_codes = c("A", "B", "C", "D"))
Rega:::.basic_validator(meta, aliases, "sample_ids", "sample_codes")
```

.datasets_extra_validator

Validate Datasets Metadata

Description

Validates the metadata by checking the following:

- datasets titles are unique
- datasets descriptions are unique
- runs for datasets are present in aliases (as nested lists)
- all runs in aliases are present in datasets (as nested lists)
- datatsets titles contains number of words within required range
- datasets descriptions contains number of sentences within required range

Usage

```
.datasets_extra_validator(meta, aliases)
```

Arguments

`meta`	A list containing parsed EGA metadata, where elements correspond to individual sheets.
`aliases`	A list of aliases, reference values used for validation.

Value

A validation object containing the results of the check.

Examples

```
meta <- list(datasets = data.frame(
    title = c("TitleA", "TitleB", NA),
    description = c("DescriptionA", "DescriptionB", "DescriptionA"),
    runs = I(list(c("Run1", NA), c("Run2", "Run3"), c("Run3")))
))

aliases <- list(
    runs = c("Run1", "Run3", "Run4")
)

validate::summary(Rega:::.datasets_extra_validator(meta, aliases))
```

.dataset_analyses_validator

Validate Datasets Analyses Metadata

Description

Validates the metadata by checking the following:

- analyses for datasets are present in aliases

Usage

```
.dataset_analyses_validator(meta, aliases)
```

Arguments

meta	A list containing parsed EGA metadata, where elements correspond to individual sheets.
aliases	A list of aliases, reference values used for validation.

Value

A validation object containing the results of the check or NULL if analyses sheet was not present in metadata

Examples

```
meta <- list(
    datasets = data.frame(
        analyses = c("AnalysisA", "AnalysisB", NA)
    ),
    analyses = data.frame(
        names = c("A", "B")
    )
)
```

```
aliases <- list(
    analyses = c("AnalysisA", "AnalysisC")
)

validate::summary(
    Rega:::.dataset_analyses_validator(meta, aliases)
)
```

.EGA_TOKEN_URL *Default URL for EGA OAuth token*

Description

Default URL for EGA OAuth token

Usage

```
.EGA_TOKEN_URL
```

Format

An object of class `character` of length 1.

.get_ega_password *Retrieve EGA Password*

Description

Fetches the EGA password from the system keyring, an environment variable, or an interactive prompt, following that priority order. If using the environment variable, the value is decrypted using a key retrieved via `.get_rega_key`.

Usage

```
.get_ega_password(keyring_name = "REGA_EGA", envvar = "REGA_EGA_PASSWORD", ...)
```

Arguments

`keyring_name`	Character. Name of the service in the keyring. Defaults to `"REGA_EGA"`.
`envvar`	Character. Name of the environment variable. Defaults to `"REGA_EGA_PASSWORD"`.
`...`	Additional arguments passed to `.get_rega_key`.

Value

A character string containing the password.

Examples

```
try(
    ega_password <- Rega:::.get_ega_password()
)
```

`.get_ega_username` *Retrieve EGA Username*

Description

Fetches the EGA username from the system keyring, an environment variable, or an interactive prompt, following that priority order.

Usage

```
.get_ega_username(keyring_name = "REGA_EGA", envvar = "REGA_EGA_USERNAME")
```

Arguments

`keyring_name`	Character. Name of the service in the keyring. Defaults to `"REGA_EGA"`.
`envvar`	Character. Name of the environment variable. Defaults to `"REGA_EGA_USERNAME"`.

Value

A character string containing the username.

Examples

```
tryCatch(
    Rega:::.get_ega_username("REGA_EGA_USERNAME")
)
```

`.get_operation_params` *Extract Operation Parameters by Location*

Description

This function organizes API operation parameters into categories based on their location (`path`, `query`, or `header`).

Usage

```
.get_operation_params(op)
```

Arguments

op	List. An operation definition containing a parameters element, which is a list of parameter definitions. Each parameter should include a name and an in field specifying its location.

Value

A named list with elements:

- path: Character vector of path parameter names.
- query: Character vector of query parameter names.
- header: Character vector of header parameter names.

Examples

```
# Convert operation parameters to function arguments
opdefs <- extract_operation_definitions(extract_api())

# Extract parameters categorized by location
Rega:::.get_operation_params(opdefs[["get__files"]])
```

.get_rega_key *Retrieve Rega secret from Environment Variable*

Description

Retrieves the Rega secret from the specified environment variable. If the key is not found, a warning message is issued. REGA secret should be generated via `httr2::secret_make_key()` and stored as environmental variable either by using `export` command in bash or at the the user-level in the `.Renviron` file

Usage

```
.get_rega_key(secret_ennvar = "REGA_KEY")
```

Arguments

secret_ennvar	A string specifying the name of the environment variable to retrieve the API key from. Defaults to "REGA_KEY".

Value

A string containing the REGA API key, or an empty string if the variable is not set.

Examples

```
try(
    rega_key <- Rega:::.get_rega_key()
)
```

`.has_analyses` *Check for Presence of Analyses Sheet in Metadata*

Description

Determines whether analyses or analysis files are specified in the provided metadata.

Usage

```
.has_analyses(meta)
```

Arguments

`meta`	A list containing metadata with elements `analyses` and `analysis_files`.

Value

A logical value: `TRUE` if analyses are absent or empty, `FALSE` otherwise.

Examples

```
meta <- list(
    aliases = list(analyses = c("A1", "A2")),
    analysis_files = data.frame(name = character(0))
)
Rega:::.has_analyses(meta)
```

`.is_client` *Validate that an object is an API client.*

Description

Checks that `client` is a list where every element is a function. Additionally, it verifies that the name of each function starts with a valid HTTP method (e.g., `get_`, `post_`, etc.). It stops execution with an error if any of the checks fail.

Usage

```
.is_client(client)
```

Arguments

client	An object to be tested, expected to be a list of functions.

Value

Returns TRUE if the validation succeeds, otherwise stops execution with an error.

Examples

```
# Returns TRUE silently for a valid client structure
valid_client <- list(
    get_resource = function(...) "GET",
    post_data = function(...) "POST"
)
Rega:::.is_client(valid_client)
```

.is_scalar *Check if an object is a non-complex atomic scalar.*

Description

An atomic scalar is defined as an atomic vector of length exactly one, excluding complex numbers.

Usage

```
.is_scalar(x)
```

Arguments

x	An object to be tested.

Value

A logical value: TRUE if x is a non-complex atomic scalar, FALSE otherwise.

Examples

```
# Returns TRUE for simple, single values
Rega:::.is_scalar(5L)
Rega:::.is_scalar("hello")
Rega:::.is_scalar(TRUE)
# Returns FALSE for vectors, lists, or complex numbers
Rega:::.is_scalar(c(1, 2))
Rega:::.is_scalar(list(1))
Rega:::.is_scalar(1 + 1i)
```

.operation_params_to_args *Convert Operation Parameters to Function Arguments*

Description

This function transforms an operation's parameter definitions into a list of function arguments. Required parameters are marked as missing arguments.

Usage

```
.operation_params_to_args(op)
```

Arguments

op	List. An operation definition containing a `parameters` element, which is a list of parameter definitions. Each parameter should include a `name` and an optional `required` field.

Value

A named list representing function arguments. Names correspond to parameter names, with required parameters set to missing (`quote(expr = )`)

Examples

```
# Convert operation parameters to function arguments
opdefs <- extract_operation_definitions(extract_api())

Rega:::.operation_params_to_args(
    opdefs[["post__submissions__provisional_id__samples"]]
)
```

.runs_extra_validator *Validate Runs Metadata*

Description

Validates the metadata by checking the following:

- experiments for runs are specified
- samples for runs are specified
- file types for runs are specified
- files for runs are specified
- files for runs are unique
- experiments for runs are present in aliases
- samples for runs are present in aliases

Usage

```
.runs_extra_validator(meta, aliases)
```

Arguments

meta	A list containing parsed EGA metadata, where elements correspond to individual sheets.
aliases	A list of aliases, reference values used for validation.

Value

A validation object containing the results of the check.

Examples

```
meta <- list(runs = data.frame(
    experiment = c("ExperimentA", "ExperimentB", NA),
    alias = c("Sample1", "Sample2", "Sample1"),
    run_file_type = c("fastq", NA, "fastq"),
    files = I(list(c("File1", "File2"), c("File3"), NA))
))

aliases <- list(
    experiments = c("ExperimentA", "ExperimentB", "ExperimentC"),
    samples = c("Sample1", "Sample3")
)

validate::summary(Rega:::.runs_extra_validator(meta, aliases))
```

.studies_extra_validator

Validate Studies Metadata

Description

Validates the metadata by checking the following:

- study titles are unique
- study descriptions are unique
- study title contains number of words within required range
- study description contains number of sentences within required range

Usage

```
.studies_extra_validator(meta, aliases)
```

Arguments

meta	A list containing parsed EGA metadata, where elements correspond to individual sheets.
aliases	A list of aliases, reference values used for validation (not used in this function but included for consistency with other validators).

Value

A validation object containing the results of the check.

Examples

```
meta <- list(studies = data.frame(
    title = c("StudyA", "StudyA", "StudyB"),
    description = c("Desc1", "Desc2", NA)
))
Rega:::.studies_extra_validator(meta, list())
```

.submission_validator *Validate Submission Metadata*

Description

Validates the metadata by checking the following:

- submission title is specified

Usage

```
.submission_validator(meta, aliases)
```

Arguments

meta	A list containing parsed EGA metadata, where elements correspond to individual sheets.
aliases	A list of aliases, reference values used for validation (not used in this function but included for consistency with other validators).

Value

A validation object containing the results of the check.

Examples

```
meta <- list(submission = data.frame(title = c("Submission A", NA)))
Rega:::.submission_validator(meta, list())
```

.summarise_validation *Summarize Validation Results*

Description

Aggregates and summarizes the results of multiple validation checks, providing counts of failures, missing values, errors, and warnings.

Usage

```
.summarise_validation(validation_summary)
```

Arguments

validation_summary	A data frame of validation object summaries created using the `validate` package, one per row.

Value

Prints a summary message indicating the number of validation issues found, or confirms that validation passed successfully.

Examples

```
v <- validate::validator(x > 0)
data <- data.frame(x = c(1, -1, 3))
validation_result <- validate::summary(validate::confront(data, v))
Rega:::.summarise_validation(validation_result)
```

.validate_character_scalar *Validate that an object is a non-empty character scalar.*

Description

Checks if `x` is a character vector with a length of exactly one, is not `NA`, and contains at least one character (i.e., not an empty string, `""`). If the check fails, the function stops execution and reports an error using the variable name.

Usage

```
.validate_character_scalar(x, varname = NULL)
```

Arguments

x	An object to be tested.
varname	A character string specifying the name of the variable being checked, used in the error message. If NULL, the variable name is taken from the name of variable being tested. Defaults to NULL

Value

Returns TRUE if the validation succeeds, otherwise stops execution with an error.

Examples

```
# Returns TRUE silently
Rega:::.validate_character_scalar("valid_string", "input_arg")

# Throws an error for an empty string
tryCatch(
    Rega:::.validate_character_scalar("", "empty_arg"),
    error = function(e) message(e$message)
)
```

.validate_logical_scalar

Validate that an object is a non-missing logical scalar.

Description

Checks if x is a logical vector with a length of exactly one and is not NA. If the check fails, the function stops execution and reports an error using the variable name.

Usage

```
.validate_logical_scalar(x, varname = NULL)
```

Arguments

x	An object to be tested.
varname	A character string specifying the name of the variable being checked, used in the error message. If NULL, the variable name is taken from the name of variable being tested. Defaults to NULL

Value

Returns TRUE if the validation succeeds, otherwise stops execution with an error.

Examples

```
# Returns TRUE silently
Rega:::.validate_logical_scalar(TRUE, "input_check")

# Throws an error for a logical NA
tryCatch(
    Rega:::.validate_logical_scalar(NA, "na_arg"),
    error = function(e) message(e$message)
)
```

`add_required_str` *Mark Required Fields with a Prefix*

Description

Mark Required Fields with a Prefix

Usage

```
add_required_str(p, r, req_str = "* ")
```

Arguments

`p`	Character vector. All fields to be processed.
`r`	Character vector. Fields that are required.
`req_str`	Character. Prefix to mark required fields. Defaults to `"* "`.

Value

A character vector with required fields prefixed and ordered to appear before non-required fields.

Examples

```
# Mark required fields with a prefix
add_required_str(c("Name", "Id", "Age"), c("Id", "Name"))
```

aliases_formatter *Format Aliases from a Table*

Description

Format Aliases from a Table

Usage

```
aliases_formatter(tab, params)
```

Arguments

tab	Data frame. The input table where the first row contains column names.
params	List. Additional parameters for formatting. Takes a formatter params value from parser parameter yaml file. Currently unused.

Value

A named list where each name corresponds to a formatted column name, and values are non-NA elements of the respective column.

Examples

```
tab <- data.frame(Alias = c("name1", "name2", NA), Value = c(1, 2, 3))
aliases_formatter(tab, params = list())
```

api_function_factory *Generate an API Function from Operation and Specification*

Description

This function dynamically creates an API function based on a given operation definition and API specification. The generated function handles URL construction, parameter validation, request execution, and response parsing.

Usage

```
api_function_factory(
  op,
  api,
  verbosity = 0,
  bearer_token = NULL,
  token_url = .EGA_TOKEN_URL
)
```

Arguments

op	List. The API operation definition, including method, path, parameters, and request body schema.
api	List. The API specification, including host and global security definitions.
verbosity	Integer, optional, values 0-3. Indicates with which verbosity level should the requests `httr2::req_perform` be performed. Default: 0.
bearer_token	Character, optional. The API bearer token for authentication, will be included in the headers of the request. Defaults to `NULL`
token_url	Character, optional. Token endpoint URL from which to obtain the access token. If `bearer_token` is specified, it will take precedence. Defaults to .EGA_TOKEN_URL = `"https://idp.ega-archive.org/realms/EGA/protocol/openid-connect/token"`.

Value

A dynamically generated function that performs the specified API operation. The function accepts arguments corresponding to operation parameters and executes the request using `httr2`.

Examples

```
api <- extract_api()
opdefs <- extract_operation_definitions(api)

# Generate an API function for a specific operation
f <- api_function_factory(
    opdefs[["get__files"]], api,
    bearer_token = "my_key"
)

# Call the generated function with parameters (requires credentials)
try(
    result <- f(status = "value1", prefix = "value2")
)
```

---

api_name_to_label *Convert API Names to Prettified Labels*

Description

This function converts API-style names with underscores into human-readable labels by replacing underscores with spaces and applying title case.

Usage

```
api_name_to_label(x)
```

Arguments

x	Character vector. API field names to be converted.

Value

A character vector with API names converted to human-readable labels.

Examples

```
api_name_to_label(c("first_name", "last_name", "instument_model"))
```

---

column_table_formatter

Format a Column Table

Description

Format a Column Table

Usage

```
column_table_formatter(tab, params)
```

Arguments

tab	Data frame. The input table sumbission metadata file where the first row contains column names.
params	List. Additional parameters for formatting. Takes a formatter params value from parser parameter yaml file. Currently unused.

Value

A cleaned data frame with column names set from the first row, empty rows removed, and whitespace trimmed from all values.

Examples

```
df <- data.frame(
    ...1 = c("* Alias", "Sample1", "Sample2"),
    ...2 = c("* Phenotype", "wt", "ko"),
    ...3 = c("Description", NA, NA)
)

column_table_formatter(df, list())
```

create_client *Generate API Client Functions*

Description

This function creates a named list of functions for interacting with an API, based on its specification and operation definitions.

Usage

```
create_client(api, ...)
```

Arguments

api	List. The API specification, including operation definitions, host, and global settings.
...	List. List of additional arguments passed to api_function_factory.

Value

A named list of functions, where each function corresponds to an API operation. The function names match the operation IDs from the specification.

Examples

```
client <- create_client(
    extract_api(),
    bearer_token = "my_key", verbosity = 1
)

# Call an operation using the client (requires credentials)
try(
    result <- client$get__files(status = "value1", prefix = "value2")
)
```

default_parser *Parser for a Default EGA Excel Template*

Description

This function parses the extdata/ega_full_template_v3.xlsx using the bundled parser parameter file in extdata/default_parser_params.yaml to extract information for EGA submission into format that can be easily passed into EGA API endpoints.

Usage

```
default_parser(metadata_file, param_file = NULL)
```

Arguments

metadata_file	Character. Path to a default template xlsx file containing the submission metadata information.
param_file	Character. Path to a yaml file with parameters for parser. If NULL, uses the extdata/default_parser_params.yaml. Defaults to NULL.

Value

List of data frames or lists. Submission information parsed from the xlsx file.

Examples

```
default_parser(
    system.file("extdata/submission_example.xlsx", package = "Rega")
)
```

---

default_validator *Validator for Default Parser*

Description

Used to validate internal consistency of submission metadata parsed using the default parser. Performs several checks on EGA dataset for submission, ensuring that aliases for studies, experiments, samples, runs, analyses and datasets are are properly linked, as they will be replaced with provisional or accession IDs during submission process. Displays a success message if all validation passed or a summary message if validation failed. In addition it returns a data frame with validation details.

Usage

```
default_validator(meta, aliases = NULL)
```

Arguments

meta	List of data frames. Correspond to tables of EGA submission.
aliases	List of lists. Aliases that should present in the EGA tables. If NULL, the function will attempt to locate it in the meta parameter. Defaults to NULL

Value

Data frame. Validator object that includes all performed validations and their statistics (number of passes, fails and NAs, or whether errors or warnings were encountered during validation)

Examples

```
minimal_metadata <- list(
    aliases = list(
        studies = "Study1", experiments = "Experiment1",
        datasets = "Dataset1", samples = "Sample1", runs = "Run1",
        analyses = "Analysis1"
    ),
    files = tibble::tibble(
        file = "raw.fastq.gz", ega_file = list("raw.fastq.gz.c4gh")
    ),
    submission = tibble::tibble(title = "Submission"),
    studies = tibble::tibble(
        study = "Study1", title = "Study Title",
        description = "Study Description",
        study_type = "Whole Genome Sequencing"
    ),
    samples = tibble::tibble(
        alias = "Sample1", phenotype = "wild-type",
        biological_sex = "female", subject_id = "ID1"
    ),
    experiments = tibble::tibble(
        study = "Study1", experiment = "Experiment1",
        design_description = "Experiment Design",
        library_selection = "RANDOM", instrument_model_id = 1L,
        library_layout = "SINGLE", library_strategy = "WGS",
        library_source = "GENOMIC"
    ),
    runs = tibble::tibble(
        run = "Run1", experiment = "Experiment1", run_file_type = "srf",
        alias = "Sample1", files = list("raw.fastq.gz.c4gh")
    ),
    datasets = tibble::tibble(
        dataset = "Dataset1", title = "Dataset Title",
        description = "Dataset Description",
        policy_accession_id = "EGAP00000000001",
        dataset_types = list("Whole genome sequencing"),
        runs = list("Run1")
    )
)

default_validator(minimal_metadata)
```

---

delete_submission *Delete a Submission and Log Responses*

Description

Deletes a submission identified by its ID using the client and logs the response if a logfile is specified.

Usage

```
delete_submission(id, client = NULL, logfile = NULL, ...)
```

Arguments

id	A string representing the submission identifier (provisional ID).
client	An API client object with a delete method for submissions.
logfile	A string specifying the path to a log file. If NULL, no log is written. Defaults to NULL.
...	Additional arguments for future extensions (currently unused).

Value

A list containing the response for the submission deletion.

Examples

```
mock_client <- list(
    delete__submissions__provisional_id =
        function(id) list(status = "deleted")
)
delete_submission("5678901", mock_client)
```

---

delete_submission_contents

Delete Submission Contents and Log Responses

Description

Deletes all data associated with a submission ID using the client and logs the responses if a logfile is specified.

Usage

```
delete_submission_contents(id, client = NULL, logfile = NULL, ...)
```

Arguments

id	A string representing the submission identifier. Can be either an accession or provisional ID.
client	List of functions. EGA API client created by create_client function from EGA API schema with delete methods. If NULL, default client will be created by create_client(extract_api()). Defaults to NULL.
logfile	A string specifying the path to a log file. If NULL, no log is written. Defaults to NULL.
...	Additional arguments for future extensions (currently unused).

Value

A list of responses for the deletion of associated datasets, analyses, runs, experiments, samples, and studies.

Examples

```
mock_client <- list(
    "delete__submissions__provisional_id__datasets" =
        function(id) list(status = "deleted")
)
delete_submission_contents(5678901, client = mock_client)
```

DELIM_CONVERTERS *Default Column Converters for EGA Metadata*

Description

A named list of functions used to coerce delimited strings into specific data types during metadata processing.

Usage

```
DELIM_CONVERTERS
```

Format

A named list:

pubmed_ids Coerces values to integer.

Examples

```
df <- list(pub_sheet = data.frame(
    pubmed_ids = c("  123456,  789", NA, "1001  ")
))

process_delimited_column(
    df, "pubmed_ids", separator = ",", converters = DELIM_CONVERTERS
)
```

ega_oauth *Set The OAUTH With EGA Username And Password*

Description

ega_oauth implements the EGA OAuth resource owner password flow, as defined by Section 4.3 of RFC 6749. It allows the user to supply their password once, exchanging it for an access token that can be cached locally. Please avoid entering the password directly when calling this function as it will be captured by .Rhistory.

Usage

```
ega_oauth(
  req,
  username = .get_ega_username(),
  password = .get_ega_password(),
  token_url = .EGA_TOKEN_URL
)
```

Arguments

req	A httr2 request.
username	Character. EGA User name. Defaults to the value returned by .get_ega_username().
password	Character. EGA user Password. Defaults to the value returned by .get_ega_password().
token_url	Character. The URL for the EGA token endpoint. Defaults to .EGA_TOKEN_URL = "https://idp.ega-archive.org/realms/EGA/protocol/openid-connect/token".

Value

returns a modified HTTP request that will use OAuth

Examples

```
req <- httr2::request("https://example.com/")

# Request OAuth with default credentials
try(oauth_req <- ega_oauth(req))

# Request OAuth with custom credentials
oauth_req <- ega_oauth(req, username = "user", password = "pass")
```

ega_token *Retrieve EGA API Bearer Token*

Description

This function retrieves an API token from the European Genome-Phenome Archive (EGA) using user credentials.

Usage

```
ega_token(
  username = .get_ega_username(),
  password = .get_ega_password(),
  token_url = .EGA_TOKEN_URL
)
```

Arguments

username	Character. The username for EGA authentication. Defaults to the value returned by .get_ega_username().
password	Character. The password for EGA authentication. Defaults to the value returned by .get_ega_password().
token_url	Character. The URL for the EGA token endpoint. Defaults to the standard EGA token URL if not provided. Defaults to .EGA_TOKEN_URL = "https://idp.ega-archive.org/realm

Value

A list containing the token details if successful. Actual token value can be retrieved by token$access_token

Examples

```
try(
    ega_token(username = "my_username", password = "my_password")
)

try(
    ega_token(token_url = "https://www.example.com")
)
```

extract_api *Extract API Specification and Host Details*

Description

This function parses an API specification file (JSON or YAML) and extracts relevant details.

Usage

```
extract_api(spec_file = NULL, host = NULL)
```

Arguments

spec_file	Character. Optional.Path to the API specification file in JSON or YAML format. If NULL default `extdata/ega_api_deref.yaml` is used. Defaults to `NULL`
host	Character. Optional. The API host URL. If not supplied, it will be inferred from the specification file's `servers` element. Defaults to `NULL`

Value

A list containing the parsed API specification, including the `host` and `basePath` elements. If the specification file lacks required elements, appropriate warnings or errors are raised.

Examples

```
# Extract API details from a default YAML specification file
api <- extract_api()

# Extract API details with a custom host
api <- extract_api(host = "https://api.example.com")
```

extract_operation_definitions *Extract API Operation Definitions*

Description

This function extracts operation definitions from an API specification, including HTTP methods, paths, parameters, request bodies, and responses.

Usage

```
extract_operation_definitions(api)
```

Arguments

api	List. Parsed API specification, generated from a JSON or YAML file. Must include a paths element with API endpoint definitions.

Value

A named list of operations, where each name corresponds to an operation ID. If operation Id is not found in the specification, unique one will be created. Each operation contains:

- method: HTTP method (e.g., GET, POST).
- path: Endpoint path.
- parameters: List of operation parameters.
- requestBody: Details of the request body (if any).
- responses: Possible responses for the operation.
- security: Security requirements for the operation.

Examples

```
# Extract operation definitions from a parsed API specification
opdefs <- extract_operation_definitions(extract_api())
opdefs[["post__submissions"]]
```

---

extract_resource_name *Extract Resource Name from API Response URL*

Description

Extracts the specific resource identifier (e.g., "users", "datasets") from the path of an httr2 response object by parsing the segment immediately following /api/.

Usage

```
extract_resource_name(resp)
```

Arguments

resp	An httr2_response object.

Value

A character string containing the resource name.

Examples

```
resp <- httr2::response(
    method = "GET",
    url = "https://www.example.com/api/files"
)
extract_resource_name(resp)
```

`fetch_files` *Fetch file information from EGA inbox*

Description

Query the remote client for requested file prefix, test whether a file is found for every element of `file_list` and return the server response.

Usage

```
fetch_files(file_list, client = NULL)
```

Arguments

`file_list`	A character vector or list of file prefixes to check.
`client`	List of functions. EGA API client created by `create_client` function from EGA API schema. If `NULL`, default client will be created by `create_client(extract_api())`. Defaults to `NULL`.

Value

Data frame. Parsed response from client API for requested files.

Examples

```
mock_client <- list(
    get__files = function(prefix = NULL) {
        data.frame(provisional_id = 12345, ega_relative_path = prefix)
      }
)
fetch_files(c("file_a", "file_b"), mock_client)
```

file_formatter *Format File Table with EGA File Paths*

Description

Format File Table with EGA File Paths

Usage

```
file_formatter(tab, params)
```

Arguments

tab	Data frame. The input table containing file information. Columns file, ega_inbox_relative_path need to be present in the data.
params	List. Additional parameters for formatting. Takes a formatter params value from parser parameter yaml file. Includes crypt_ext for encryption file extensions and prepend_slash to control path prefix.

Value

A formatted data frame with cleaned column names, and updated ega_file paths based on file and relative path information.

Examples

```
params <- list(prefix = "", crypt_ext = "c4gh", prepend_slash = FALSE)

# Dummy data, first row will be moved to column names
tab <- data.frame(
    x1 = c("file", "value1", "value2"),
    x2 = c("ega_inbox_relative_path", NA, "proj1")
)

file_formatter(tab, params)
```

filter_id_fields *Filter Out ID Fields from a Character Vector*

Description

This function filters out elements from a character vector that match a specified regular expression pattern, removing ID fields.

Usage

```
filter_id_fields(x, pattern = NULL)
```

Arguments

x	Character vector. The input vector to filter.
pattern	Character. Optional. A regular expression pattern for matching ID fields to exclude. Defaults to "(?<!policy_)accession_id|provisional_id", which removes accession IDs and provisional IDs, but not policy accession ID.

Value

A character vector with elements matching the pattern removed.

Examples

```
# Filter out default ID fields
fields <- c("accession_id", "policy_accession_id", "name", "provisional_id")
filter_id_fields(fields)

# Filter with a custom pattern
filter_id_fields(fields, pattern = "_id")
```

---

finalise_submission *Finalise an EGA submission*

Description

Submits the finalisation request for a submission identified by either an accession or provisional ID. Validates the release date and sends optional dataset changelogs.

Usage

```
finalise_submission(
  id,
  release_date,
  dataset_changelogs = data.frame(),
  client = NULL,
  logfile = NULL,
  ...
)
```

Arguments

id	Character scalar. The submission accession or provisional ID.
release_date	Character scalar. Expected release date in YYYY-MM-DD format.
dataset_changelogs	Data frame. Optional changelog metadata for associated datasets. If specified, the requred columns are dataset and message. Defaults to empty data.frame.
client	List of functions. EGA API client created by create_client function from EGA API schema. If NULL, default client will be created by create_client(extract_api()). Defaults to NULL.
logfile	Character. Path of log file to log the httr2 responses from individual operations or NULL. Defaults to NULL.
...	List. Additional arguments to the function.

Value

The API response object from the finalisation request.

Examples

```
# Requires credentials
try(
    finalise_submission("123456", "2025-12-31")
)
```

---

first_row_to_colnames *Use First Row as Column Names for a Data Frame*

Description

Use First Row as Column Names for a Data Frame

Usage

```
first_row_to_colnames(df, to_api = TRUE)
```

Arguments

df	Data frame. The input data frame whose first row will become column names.
to_api	Logical. Whether to convert labels to API-style names using label_to_api_name(). Defaults to TRUE.

Value

A data frame with updated column names and the first row removed.

Examples

```
df <- data.frame(id = c("A B", "C D_"), value = c("* E F", "GH"))

first_row_to_colnames(df)
first_row_to_colnames(df, to_api = FALSE)
```

`fold_column` *Fold Columns with a Common Prefix into a Single Column Nested as List*

Description

If `NA` values are present in any of the columns to be nested, they will be removed. If the column is not present it will be added with `NA` as a single value.

Usage

```
fold_column(tab, column_prefix, new_name)
```

Arguments

`tab`	Data frame. The input table with columns to fold.
`column_prefix`	Character. The prefix of columns to nest into a single column represented as list.
`new_name`	Character. The name of the new folded column.

Value

A data frame with the specified columns nested into a single column.

Examples

```
tab <- data.frame(id = c(1, 2), name.1 = c("A1", NA), name.2 = c("B1", "B2"))
fold_column(tab, "name", "folded_column")
```

format_chromosomes *Format Chromosome Metadata*

Description

Formats and processes chromosome-related metadata from an input object by applying chromosome group lookups or splitting chromosome strings from the EGA enums.

Usage

```
format_chromosomes(metadata)
```

Arguments

metadata	List. A list of data frames representing metadata sheets, containing analyses. Each row in analyses can have entry in chromosomes or chromosome_groups column.

Value

A list of formatted chromosome data extracted or computed from the input metadata.

Examples

```
# Mock metadata data frame
metadata <- list(
    analyses = data.frame(
        chromosomes = I(list(
            NA,
            list("group1--1--chr1--name1", "group2--3--chr3--name3"),
            "group1--2--chr2--name2"
        )),
        chromosome_groups = c("group1", NA, "group3"),
        stringsAsFactors = FALSE
    ),
    select_input_data = list(
        chromosomes = c("group1--1--chr1--name1", "group1--2--chr2--name2")
    )
)

format_chromosomes(metadata)
```

get_chr_group *Retrieve Chromosome Belonging to a Group*

Description

Retrieve Chromosome Belonging to a Group

Usage

```
get_chr_group(group_id, chr_enum, sep = "--")
```

Arguments

group_id	Character. The group ID to filter by.
chr_enum	Character vector. Chromosome enumeration data, where each element is a string containing fields separated by sep. Enum data is created by parse_enum and get_enum functions. See vignette for more details.
sep	Character. Field separator in a string. Defaults to "--"

Value

An integer vector of chromosome IDs corresponding to the specified group ID.

Examples

```
get_chr_group(
    "group1", c("group1--1--chr1--name1", "group2--2--chr2--name2")
)
```

get_entry_by_title *Retrieve EGA entries by title*

Description

Searches for entries across specified EGA metadata types that match a given title string. Returns a list of data frames for each type.

Usage

```
get_entry_by_title(title, type = NULL, client = NULL, logfile = NULL, ...)
```

Arguments

title	Character scalar. The title or substring to search for.
type	Character vector. One or more metadata types ("submissions", "studies", "samples", "experiments", "runs", "analyses" and "datasets"). If NULL, searches all valid types.
client	List of functions. EGA API client created by create_client function from EGA API schema. If NULL, default client will be created by create_client(extract_api()). Defaults to NULL.
logfile	Character. Path of log file to log the httr2 responses from individual operations or NULL. Defaults to NULL.
...	List. Additional arguments to the function.

Value

A named list of data frames containing entries matching the title.

Examples

```
# Requires credentials
try(
    get_entry_by_title("My Study", type = "studies")
)
```

---

get_enum *Retrieve Enum Values from an API*

Description

This function retrieves the values of a specified enum from an API by invoking the corresponding client function.

Usage

```
get_enum(client, enum_name, enum_prefix = "get__enums__")
```

Arguments

client	List. The API client, typically generated by create_client, containing functions for API operations.
enum_name	Character. The name of the enum to retrieve.
enum_prefix	Character. Optional. The prefix used in the client for enum functions. Defaults to "get__enums__".

Value

The values of the specified enum.

Examples

```
# Create API client with mock api_key
client <- create_client(extract_api(), token_url = "ABCD")

# Retrieve enum values from the API client (requires credentials to work)
try(
    platform_models <- get_enum(client, enum_name = "platform_models")
)
```

`get_formatter` *Retrieve a Formatter Function by Type of Submission Metadata Table*

Description

Retrieve a Formatter Function by Type of Submission Metadata Table

Usage

```
get_formatter(x, params)
```

Arguments

`x`	Character. The name of the submission metadata table/sheet.
`params`	List. A list containing a `formatter` element from parser parameter yaml file.

Value

The formatter function corresponding to the specified table.

Examples

```
# Load formatter params
params <- yaml::read_yaml(system.file(
    "extdata/default_parser_params.yaml",
    package = "Rega"
))

# Dummy data, first row will be moved to column names
tab <- data.frame(
    x1 = c("file", "value1", "value2"),
    x2 = c("ega_inbox_relative_path", NA, "proj1")
)

ff <- get_formatter("files", params)
ff_params <- get_formatter_params("files", params)

ff(tab, ff_params)
```

`get_formatter_params` *Retrieve Formatter Parameters by Name*

Description

Retrieve Formatter Parameters by Name

Usage

```
get_formatter_params(x, params)
```

Arguments

`x`	Character. The name of the formatter for which to retrieve parameters.
`params`	List. A list containing a `formatter` element from parser parameter yaml file.

Value

A list of parameters for the specified formatter.

Examples

```
# Load formatter params
params <- yaml::read_yaml(system.file(
    "extdata/default_parser_params.yaml",
    package = "Rega"
))

# Dummy data, first row will be moved to column names
tab <- data.frame(
    x1 = c("file", "value1", "value2"),
    x2 = c("ega_inbox_relative_path", NA, "proj1")
)

ff <- get_formatter("files", params)
ff_params <- get_formatter_params("files", params)

ff(tab, ff_params)
```

get_operation_schema *Retrieve the Schema for an API Operation*

Description

Retrieve the Schema for an API Operation

Usage

```
get_operation_schema(op)
```

Arguments

op	List. The API operation definition containing a `requestBody` element with content and schema details.

Value

The schema for the operation's JSON request body, or `NULL` if no schema is defined.

Examples

```
# Get operations from API
opdefs <- extract_operation_definitions(extract_api())

# Retrieve the schema for a specific operation
schema <- get_operation_schema(opdefs[["post__submissions"]])
```

get_or_post *Retrieve or Submit Data to an EGA API Endpoint*

Description

This function retrieves existing data from an API or submits new data if it does not exist, with optional error handling and retrieval options.

- If no data is present in the database, supplied data will be inserted.
- If there is data already present in the database and the number of records don't match, error will raised.
- If the number of records match and `retrieve` is set to TRUE data will be retrieved from database and nothing will be inserted. If `retrieve` is set to FALSE, error will be raised.

Usage

```
get_or_post(
  submission_id,
  data,
  client,
  endpoint,
  retrieve = FALSE,
  id_type = "provisional"
)
```

Arguments

submission_id	An integer representing the submission provisional ID.
data	A data frame to be submitted.
client	An API client object with get and post methods.
endpoint	A string specifying the EGA API endpoint. The endpoint will be a submission type endpoint identified with provisional ID.
retrieve	A logical flag indicating whether to retrieve data if it already exists. Defaults to FALSE.
id_type	A string specifying type of EGA id. One of 'provisional' or 'accession'. Defaults to provisional.

Value

A data frame containing the response from the API.

Examples

```
# Create mock client for API endpoint
mock_client <- list(
    get__submissions__provisional_id__endpoint = function(id) {
        message("Mock GET request")
        # Simulate an empty response (no existing data)
        return(NULL)
    },
    post__submissions__provisional_id__endpoint = function(id, body) {
        message("Mock POST request")
        message(body) # Simulate returning submitted data
    }
)

# Create mock data to test the function
test_data <- data.frame(id = 1:3, value = c("A", "B", "C"))

# Test the function with mock data and client
result <- get_or_post(
    submission_id = 12345,
    data = test_data,
    client = mock_client,
```

```
    endpoint = "endpoint",
    retrieve = FALSE
)
```

`get_properties` *Extract and Format Properties from a Schema*

Description

This function extracts property names from a schema, optionally filters out ID fields, and applies formatting such as marking required fields and prettifying the labels.

Usage

```
get_properties(schema, filter_ids = TRUE)
```

Arguments

`schema`	List. The schema containing `properties` and `required` fields.
`filter_ids`	Logical. Whether to filter out ID fields from the properties. Defaults to `TRUE`.

Value

A character vector of formatted property names and indications of required fields.

Examples

```
schemas <- get_schemas(extract_api())

# Extract and format properties from a schema
get_properties(schemas[[5]])

# Extract properties without filtering ID fields
get_properties(schemas[[6]], filter_ids = FALSE)
```

get_schemas *Retrieve Request schemas from an API Specification*

Description

This function extracts and returns schemas related to requests from the API specification.

Usage

```
get_schemas(api)
```

Arguments

api	List. The API specification, typically containing a components element with nested schemas.

Value

A list of schemas whose names contain "Request", filtered from the schemas element of the API specification.

Examples

```
# Extract request schemas from an API specification
request_schemas <- get_schemas(extract_api())
```

get_sentence_number *Count the number of sentences in text*

Description

Count how many sentences are in a character string, based on terminal punctuation marks ., !, or ? following an alphanumeric character. If there is no punctuation character at the end, it will still count it as another sentence.

Usage

```
get_sentence_number(text)
```

Arguments

text	A character vector.

Value

An integer scalar giving the number of sentences in text.

Examples

```
get_sentence_number("First sentence. Second sentence? Third!")
```

`get_submission` *Retrieve Submission Data and Log Responses*

Description

Retrieves data associated with a submission ID using the client and logs the responses if a logfile is specified.

Usage

```
get_submission(id, client = NULL, logfile = NULL, ...)
```

Arguments

`id`	A string representing the submission identifier. Can be either an accession or provisional ID.
`client`	List of functions. EGA API client created by `create_client` function from EGA API schema with `get` methods. If `NULL`, default client will be created by `create_client(extract_api())`. Defaults to `NULL`.
`logfile`	A string specifying the path to a log file. If `NULL`, no log is written. Defaults to `NULL`.
`...`	Additional arguments for future extensions (currently unused).

Value

A list of responses including submission data and associated datasets, analyses, runs, experiments, samples, and studies.

Examples

```
mock_client <- list(
    "get__submissions__accession_id" = function(id) list(data = id),
    "get__submissions__accession_id__datasets" =
        function(id) list(datasets = id)
)
get_submission("EGAB12345678901", mock_client)
```

get_word_number *Count the number of words in text*

Description

Compute the number of words in each element of a character vector using non-word separators.

Usage

```
get_word_number(text)
```

Arguments

text	A character vector with text.

Value

An integer vector giving the number of words per element of text.

Examples

```
get_word_number(c("one two", "three four five"))
```

has_linked_sheets *Check for Linked Sheets in Metadata*

Description

Determines whether a specified sheet is present and contains at least one non-NA value in the provided metadata.

Usage

```
has_linked_sheets(metadata, colname)
```

Arguments

metadata	A list of data frame objects to check.
colname	A string specifying the name of the column to look for.

Value

A logical vector indicating whether each element of metadata contains the specified column with at least one non-NA value.

Examples

```
metadata <- list(
    sheet1 = list(sheet_name = c(1, NA)),
    sheet2 = list(other_name = NA)
)
has_linked_sheets(metadata, "sheet_name")
```

`is_accession` *Check if a String is a Valid Accession Identifier.*

Description

Verifies whether the input string matches the format of a valid accession identifier based on a specified schema.

Usage

```
is_accession(x, schema = NULL)
```

Arguments

`x`	A character vector to be tested for validity as accessions.
`schema`	A character string specifying the schema. Valid options include "study", "studies", "sample", "samples", "experiment", "experiments", "analysis", "analyses", "run", "runs", "policy", "DAC", "dataset", "datasets", "submission" and `NULL`. `NULL` will check against any of the schemas. Defaults to `NULL`.

Value

A logical vector indicating which values are accession IDs.

Examples

```
is_accession("EGAB00000000001", "submission") # TRUE
is_accession("EGA12345678901", "sample") # FALSE
```

is_provisional *Check whether IDs is a provisional ID.*

Description

Determine if input values match the format of provisional IDs, either as whole-number numerics or as character strings of at least two digits without leading zeros.

Usage

```
is_provisional(x)
```

Arguments

x	A numeric or character vector of candidate provisional IDs.

Value

A logical vector indicating which values are provisional IDs.

Examples

```
is_provisional(c(10, 11, 3.5, 9))
```

is_valid_http_method *Validate HTTP Method*

Description

Checks whether a given HTTP method is valid based on a predefined list of accepted methods (matches on lowercase).

Usage

```
is_valid_http_method(m)
```

Arguments

m	A string representing the HTTP method to validate.

Value

A logical value: TRUE if m is a valid HTTP method, otherwise FALSE.

Examples

```
is_valid_http_method("GET") # TRUE
is_valid_http_method("get") # TRUE
is_valid_http_method("DELETE") # TRUE
is_valid_http_method("foo") # FALSE
is_valid_http_method(NULL) # FALSE
```

`label_to_api_name` *Convert Prettified Labels to API Names*

Description

Convert Prettified Labels to API Names

Usage

```
label_to_api_name(x, req_str = "* ")
```

Arguments

`x`	Character vector. Prettified labels to convert.
`req_str`	Character. Optional prefix to remove from labels. Defaults to `"* "`.

Value

A character vector with labels converted to API-style names.

Examples

```
label_to_api_name(c("* First Name", "Last Name"))
label_to_api_name(c("# Instrument Model", "# Fragment SD"), req_str = "# ")
```

`link_sheet` *Link Data Between Metadata Sheets*

Description

Data frames representing metadata sheets that contain column names corresponding to `sheet_name` containing an ID reference (to a first column in `sheet_name`) will be replaced with the rest of the values nested as a list.

Usage

```
link_sheet(metadata, sheet_name)
```

Arguments

metadata	List. A list of data frames representing metadata sheets.
sheet_name	Character. The name of the sheet to link with other sheets.

Value

A list of updated metadata with the specified values replaced based on referenced values present in sheet_name.

Examples

```
# Link data from a specific sheet to other sheets in metadata
metadata <- list(
    sheet1 = data.frame(id = c(1, 2), linked_sheet = c("A", "B")),
    linked_sheet = data.frame(id = c("A", "B"), value = c(10, 20))
)
updated_metadata <- link_sheet(metadata, "linked_sheet")
```

---

lut_add *Add a Column to a Data Frame Based on Lookup Table*

Description

This function adds a new column to a data frame by mapping values from an existing column through a lookup table.

Usage

```
lut_add(df, to, from, lut)
```

Arguments

df	A data frame to which the new column will be added.
to	A string specifying the name of the new column.
from	A string specifying the name of the column to map values from.
lut	A named list or vector serving as the lookup table.

Value

The input data frame with the added column.

Examples

```
df <- data.frame(id = c("A", "B", "C"), stringsAsFactors = FALSE)
lut <- list(A = 1, B = 2, C = 3)
lut_add(df, "value", "id", lut)
```

merge_linked_sheet *Merge Linked Sheets with Source Data*

Description

Merges a target column with a source column in a linked sheet's data, processing it into a format suitable for JSON parsing. Includes API-specific adjustments for certain data.

Usage

```
merge_linked_sheet(target, source, dat, sheet)
```

Arguments

target	A vector containing the target values.
source	A string specifying the source column to merge on.
dat	A data frame representing the data to be linked.
sheet	A string specifying the name of the sheet, used for API-specific processing.

Value

A data frame containing the merged data, or an empty list if the target is entirely NA.

Examples

```
target <- c(1, 2, 3)
source <- "id"
dat <- data.frame(id = c(1, 2, 3), value = c("A", "B", "C"))
merge_linked_sheet(target, source, dat, "collaborators")
```

multi_lut_add *Add Multiple Lookup-Based Columns to a Data Frame*

Description

Adds multiple columns to a data frame by applying multiple lookup tables, each defined by a set of arguments specifying the new column, the source column, and the lookup table.

Usage

```
multi_lut_add(df, ...)
```

Arguments

df	A data frame to which new columns will be added.
...	A series of lists, each containing three elements: the name of the new column (to), the name of the source column (from), and the lookup table (lut). See lut_add for details.

Value

The input data frame with the added columns.

Examples

```
df <- data.frame(id = c("A", "B", "C"), stringsAsFactors = FALSE)
lut1 <- list(A = 1, B = 2, C = 3)
lut2 <- list(A = "x", B = "y", C = "z")
multi_lut_add(df, list("value1", "id", lut1), list("value2", "id", lut2))
```

---

na_to_empty_list *Convert NA Values to Empty Lists*

Description

Replaces NA values in a list with empty lists, preserving the original structure of the list. Doesn't work on nested lists.

Usage

```
na_to_empty_list(l)
```

Arguments

l	A list containing elements that may include NA values.

Value

A list where any NA values have been replaced with empty lists.

Examples

```
input_list <- list(1, NA, "text", NA)
na_to_empty_list(input_list)
```

`new_submission` *Submit New Data to EGA*

Description

This function creates a new submission and associates all specified data with it. Following data has to be present in the request data object: submission studies, experiments, samples, runs, analyses, datasets. The files associated with the submission must be present in the EGA Inbox and they are fetched and matched according to Inbox path. In case the submission is interrupted or fails, all the information entered into EGA database is rolled back apart from the submission itself. If the workflow successfully creates a submission, but fails in the following steps, the returned submission ID can be used as a parameter to the workflow to continue entering data into existing submission. If logfile is specified, the responses from successfully executed steps (even if the error occurs), will be saved.

Usage

```
new_submission(
  dat,
  client = NULL,
  logfile = NULL,
  submission_id = NULL,
  retrieve = FALSE,
  ...
)
```

Arguments

`dat`	List of data frames. Parsed submission metadata containing correctly formatted and linked information for submission
`client`	List of functions. EGA API client created by `create_client` function from EGA API schema. If `NULL`, default client will be created by `create_client(extract_api())`. Defaults to `NULL`.
`logfile`	Character. Path of log file to log the `httr2` responses from individual operations or `NULL`. Defaults to `NULL`.
`submission_id`	Integer.
`retrieve`	Logical.
`...`	List. Additional arguments to the function.

Value

List of data frames. Parsed response objects from httr2 requests

Examples

```
minimal_metadata <- list(
    aliases = list(
        studies = "Study1", experiments = "Experiment1",
        datasets = "Dataset1", samples = "Sample1", runs = "Run1",
        analyses = "Analysis1"
    ),
    files = tibble::tibble(
        file = "raw.fastq.gz", ega_file = list("raw.fastq.gz.c4gh")
    ),
    submission = tibble::tibble(title = "Submission"),
    studies = tibble::tibble(
        study = "Study1", title = "Study Title",
        description = "Study Description",
        study_type = "Whole Genome Sequencing"
    ),
    samples = tibble::tibble(
        alias = "Sample1", phenotype = "wild-type",
        biological_sex = "female", subject_id = "ID1"
    ),
    experiments = tibble::tibble(
        study = "Study1", experiment = "Experiment1",
        design_description = "Experiment Design",
        library_selection = "RANDOM", instrument_model_id = 1L,
        library_layout = "SINGLE", library_strategy = "WGS",
        library_source = "GENOMIC"
    ),
    runs = tibble::tibble(
        run = "Run1", experiment = "Experiment1", run_file_type = "srf",
        alias = "Sample1", files = list("raw.fastq.gz.c4gh")
    ),
    datasets = tibble::tibble(
        dataset = "Dataset1", title = "Dataset Title",
        description = "Dataset Description",
        policy_accession_id = "EGAP00000000001",
        dataset_types = list("Whole genome sequencing"),
        runs = list("Run1")
    )
)

ega <- create_client(extract_api(), verbosity = 0)

# Requires credentials
try(
    new_submission(minimal_metadata, ega)
)
```

---

parse_ega_body *Parse The Information From EGA httr2 Response Object.*

Description

Parses the body of a body of `httr2` response object from the EGA API, handling JSON and plain text content, and formats it into a tibble for further processing.

Usage

```
parse_ega_body(resp)
```

Arguments

`resp`	An HTTP response object from the EGA API.

Value

A tibble containing the parsed and formatted response data. If the response is plain text without a JSON-like structure, a one-column tibble is returned with the raw content.

Examples

```
# Example with JSON response
json_resp <- httr2::response(
    method = "GET",
    url = "https://www.example.com/api/files",
    status = 200,
    headers = list("content-type" = "application/json"),
    body = charToRaw('[{"id": 1, "name": "test"}]')
)
parse_ega_body(json_resp)

# Example with plain text response
text_resp <- httr2::response(
    method = "POST",
    url = "https://www.example.com/api/submissions",
    status = 200,
    headers = list("content-type" = "text/plain"),
    body = charToRaw("Sample response text")
)
parse_ega_body(text_resp)
```

`parse_enum` *Parse Enum into a Formatted String*

Description

This function parses an enum, represented as a data frame or character vector, into a formatted string for display or further use.

Usage

```
parse_enum(enum, sep = "--")
```

Arguments

enum	Data frame or character vector. The enum to parse. If a data frame, its rows are concatenated into strings. If a character vector, its elements are joined with newlines.
sep	Character. If enum has multiple fields, they will be pasted into a single string using this separator. Defaults to `--`

Value

A single string representing the parsed enum. Rows are joined by newlines and multiple enum fields are joined by sep.

Examples

```
# Parse an enum as a data frame
df_enum <- data.frame(key = c("A", "B"), value = c("1", "2"))
parse_enum(df_enum)

# Parse an enum as a character vector
vec_enum <- c("A", "B", "C")
parse_enum(vec_enum)
```

---

parse_json_body *Parse and Standardize JSON Response Body*

Description

Extracts the JSON body from a response and ensures the output is structured as a list of objects. Named lists (single records) are wrapped in a parent list to maintain consistency for downstream unnesting.

Usage

```
parse_json_body(resp)
```

Arguments

resp	An `httr2_response` object containing JSON content.

Value

A list of lists, where each inner list represents a record.

Examples

```
json_resp <- httr2::response(
    method = "GET",
    url = "https://www.example.com/api/files",
    status = 200,
    headers = list("content-type" = "application/json"),
    body = charToRaw('[{"id": 1, "name": "test"}]')
)
parse_json_body(json_resp)
```

`parse_text_body` *Parse Plain Text or JSON-like Response Body*

Description

Processes a text response by either parsing it as JSON (if structured with curly braces or square brackets) or returning it as a list. Null JSON elements are converted to empty lists to facilitate unnesting.

Usage

```
parse_text_body(resp)
```

Arguments

`resp`	An `httr2_response` object with "text/plain" content.

Value

A list of parsed data or a tibble if the content is raw text.

Examples

```
text_resp <- httr2::response(
    method = "POST",
    url = "https://www.example.com/api/submissions",
    status = 200,
    headers = list("content-type" = "text/plain"),
    body = charToRaw("Sample response text")
)
parse_text_body(text_resp)
```

process_chromosomes *Process a Vector or List of Chromosome Data*

Description

This function processes chromosome data by extracting unique chromosome IDs and labels or retrieving chromosome group information from a lookup where applicable.

Usage

```
process_chromosomes(chr_data, select_input_data)
```

Arguments

chr_data	A list containing chromosome-related information. Expected to have items chromosomes (scalar, vector or list) and/or chromosome_groups (scalar).
select_input_data	A list containing look-up data for chromosomes.

Value

A data frame with chromosome id and label if chromosomes are present. If only chromosome groups exist, returns the result of lookup against the select_input_data with get_chr_group(). If neither are present, returns an empty list.

Examples

```
select_input_data <- list(
    chromosomes = c("group1--1--chr1--name1", "group1--2--chr2--name2")
)

chr_data_1 <- list(
    chromosomes = list("group1--1--chr1--name1", "group2--3--chr3--name3"),
    chromosome_groups = NA_character_
)
process_chromosomes(chr_data_1, select_input_data)

chr_data_2 <- list(
    chromosomes = NA,
    chromosome_groups = "group1"
)

process_chromosomes(chr_data_2, select_input_data)
```

process_delimited_column *Process Delimited Columns in Metadata*

Description

The specified column name is searched for across all the data frames. If the column is `pubmed_ids`, values are converted to integers.

Usage

```
process_delimited_column(
  metadata,
  column_name,
  separator,
  converters = DELIM_CONVERTERS
)
```

Arguments

`metadata`	List. A list of data frames representing metadata sheets.
`column_name`	Character. The name of the column to process.
`separator`	Character. The delimiter used to split column values.
`converters`	Named list of functions. Specifies how to coerce columns with delimited strings into specific types or values. Defaults to `DELIM_CONVERTERS`

Value

A list of updated metadata with the specified column split into lists based on the delimiter and trimmed.

Examples

```
metadata <- list(
    sheet1 = data.frame(pubmed_ids = c("123; 456", "130; 789; 102", NA))
)

process_delimited_column(metadata, "pubmed_ids", ";")
```

rollback_submission *Rollback Submission Endpoints and Log Responses*

Description

Rolls back specified endpoints for a submission identified by its accession ID using the client and logs the responses if a logfile is specified.

Usage

```
rollback_submission(id, endpoints, client = NULL, logfile = NULL, ...)
```

Arguments

id	A string representing the submission identifier. Must be an accession ID.
endpoints	A character vector of endpoint names to rollback.
client	List of functions. EGA API client created by `create_client` function from EGA API schema with `put` methods and rollback operations. If `NULL`, default client will be created by `create_client(extract_api())`. Defaults to `NULL`.
logfile	A string specifying the path to a log file. If `NULL`, no log is written. Defaults to `NULL`.
...	Additional arguments for future extensions (currently unused).

Value

A list of responses from the rollback operations for each endpoint.

Examples

```
mock_client <- list(
    "put__submissions__accession_id__datasets__rollback" =
        function(id) list(status = "rolled back")
)
rollback_submission("EGAB00000000001", list("datasets"), mock_client)
```

row_table_formatter *Format a Row Table*

Description

Format a Row Table

Usage

```
row_table_formatter(tab, params)
```

Arguments

tab	Data frame. The input table from a submission metadata file
params	List. Additional parameters for formatting. Takes a formatter params value from parser parameter yaml file.

Value

A cleaned and formatted tibble with correctly organized rows and columns, whitespace trimmed, and folding applied to specified columns.

Examples

```
# Formatter parameters
params <- list(fold = "extra_attributes")

# Sample data frame
df <- data.frame(
    ...1 = c("* Study", "* Title", "Extra Attributes", "Extra Attributes"),
    ...2 = c("Study1", "Title1", "A", "B"),
    ...3 = c("* Study", "* Title", "Extra Attributes", NA),
    ...4 = c("Study2", "Title2", "C", NA)
)

row_table_formatter(df, params)
```

---

samples_in_db *Check if sample aliases exist in the EGA database*

Description

Validates uniqueness of sample aliases by comparing input against existing records in the EGA database. Throws an error if duplicates are found and retrieval is not enabled.

Usage

```
samples_in_db(samples, client = NULL, retrieve = FALSE)
```

Arguments

samples	Character vector of sample aliases to check.
client	An EGA API client object. If NULL, one is created.
retrieve	Logical scalar. If TRUE, exists without error even if samples are found in the database.

Value

Logical TRUE if validation passes.

Examples

```
my_client <- list(
    get__samples = function(prefix = NULL) {
        data.frame(alias = c("unique_sample1", "unique_sample_2"))
      }
)

samples_in_db(c("sample1", "sample2"), client = my_client, retrieve = FALSE)
```

---

save_log *Save API Responses to a Log File*

Description

This function saves a list of API responses to a specified log file in YAML format.

Usage

```
save_log(responses, logfile)
```

Arguments

responses	A list of responses to be saved.
logfile	A string specifying the path to the log file. If NULL, no file is written.

Value

Invisibly returns NULL

Examples

```
responses <- list(status = "success", data = list(a = 1, b = "text"))
save_log(responses, logfile = NULL)
```

step_msg *Generate a Step-by-Step Message Function*

Description

Creates a closure function to display sequential progress messages for a specified number of steps.

Usage

```
step_msg(steps)
```

Arguments

steps	An integer specifying the total number of steps.

Value

A function that takes a message string as input and displays it along with the current step and total steps. The step count increments automatically with each call.

Examples

```
stepper <- step_msg(3)
stepper("Initializing") # "Step 1/3 - Initializing"
stepper("Processing") # "Step 2/3 - Processing"
stepper("Finalizing") # "Step 3/3 - Finalizing"
```

submit_table *Submit a Data Frame to an API Endpoint Row by Row*

Description

This function iterates over rows of a data frame, submitting each row to a specified API endpoint function, and combines the responses into a single data structure.

Usage

```
submit_table(tab, id, endpoint_func)
```

Arguments

tab	A data frame containing the data to be submitted.
id	An EGA accession/provisional ID passed to the endpoint_func.
endpoint_func	A function that handles the API request. It should accept id and a JSON body as arguments.

Value

Data frame. A combined response object from the API.

Examples

```
tab <- data.frame(a = 1:2, b = c("x", "y"))
mock_endpoint <- function(id, body) list(id = id, body = body)
submit_table(tab, 12345, mock_endpoint)
```

---

try_step *Execute a submission step with error handling and rollback*

Description

Wraps a logic function in a tryCatch block to handle errors during a specific submission step, optionally triggering a rollback function.

Usage

```
try_step(step_name, logic_fn, rollback_fn, responses, logfile)
```

Arguments

step_name	Character. The name of the current workflow step.
logic_fn	Function. The primary logic to execute for this step.
rollback_fn	Function. A function to clean up if an error occurs.
responses	List. Current collection of API responses for logging.
logfile	Character. Path to the log file.

Value

The result of `logic_fn()`.

Examples

```
try_step(
    "test", function() 1 + 1, function() print("fail"), list(), "log.txt"
)
```

unbox_list *Convert a List to an Unboxed JSON-Compatible Data Frame*

Description

Converts a list into a single-row data frame with unboxed elements if all elements have a length of 1. Otherwise, an error is raised.

Usage

```
unbox_list(l)
```

Arguments

l	A list where all elements must have a length of 1.

Value

A data frame with unboxed elements, suitable for JSON conversion.

Examples

```
input_list <- list(a = 1, b = "text", c = TRUE)
unbox_list(input_list)
```

unbox_row *Convert a Data Frame Row to an Unboxed JSON Object*

Description

This function converts a single row of a data frame into an unboxed JSON object, effectively removing the array structure.

Usage

```
unbox_row(row)
```

Arguments

row	A single row of a data frame.

Value

A JSON object with unboxed values for the input row.

Examples

```
row <- data.frame(a = 1, b = "text", stringsAsFactors = FALSE)[1, ]
unbox_row(row)
```

`use_submission` *Retrieve or Delete Submission Data*

Description

Handles retrieval or deletion of data associated with a submission accession/provisional ID using a specified client and method.

Usage

```
use_submission(id, method, client = NULL)
```

Arguments

`id`	Character or numeric. Represents the submission identifier. Can be either an accession or provisional ID.
`method`	A string specifying the operation to perform. Valid options are "get" or "delete".
`client`	List of functions. EGA API client created by `create_client` function from EGA API schema with `get` and `delete` methods. If `NULL`, default client will be created by `create_client(extract_api())`. Defaults to `NULL`.

Value

A named list containing responses for datasets, analyses, runs, experiments, samples, and studies.

Examples

```
mock_client <- list(
    "get__submissions__accession_id__datasets" = function(id) {
        list(data = id)
    },
    "delete__submissions__provisional_id__datasets" =
        function(id) list(status = "deleted")
)
use_submission("EGAB12345678901", "get", mock_client)
```

validate_schema *Validate a Payload Against a JSON Schema*

Description

Function handles `oneOf` directives in a way that it in a case of validation fail, it displays the overall result of the validation as first and then it tests separately against all `oneOf` sub schemas.

Usage

```
validate_schema(payload, schema)
```

Arguments

`payload`	The payload to validate against the schema. JSON string or single row of data frame converted to JSON representation with `unbox_row` function or a list with all items of length 1 converted to JSON representation with `unbox_list` function.
`schema`	List. The JSON schema defining the validation rules.

Value

Logical value indicating whether the payload is valid. If invalid, the result includes an `errors` attribute detailing the validation errors.

Examples

```
schema <- list(
    type = "object",
    properties = list(
        id = list(type = "integer"),
        title = list(type = "string")
    ),
    required = c("id")
)

payload_true <- data.frame(id = c(12345), title = c("abcd"))
payload_false <- data.frame(id = c("12345"), title = c(0.355))

validate_schema(jsonlite::unbox(payload_true), schema)
validate_schema(jsonlite::unbox(payload_false), schema)
```

validation_to_msg *Convert Validation Results to a Message*

Description

Convert Validation Results to a Message

Usage

```
validation_to_msg(v)
```

Arguments

v	Logical. The validation result, which may include an `errors` attribute detailing validation errors.

Value

A character string summarizing the validation results. If validation errors are present, they are included in the message; otherwise, a success message is returned.

Examples

```
validation_result <- FALSE
attr(validation_result, "errors") <- data.frame(
    field = c("name"),
    message = c("Missing")
)
msg <- validation_to_msg(validation_result)
message(msg)
```

workflow_error_handler *Workflow Error Handler*

Description

Creates a custom error handler for managing errors during a workflow step. Logs responses, executes additional expressions, and stops execution with a detailed message and a stack trace.

Usage

```
workflow_error_handler(step, responses, logfile, ...)
```

Arguments

step	A string representing the current workflow step.
responses	A list of responses to be logged in case of an error.
logfile	A string specifying the path to the log file. If NULL, no file is written.
...	Additional expressions to evaluate when an error occurs.

Value

A function to handle errors during the specified workflow step.

Examples

```
handler <- workflow_error_handler(
    step = "submission",
    responses = list(),
    logfile = NULL
)

tryCatch("Example code without error", error = handler)
```

Index

∗ **datasets**
DELIM_CONVERTERS, 29
∗ **internal**
.EGA_TOKEN_URL, 11
.add_headers, 4
.add_json_validation, 4
.add_paths, 5
.add_queries, 6
.add_request_body, 6
.analyses_extra_validator, 7
.basic_validator, 8
.dataset_analyses_validator, 10
.datasets_extra_validator, 9
.get_ega_password, 11
.get_ega_username, 12
.get_operation_params, 12
.get_rega_key, 13
.has_analyses, 14
.is_client, 14
.is_scalar, 15
.operation_params_to_args, 16
.runs_extra_validator, 16
.studies_extra_validator, 17
.submission_validator, 18
.summarise_validation, 19
.validate_character_scalar, 19
.validate_logical_scalar, 20
.EGA_TOKEN_URL, 11
.add_headers, 4
.add_json_validation, 4
.add_paths, 5
.add_queries, 6
.add_request_body, 6
.analyses_extra_validator, 7
.basic_validator, 8
.dataset_analyses_validator, 10
.datasets_extra_validator, 9
.get_ega_password, 11
.get_ega_username, 12
.get_operation_params, 12
.get_rega_key, 13
.has_analyses, 14
.is_client, 14
.is_scalar, 15
.operation_params_to_args, 16
.runs_extra_validator, 16
.studies_extra_validator, 17
.submission_validator, 18
.summarise_validation, 19
.validate_character_scalar, 19
.validate_logical_scalar, 20

add_required_str, 21
aliases_formatter, 22
api_function_factory, 22
api_name_to_label, 23

column_table_formatter, 24
create_client, 25

default_parser, 25
default_validator, 26
delete_submission, 27
delete_submission_contents, 28
DELIM_CONVERTERS, 29

ega_oauth, 30
ega_token, 31
extract_api, 32
extract_operation_definitions, 32
extract_resource_name, 33

fetch_files, 34
file_formatter, 35
filter_id_fields, 35
finalise_submission, 36
first_row_to_colnames, 37
fold_column, 38
format_chromosomes, 39

get_chr_group, 40
get_entry_by_title, 40
get_enum, 41
get_formatter, 42
get_formatter_params, 43
get_operation_schema, 44
get_or_post, 44
get_properties, 46
get_schemas, 47
get_sentence_number, 47
get_submission, 48
get_word_number, 49

has_linked_sheets, 49

is_accession, 50
is_provisional, 51
is_valid_http_method, 51

label_to_api_name, 52
link_sheet, 52
lut_add, 53, *55*

merge_linked_sheet, 54
multi_lut_add, 54

na_to_empty_list, 55
new_submission, 56

parse_ega_body, 57
parse_enum, 58
parse_json_body, 59
parse_text_body, 60
process_chromosomes, 61
process_delimited_column, 62

rollback_submission, 63
row_table_formatter, 63

samples_in_db, 64
save_log, 65
step_msg, 66
submit_table, 66

try_step, 67

unbox_list, 68
unbox_row, 68
use_submission, 69

validate_schema, 70
validation_to_msg, 71

workflow_error_handler, 71