Package: hubValidations (via r-universe)

October 29, 2026

Title Testing framework for hubverse hub validations

Version 3.0.0

Description This package aims at providing a simple interface to run validations on data and metadata submitted to a hubverse modeling hub. Validation tests can be run at different levels (single file, single folder, whole repository) and locally as well as part of a continuous integration workflow.

License MIT + file LICENSE

Imports arrow (>= 17.0.0), checkmate, cli, config, dplyr (>= 1.1.0), fs, gh, hubAdmin (>= 1.9.0), hubData (>= 2.1.0), hubUtils (>= 1.2.0), jsonlite, jsonvalidate, lifecycle, lubridate, purrr, rlang, stringr, tibble, whisker, yaml

Suggests covr, curl, DT, gert, kableExtra, pak, readr, rmarkdown, testthat (>= 3.2.0), testthis, withr

Remotes hubverse-org/hubUtils, hubverse-org/hubData, hubverse-org/hubAdmin

Config/testthat/edition 3

Config/Needs/website pkgdown, hubverse-org/hubStyle

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://github.com/hubverse-org/hubValidations, https://hubverse-org.github.io/hubValidations/

BugReports https://github.com/hubverse-org/hubValidations/issues

Depends R (>= 4.1.0)

Config/pak/sysreqs cmake git make libicu-dev libuv1-dev libxml2-dev libssl-dev libnode-dev libx11-dev

Repository https://hubverse-org.r-universe.dev

Date/Publication 2026-10-07 07:35:20 UTC

RemoteUrl https://github.com/hubverse-org/hubValidations

RemoteRef v3.0.0

RemoteSha 4d4596f03dad5fb5dc2cf60eedc3f4d06f331631

Contents

capture_check_cnd . . . 4
capture_check_info . . . 5
capture_exec_error . . . 6
capture_exec_warning . . . 6
capture_validation_warning . . . 7
check_config_hub_valid . . . 8
check_file_exists . . . 9
check_file_format . . . 9
check_file_location . . . 10
check_file_n . . . 11
check_file_name . . . 11
check_file_read . . . 12
check_for_errors . . . 13
check_metadata_file_exists . . . 13
check_metadata_file_ext . . . 14
check_metadata_file_location . . . 15
check_metadata_file_name . . . 15
check_metadata_matches_schema . . . 16
check_metadata_schema_exists . . . 17
check_submission_metadata_file_exists . . . 18
check_submission_time . . . 18
check_target_dataset_exists . . . 19
check_target_dataset_file_ext_unique . . . 20
check_target_dataset_rows_unique . . . 21
check_target_dataset_unique . . . 22
check_target_file_ext_valid . . . 23
check_target_file_name . . . 24
check_target_file_read . . . 24
check_target_tbl_colnames . . . 25
check_target_tbl_coltypes . . . 27
check_target_tbl_oracle_value . . . 28
check_target_tbl_output_type_ids . . . 30
check_target_tbl_rows_unique . . . 31
check_target_tbl_ts_targets . . . 33
check_target_tbl_values . . . 34
check_tbl_col_types . . . 35
check_tbl_colnames . . . 36
check_tbl_derived_task_id_vals . . . 37
check_tbl_match_round_id . . . 38
check_tbl_rows_unique . . . 39

check_tbl_spl_compound_taskid_set . . . 40
check_tbl_spl_compound_tid . . . 42
check_tbl_spl_mt_unique . . . 43
check_tbl_spl_n . . . 44
check_tbl_spl_non_compound_tid . . . 46
check_tbl_unique_round_id . . . 47
check_tbl_value_col . . . 48
check_tbl_value_col_ascending . . . 49
check_tbl_value_col_sum1 . . . 50
check_tbl_values . . . 51
check_tbl_values_required . . . 52
check_valid_round_id . . . 53
check_valid_round_id_col . . . 54
combine . . . 55
create_custom_check . . . 56
expand_model_out_grid . . . 57
get_config_derived_task_ids . . . 62
get_target_task_id . . . 63
get_tbl_compound_taskid_set . . . 64
is_success . . . 65
match_tbl_to_model_task . . . 66
new_hub_validations . . . 68
new_hub_validations_collection . . . 69
new_target_validations . . . 70
new_target_validations_collection . . . 71
opt_check_metadata_team_max_model_n . . . 73
opt_check_tbl_col_timediff . . . 74
opt_check_tbl_counts_lt_popn . . . 75
opt_check_tbl_horizon_timediff . . . 76
parse_file_name . . . 78
print.hub_validations . . . 79
print.hub_validations_collection . . . 80
read_model_out_file . . . 81
read_target_file . . . 82
submission_tmpl . . . 83
try_check . . . 87
validate_model_data . . . 88
validate_model_file . . . 90
validate_model_metadata . . . 91
validate_pr . . . 92
validate_submission . . . 95
validate_submission_time . . . 97
validate_target_data . . . 98
validate_target_dataset . . . 100
validate_target_file . . . 101
validate_target_pr . . . 103
validate_target_submission . . . 106

Index **109**

`capture_check_cnd` *Capture a condition of the result of validation check.*

Description

Capture a condition of the result of validation check.

Usage

```
capture_check_cnd(
  check,
  file_path,
  msg_subject,
  msg_attribute,
  msg_verbs = c("is", "must be"),
  error = FALSE,
  details = NULL,
  ...
)
```

Arguments

`check`	logical, the result of a validation check. If `check` is `FALSE`, validation has failed. If `check` is `TRUE`, validation has succeeded.
`file_path`	character string. Path to the file being validated. Must be the relative path to the hub's `model-output` (or equivalent) directory.
`msg_subject`	character string. The subject of the validation.
`msg_attribute`	character string. The attribute of subject being validated.
`msg_verbs`	character vector of length 2. The verbs describing the state of the attribute in relation to the validation subject. The first element describes the state when validation succeeds, the second element, when validation fails.
`error`	logical. In the case of validation failure, whether the function should return an object of class `<error/check_error>` (`TRUE`) or `<error/check_failure>` (`FALSE`, default).
`details`	further details to be appended to the output message.
`...`	<dynamic> Named data fields stored inside the condition object.

Details

Arguments `msg_subject`, `msg_attribute`, `msg_verbs` and `details` accept text that can interpreted and formatted by `cli::format_inline()`.

Value

Depending on whether validation has succeeded and the value of the `error` argument, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

Examples

```
capture_check_cnd(
  check = TRUE, file_path = "test/file.csv",
  msg_subject = "{.var round_id}", msg_attribute = "valid.", error = FALSE
)
capture_check_cnd(
  check = FALSE, file_path = "test/file.csv",
  msg_subject = "{.var round_id}", msg_attribute = "valid.", error = FALSE,
  details = "Must be one of 'A' or 'B', not 'C'"
)
capture_check_cnd(
  check = FALSE, file_path = "test/file.csv",
  msg_subject = "{.var round_id}", msg_attribute = "valid.", error = TRUE,
  details = "Must be one of {.val {c('A', 'B')}}, not {.val C}"
)
```

`capture_check_info` *Capture a simple info message condition*

Description

Capture a simple info message condition. Useful for communicating when a check is ignored or skipped.

Usage

```
capture_check_info(file_path, msg, call = rlang::caller_call())
```

Arguments

`file_path`	character string. Path to the file being validated. Must be the relative path to the hub's `model-output` (or equivalent) directory.
`msg`	Character string. Accepts text that can interpreted and formatted by `cli::format_inline()`.
`call`	The defused call of the function that generated the message. Use to override default which uses the caller call. See rlang::stack for more details.

Value

A `<message/check_info>` condition class object. Returned object also inherits from subclass `<hub_check>`.

---

capture_exec_error *Capture an execution error condition*

Description

Capture an execution error condition. Useful for communicating when a check execution has failed. Usually used in conjunction with `try`.

Usage

```
capture_exec_error(file_path, msg, call = NULL)
```

Arguments

`file_path`	character string. Path to the file being validated. Must be the relative path to the hub's `model-output` (or equivalent) directory.
`msg`	Character string.
`call`	Character string. Name of the parent call that failed to execute. If `NULL` (default), the caller's call name is captured.

Value

A `<error/check_exec_error>` condition class object. Returned object also inherits from subclass `<hub_check>`.

---

capture_exec_warning *Capture an execution warning condition*

Description

Capture an execution warning condition. Useful for communicating when a check execution has failed. Usually used in conjunction with `try`.

Usage

```
capture_exec_warning(file_path, msg, call = NULL)
```

Arguments

`file_path`	character string. Path to the file being validated. Must be the relative path to the hub's `model-output` (or equivalent) directory.
`msg`	Character string.
`call`	Character string. Name of the parent call that failed to execute. If `NULL` (default), the caller's call name is captured.

Value

A `<warning/check_exec_warn>` condition class object. Returned object also inherits from subclass `<hub_check>`.

`capture_validation_warning` *Capture a validation warning condition*

Description

Capture a warning about the validation process. Unlike check results (success/failure/error), validation warnings are informational messages about the validation process itself rather than validation outcomes. They do **not** cause validation to fail — `check_for_errors()` only aborts on `check_failure`, `check_error`, `check_exec_error` or `check_exec_warn` objects.

Usage

```
capture_validation_warning(msg, where = NULL, call = rlang::caller_call(), ...)
```

Arguments

`msg`	Character string. The warning message. Accepts text that can be interpreted and formatted by `cli::format_inline()`.
`where`	Optional. Character string indicating the location or context of the warning (e.g., file path, `"hub-config"`). Used as metadata.
`call`	The defused call of the function that generated the warning. Use to override default which uses the caller call. See rlang::stack for more details.
`...`	Additional named fields to include in the warning condition object. Useful for attaching structured data (e.g., `config_files = c("tasks.json")`).

Details

Validation warnings can be attached at two levels:

- **Validation-level**: Stored as an attribute on `hub_validations` objects, printed prominently by default.
- **Check-level**: Stored in a `warnings` field on individual check results, printed only when `verbose = TRUE`.

Value

A <warning/validation_warning> condition class object.

Examples

```
# Simple warning
capture_validation_warning(
  msg = "Configuration files were modified"
)

# Warning with location and additional structured data
config_files <- c("tasks.json", "admin.json")
capture_validation_warning(
  msg = "Config files modified: {.path {config_files}}",
  where = "hub-config",
  config_files = config_files
)
```

---

check_config_hub_valid

Check hub correctly configured

Description

Checks that admin and tasks configuration files in directory hub-config are valid.

Usage

```
check_config_hub_valid(hub_path)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_error> condition class object.

Returned object also inherits from subclass <hub_check>.

check_file_exists *Check file exists at the file path specified*

Description

Check file exists at the file path specified

Usage

```
check_file_exists(
  file_path,
  hub_path = ".",
  subdir = c("model-output", "model-metadata", "hub-config", "target-data")
)
```

Arguments

file_path	character string. Path to the file being validated relative to the hub's model-output directory.
hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
subdir	subdirectory within the hub

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_error> condition class object.

Returned object also inherits from subclass <hub_check>.

check_file_format *Check file format is accepted by hub.*

Description

Check file format is accepted by hub.

Usage

```
check_file_format(file_path, hub_path, round_id)
```

Arguments

file_path	character string. Path to the file being validated relative to the hub's model-output directory.
hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
round_id	character string. The round identifier.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_error> condition class object.

Returned object also inherits from subclass <hub_check>.

---

check_file_location *Check file is being submitted to the correct folder*

Description

Checks that the model_id metadata in the file name matches the directory name the file is being submitted to.

Usage

```
check_file_location(file_path)
```

Arguments

file_path	character string. Path to the file being validated relative to the hub's model-output directory.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_failure> condition class object.

Returned object also inherits from subclass <hub_check>.

`check_file_n` *Check number of files submitted per round does not exceed the allowed number of submissions per team.*

Description

Check number of files submitted per round does not exceed the allowed number of submissions per team.

Usage

```
check_file_n(file_path, hub_path, allowed_n = 1L)
```

Arguments

`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`allowed_n`	integer(1). The maximum number of files allowed per round.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_file_name` *Check a model output file name can be correctly parsed.*

Description

Check a model output file name can be correctly parsed.

Usage

```
check_file_name(file_path)
```

Arguments

file_path	character string. Path to the file being validated relative to the hub's model-output directory.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_error> condition class object.

Returned object also inherits from subclass <hub_check>.

---

check_file_read *Check file can be read successfully*

Description

Check file can be read successfully

Usage

```
check_file_read(file_path, hub_path = ".")
```

Arguments

file_path	character string. Path to the file being validated relative to the hub's model-output directory.
hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_error> condition class object.

Returned object also inherits from subclass <hub_check>.

check_for_errors *Raise conditions stored in validation objects*

Description

Checks validation objects for errors and raises conditions if any are found. Works with `hub_validations` and `hub_validations_collection` objects, as well as their subclasses (`target_validations` and `target_validations_collection`). Can be used in CI workflows to signal validation failures, or locally to summarise validation results.

Usage

```
check_for_errors(x, verbose = FALSE, show_warnings = FALSE)
```

Arguments

`x`	A `hub_validations` or `hub_validations_collection` object (including subclasses `target_validations` and `target_validations_collection`).
`verbose`	Logical. If `TRUE`, print the results of all checks prior to raising condition and summarising validation object check results.
`show_warnings`	Logical. If `TRUE`, print check-level warnings inline with their checks. Validation-level warnings are always printed. Default `FALSE`.

Details

For more details on these classes, see article on `<hub_validations>` S3 class objects.

Value

An error if one of the elements of `x` is of class `check_failure`, `check_error`, `check_exec_error` or `check_exec_warning`. `TRUE` invisibly otherwise.

See Also

`validate_submission()`, `validate_pr()`, `validate_target_submission()`, `validate_target_pr()`

check_metadata_file_exists *Check whether a metadata schema file exists*

Description

Check whether a metadata schema file exists

Usage

```
check_metadata_file_exists(hub_path = ".", file_path)
```

Arguments

`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`file_path`	character string. Path to the file being validated relative to the hub's model-metadata directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_metadata_file_ext`

Check file is being submitted to the correct folder

Description

Checks that the `model_id` metadata in the file name matches the directory name the file is being submitted to.

Usage

```
check_metadata_file_ext(file_path)
```

Arguments

`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_metadata_file_location`
Check that the metadata file is being submitted to the correct folder

---

Description

Check that the metadata file is being submitted to the correct folder

Usage

```
check_metadata_file_location(file_path)
```

Arguments

`file_path`	character string. Path to the file being validated relative to the hub's model-metadata directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_metadata_file_name`
Check whether the file name of a metadata file matches the model_id or combination of team_abbr and model_abbr specified within the metadata file

---

Description

Check whether the file name of a metadata file matches the model_id or combination of team_abbr and model_abbr specified within the metadata file

Usage

```
check_metadata_file_name(file_path, hub_path = ".")
```

Arguments

file_path	character string. Path to the file being validated relative to the hub's model-metadata directory.
hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_failure> condition class object.

Returned object also inherits from subclass <hub_check>.

---

check_metadata_matches_schema

Check whether a metadata file matches the schema provided by the hub

---

Description

Check whether a metadata file matches the schema provided by the hub

Usage

```
check_metadata_matches_schema(file_path, hub_path = ".")
```

Arguments

file_path	character string. Path to the file being validated relative to the hub's model-metadata directory.
hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_metadata_schema_exists`
Check whether a metadata schema file exists

Description

Check whether a metadata schema file exists

Usage

```
check_metadata_schema_exists(hub_path = ".", file_path)
```

Arguments

`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`file_path`	Path to the model metadata file being validated. Used as the `$where` in the returned check object, since this check is a prerequisite for validating that file.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_submission_metadata_file_exists`
Check whether a metadata file for the given model exists

Description

Check whether a metadata file for the given model exists

Usage

```
check_submission_metadata_file_exists(file_path, hub_path = ".")
```

Arguments

`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_submission_time`
Checks submission is within the valid submission window for a given round.

Description

Checks submission is within the valid submission window for a given round.

Usage

```
check_submission_time(
  hub_path,
  file_path,
  ref_date_from = c("file", "file_path")
)
```

Arguments

`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`ref_date_from`	whether to get the reference date around which relative submission windows will be determined from the file's `file_path` round ID or the `file` contents themselves. `file` requires that the file can be read. Only applicable when a round is configured to determine the submission windows relative to the value in a date column in model output files. Not applicable when explicit submission window start and end dates are provided in the hub's config.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_dataset_exists` *Check target dataset can be detected for a given target type*

Description

Check target dataset can be detected for a given target type

Usage

```
check_target_dataset_exists(
  hub_path,
  target_type = c("time-series", "oracle-output")
)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
target_type	Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_error> condition class object.

Returned object also inherits from subclass <hub_check>.

---

check_target_dataset_file_ext_unique

Check that all files of a given target type share a single unique file format

---

Description

Check that all files of a given target type share a single unique file format

Usage

```
check_target_dataset_file_ext_unique(
  hub_path,
  target_type = c("time-series", "oracle-output")
)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
target_type	Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_dataset_rows_unique`
Check target dataset rows are all unique

Description

Check that there are no duplicate rows in a target dataset. Function designed to be used as part of overall target data integrity check.

Usage

```
check_target_dataset_rows_unique(
  target_type = c("time-series", "oracle-output"),
  na = c("NA", ""),
  date_col = NULL,
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  hub_path
)
```

Arguments

`target_type`	Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".
`na`	A character vector of strings to interpret as missing values. Only applies to CSV files. The default is `c("NA", "")`. Useful when actual character string `"NA"` values are used in the data. In such a case, use empty cells to indicate missing values in your files and set `na = ""`.
`date_col`	Optional column name to be interpreted as date for dataset connection. Useful when the date column does not correspond to a valid task ID (e.g., calculated from other task IDs like `origin_date + horizon`), particularly when it is also a partitioning column. Ignored when `target-data.json` config is provided.
`output_type_id_datatype`	character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json` config file if available. If the property is not set in the config, the argument

falls back to "auto" which determines the output_type_id data type automatically from the tasks.json config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where output_type_ids are NA,) are being collected by a hub, the output_type_id column is assigned a character data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce output_type_id to a data type that is not valid for the data (e.g. trying to coerce"character" values to "double") will likely result in an error or potentially unexpected behaviour so use with care.

hub_path — Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.

Details

If datasets are versioned, multiple observations are allowed in time-series target data, so long as they have different as_of values. The as_of column is therefore included when determining duplicates. In oracle-output data, there should be only a single observation, regardless of the as_of value so the column it is not be included when determining duplicates.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_failure> condition class object.

Returned object also inherits from subclass <hub_check>.

---

check_target_dataset_unique

Check that a single unique target dataset exists for a given target type.

Description

Check that a single unique target dataset exists for a given target type.

Usage

```
check_target_dataset_unique(
  hub_path,
  target_type = c("time-series", "oracle-output")
)
```

Arguments

`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`target_type`	Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_file_ext_valid`

Check that a target data file has a valid extension.

Description

Note that files which are part of a hive partitioned dataset must have parquet file extension only.

Usage

```
check_target_file_ext_valid(file_path)
```

Arguments

`file_path`	A character string representing the path to the target data file relative to the `target-data` directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_file_name`
Check that a hive-partitioned target data file name can be correctly parsed.

Description

Check that a hive-partitioned target data file name can be correctly parsed.

Usage

```
check_target_file_name(file_path)
```

Arguments

`file_path`	A character string representing the path to the target data file relative to the `target-data` directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_file_read`
Check target file can be read successfully

Description

Check target file can be read successfully

Usage

```
check_target_file_read(file_path, hub_path = ".")
```

Arguments

`file_path`	A character string representing the path to the target data file relative to the `target-data` directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_tbl_colnames` *Check that a target data file has the correct column names according to target type*

Description

Check that a target data file has the correct column names according to target type

Usage

```
check_target_tbl_colnames(
  target_tbl,
  target_type = c("time-series", "oracle-output"),
  file_path,
  hub_path,
  config_target_data = NULL,
  date_col = NULL
)
```

Arguments

`target_tbl`	A tibble/data.frame of the contents of the target data file being validated.
`target_type`	Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".
`file_path`	A character string representing the path to the target data file relative to the `target-data` directory.

`hub_path` Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

`config_target_data` Optional. A `target-data.json` config object. If provided, validation uses deterministic schema from config. If `NULL` (default), validation uses inference from `tasks.json`.

`date_col` Optional. Name of the date column in target data (e.g., `"target_end_date"`) representing the date observations actually occurred. Only relevant when it is not a task ID defined in `tasks.json`. Enables deterministic validation in inference mode. Ignored when `config_target_data` is provided.

Details

Column name validation depends on whether a `target-data.json` configuration file is provided:

With `target-data.json` config: Expected columns are determined directly from the configuration. The target table must contain exactly the columns defined in the config.

Without `target-data.json` config (inference mode): Expected columns are inferred from the task ID configuration in `tasks.json`, allowed columns according to the target type, and expectations based on the detected output types in the target data. Additional optional columns (e.g., `as_of`) are allowed for time-series data.

Note on date columns: Target data always contains a date column (e.g., `target_end_date`) representing when observations occurred. However, in horizon-based forecast hubs, task IDs may only define `origin_date` and `horizon` (with target dates calculated from these). In such cases, provide `date_col` to enable deterministic validation of the date column when it is not a valid task ID. Validation of date column existence and type is performed by `check_target_tbl_coltypes()`.

Inference mode validation for time-series data is limited. For robust validation, create a `target-data.json` config file. See `target-data.json` schema # nolint: line_length_linter. for more information on the json schema scpecifics.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_tbl_coltypes` *Check that a target data file has the correct column types according to target type*

Description

Check that a target data file has the correct column types according to target type

Usage

```
check_target_tbl_coltypes(
  target_tbl,
  target_type = c("time-series", "oracle-output"),
  date_col = NULL,
  na = c("NA", ""),
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  file_path,
  hub_path
)
```

Arguments

`target_tbl`	A tibble/data.frame of the contents of the target data file being validated.
`target_type`	Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".
`date_col`	Optional column name to be interpreted as date for schema creation. Useful when the date column does not correspond to a valid task ID (e.g., calculated from other task IDs like `origin_date + horizon`), particularly when it is also a partitioning column. Ignored when `target-data.json` config is provided.
`na`	A character vector of strings to interpret as missing values. Only applies to CSV files. The default is `c("NA", "")`. Useful when actual character string `"NA"` values are used in the data. In such a case, use empty cells to indicate missing values in your files and set `na = ""`.
`output_type_id_datatype`	character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json` config file if available. If the property is not set in the config, the argument falls back to `"auto"` which determines the `output_type_id` data type automatically from the `tasks.json` config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where `output_type_id`s

are NA,) are being collected by a hub, the output_type_id column is assigned a character data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce output_type_id to a data type that is not valid for the data (e.g. trying to coerce"character" values to "double") will likely result in an error or potentially unexpected behaviour so use with care.

file_path — A character string representing the path to the target data file relative to the target-data directory.

hub_path — Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.

Details

Column type validation depends on whether a target-data.json configuration file is provided:

With target-data.json config: Expected column types are determined directly from the schema defined in the configuration. Validation is performed against the schema specifications in target-data.json.

Without target-data.json config: Expected column types are determined from the dataset itself and validated for internal consistency across files, which mainly applies to partitioned datasets.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_error> condition class object.

Returned object also inherits from subclass <hub_check>.

---

check_target_tbl_oracle_value

Check that oracle values in an oracle output target data file are valid

---

Description

This check is only performed when the target data file contains an output_type_id column and cdf or pmf output types. It verifies that distributional output type (cdf and pmf) oracle values meet the following criteria:

- `oracle_value` values are either `0` or `1`.
- `pmf` oracle values sum to `1` for each observation unit.
- `cdf` oracle values are non-decreasing for each observation unit when sorted by the `output_type_id` set defined in the hub config.

Usage

```
check_target_tbl_oracle_value(
  target_tbl,
  target_type = c("oracle-output", "time-series"),
  file_path,
  hub_path,
  config_target_data = NULL
)
```

Arguments

`target_tbl`	A tibble/data.frame of the contents of the target data file being validated.
`target_type`	Type of target data to validate. One of `"time-series"` or `"oracle-output"`. Defaults to `"oracle-output"`.
`file_path`	A character string representing the path to the target data file relative to the `target-data` directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`config_target_data`	Target data configuration object from `read_config(hub_path, "target-data")`, or NULL (default) if config does not exist. When target-data.json exists, this should be provided to enable date column extraction for date relaxation. If NULL and date_col is not provided, date relaxation cannot be applied and a warning will be issued if allow_extra_dates is TRUE.

Details

When validating oracle values, data is grouped by observation unit to check PMF sums and CDF monotonicity within each unit.

With `target-data.json` config: Observable unit is determined from the config's `observable_unit` specification.

Without `target-data.json` config: Observable unit is inferred from task ID columns present in the data.

The `as_of` column is NOT included in the grouping. Oracle data is designed to contain a single version per observable unit with a one-to-one mapping to model output data.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_tbl_output_type_ids` *Check that output type ID values in a target data file are valid and complete*

Description

This check is only performed when the target data file contains an `output_type_id` column. It verifies that non-distributional output types have all NA output type IDs, and that distributional output types (`cdf`, `pmf`) include the complete output_type_id set defined in the hub config.

Usage

```
check_target_tbl_output_type_ids(
  target_tbl_chr,
  target_type = c("oracle-output", "time-series"),
  file_path,
  hub_path,
  config_target_data = NULL
)
```

Arguments

`target_tbl_chr`	A tibble/data.frame of the contents of the target data file being validated. All columns should be coerced to character.
`target_type`	Type of target data to validate. One of `"time-series"` or `"oracle-output"`. Defaults to `"oracle-output"`.
`file_path`	A character string representing the path to the target data file relative to the `target-data` directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

`config_target_data`
: Target data configuration object from `read_config(hub_path, "target-data")`, or NULL (default) if config does not exist. When target-data.json exists, this should be provided to enable date column extraction for date relaxation. If NULL and date_col is not provided, date relaxation cannot be applied and a warning will be issued if allow_extra_dates is TRUE.

Details

When checking for completeness of distributional output types, data is grouped by observation unit to verify each unit has the complete set of output_type_id values.

With `target-data.json` config: Observable unit is determined from the config's `observable_unit` specification.

Without `target-data.json` config: Observable unit is inferred from task ID columns present in the data.

The `as_of` column is NOT included in the grouping. Oracle data is designed to contain a single version per observable unit with a one-to-one mapping to model output data.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_tbl_rows_unique`

Check target data rows are all unique

Description

Check that there are no duplicate rows in target data files being validated.

Usage

```
check_target_tbl_rows_unique(
  target_tbl,
  target_type = c("time-series", "oracle-output"),
  file_path,
  hub_path,
  config_target_data = NULL
)
```

Arguments

`target_tbl`	A tibble/data.frame of the contents of the target data file being validated.
`target_type`	Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".
`file_path`	A character string representing the path to the target data file relative to the `target-data` directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`config_target_data`	Optional. A `target-data.json` config object. If provided, validation uses deterministic schema from config. If `NULL` (default), validation uses inference from `tasks.json`.

Details

Row uniqueness is determined by checking for duplicate combinations of key columns (excluding value columns).

With `target-data.json` config: Columns to check are determined from the config's `observable_unit` specification. For `oracle-output` data with output type IDs, the `output_type` and `output_type_id` columns are also included in the uniqueness check.

Without `target-data.json` config: For `time-series` data, if versioned, multiple observations are allowed so long as they have different `as_of` values. The `as_of` column is therefore included when determining duplicates. For `oracle-output` data, there should be only a single observation, regardless of the `as_of` value, so the column is not included when determining duplicates.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_tbl_ts_targets` *Check that targets in a time-series target data file are valid*

Description

Check is only performed when the target data type is `time-series`. When the target task ID is not specified in the config (i.e. hub has single target and `target_keys = NULL`), the validity of the target is only checked through the config file. Otherwise, the values in the target task ID column of `target_tbl` are checked. Note that valid `time-series` targets must be step ahead and their target type must be one of `"continuous"`, `"discrete"`, `"binary"` or `"compositional"`. If the hub contains no valid time-series targets, no time-series target data should be present and validation of such data will be skipped.

Usage

```
check_target_tbl_ts_targets(
  target_tbl,
  target_type = c("time-series", "oracle-output"),
  file_path,
  hub_path
)
```

Arguments

`target_tbl`	A tibble/data.frame of the contents of the target data file being validated.
`target_type`	Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".
`file_path`	A character string representing the path to the target data file relative to the `target-data` directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_target_tbl_values` *Check that task ID columns in a target data file have valid task ID values*

Description

Check is only performed when the target data file contains columns that map onto task IDs or output types defined in the hub configuration.

Usage

```
check_target_tbl_values(
  target_tbl_chr,
  target_type = c("time-series", "oracle-output"),
  file_path,
  hub_path,
  date_col = NULL,
  allow_extra_dates = FALSE,
  config_target_data = NULL
)
```

Arguments

`target_tbl_chr`
: A tibble/data.frame of the contents of the target data file being validated. All columns should be coerced to character.

`target_type`
: Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".

`file_path`
: A character string representing the path to the target data file relative to the `target-data` directory.

`hub_path`
: Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

`date_col`
: Optional. Name of the date column (e.g., "target_end_date"). Only used when target-data.json config does not exist. When target-data.json exists, date column is extracted from config (this parameter is ignored). If cannot determine date column, date relaxation is skipped.

`allow_extra_dates`
: Logical. If TRUE and target_type is "time-series", allows date values not in tasks.json. Other task ID columns are still strictly validated. Ignored for oracle-output (always strict).

config_target_data
: Target data configuration object from `read_config(hub_path, "target-data")`, or NULL (default) if config does not exist. When target-data.json exists, this should be provided to enable date column extraction for date relaxation. If NULL and date_col is not provided, date relaxation cannot be applied and a warning will be issued if allow_extra_dates is TRUE.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

---

check_tbl_col_types *Check model data column data types*

Description

Check that model output data column datatypes conform to those define in the hub config.

Usage

```
check_tbl_col_types(
  tbl,
  file_path,
  hub_path,
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date")
)
```

Arguments

tbl	a tibble/data.frame of the contents of the file being validated.
file_path	character string. Path to the file being validated relative to the hub's model-output directory.
hub_path	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

`output_type_id_datatype`
: character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json` config file if available. If the property is not set in the config, the argument falls back to `"auto"` which determines the `output_type_id` data type automatically from the `tasks.json` config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where `output_type_id`s are `NA`,) are being collected by a hub, the `output_type_id` column is assigned a `character` data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce `output_type_id` to a data type that is not valid for the data (e.g. trying to coerce`"character"` values to `"double"`) will likely result in an error or potentially unexpected behaviour so use with care.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_tbl_colnames` *Check column names of model output data*

Description

Checks that a tibble/data.frame of data read in from the file being validated contains the expected task ID and standard column names according the round configuration being validated against.

Usage

```
check_tbl_colnames(tbl, round_id, file_path, hub_path = ".")
```

Arguments

`tbl`	a tibble/data.frame of the contents of the file being validated.
`round_id`	character string. The round identifier.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.

`hub_path` Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_tbl_derived_task_id_vals` *Check derived task ID columns contain valid values*

Description

[Superseded]

Usage

```
check_tbl_derived_task_id_vals(
  tbl_chr,
  round_id,
  file_path,
  hub_path,
  derived_task_ids = get_hub_derived_task_ids(hub_path, round_id)
)
```

Arguments

`tbl_chr` a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.

`round_id` character string. The round identifier.

`file_path` character string. Path to the file being validated relative to the hub's model-output directory.

`hub_path` Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

`derived_task_ids`
: Character vector of derived task ID names (task IDs whose values depend on other task IDs) to validate. Defaults to extracting derived task IDs from hub `task.json`. See `get_hub_derived_task_ids()` for more details.

Details

`check_tbl_values()` validates derived task ID columns like any other task ID column, so `validate_model_data()` no longer runs this check. It remains available as a custom check.

This check is used to validate that values in any derived task ID columns matches accepted values for each derived task ID in the config. Given the dependence of derived task IDs on the values of other values, it ignores the combinations of derived task ID values with those of other task IDs and focuses only on identifying values that do not match the accepted values.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

If no `derived_task_ids` are specified, the check is skipped and a `<message/check_info>` condition class object is retuned.

Returned object also inherits from subclass `<hub_check>`.

`check_tbl_match_round_id`

Check model output data tbl round ID matches submission round ID.

Description

Check model output data tbl round ID matches submission round ID.

Usage

```
check_tbl_match_round_id(tbl, file_path, hub_path, round_id_col = NULL)
```

Arguments

`tbl`	a tibble/data.frame of the contents of the file being validated.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.

`hub_path` Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

`round_id_col` Character string. The name of the column containing `round_id`s. Usually, the value of round property `round_id` in hub `tasks.json` config file. Defaults to `NULL` and determined from the config if applicable.

Details

This check only applies to files being submitted to rounds where `round_id_from_variable: true` or where a `round_id_col` name is explicitly provided. Skipped otherwise.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

If `round_id_from_variable: false` and no `round_id_col` name is provided, check is skipped and a `<message/check_info>` condition class object is returned. If no valid `round_id_col` name is provided or can extracted from config (check through `check_valid_round_id_col`), a `<message/check_error>` condition class object is returned and the rest of the check skipped.

`check_tbl_rows_unique`

Check model data rows are all unique

Description

Checks that combinations of task ID, output type and output type ID value combinations are unique, by checking that there are no duplicate rows across all `tbl_chr` columns excluding the `value` column.

Usage

```
check_tbl_rows_unique(tbl_chr, file_path, hub_path)
```

Arguments

`tbl_chr`	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_tbl_spl_compound_taskid_set`

Check model output data tbl sample compound task id sets for each modeling task match or are coarser than the expected set defined in the config.

Description

This check detects the compound task ID sets of samples, implied by the `output_type_id` and task ID values, and checks them for internal consistency and compliance with the `compound_taskid_set` defined for each round modeling task in the `tasks.json` config.

Usage

```
check_tbl_spl_compound_taskid_set(
  tbl_chr,
  round_id,
  file_path,
  hub_path,
  derived_task_ids = get_hub_derived_task_ids(hub_path, round_id)
)
```

Arguments

`tbl_chr`	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
`round_id`	character string. The round identifier.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`derived_task_ids`	Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore during validation. Defaults to extracting derived task IDs from hub `task.json`. See `get_hub_derived_task_ids()` for more details.

Details

If the check fails, the output of the check includes an `errors` element, a list of items, one for each modeling task failing validation. The structure depends on the reason the check failed.

If the check failed because more that a single unique `compound_taskid_set` was found for a given model task, the `errors` object will be a list with one element for each `compound_taskid_set` detected and will have the following structure:

- `tbl_comp_tids`: a compound task id set detected in the the tbl.
- `output_type_ids`: The output type ID of the sample that does not contain a single, unique value for each compound task ID.

If the check failed because task IDs the hub does not accept as compound task IDs were identified as such (i.e. samples describe "finer" compound modeling tasks) for a given model task, the `errors` object will be a list with the structure described above as well as the additional following elements:

- `config_comp_tids`: the `compound_taskid_set` the modeling task config expects.
- `invalid_tbl_comp_tids`: the names of invalid compound task IDs.

The name of each element is the index identifying the config modeling task the sample is associated with `mt_id`. See hubverse documentation on samples for more details.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.

- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_tbl_spl_compound_tid`

Check model output data tbl samples contain single unique values for each compound task ID within individual samples

---

Description

Check model output data tbl samples contain single unique values for each compound task ID within individual samples

Usage

```
check_tbl_spl_compound_tid(
  tbl_chr,
  round_id,
  file_path,
  hub_path,
  compound_taskid_set = NULL,
  derived_task_ids = get_hub_derived_task_ids(hub_path, round_id)
)
```

Arguments

`tbl_chr`	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
`round_id`	character string. The round identifier.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`compound_taskid_set`	a list of `compound_taskid_sets` (characters vector of compound task IDs), one for each modeling task. Used to override the compound task ID set in the config file, for example, when validating coarser samples.
`derived_task_ids`	Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore during validation. Defaults to extracting derived task IDs from hub `task.json`. See `get_hub_derived_task_ids()` for more details.

Details

Output of the check includes an `errors` element, a list of items, one for each sample failing validation, with the following structure:

- `mt_id`: Index identifying the config modeling task the sample is associated with.
- `output_type_id`: The output type ID of the sample that does not contain a single, unique value for each compound task ID.
- `values`: The unique values of each compound task ID. See hubverse documentation on samples for more details.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_tbl_spl_mt_unique`

Check that individual sample output_type_ids do not span multiple model tasks

---

Description

Check that individual sample output_type_ids do not span multiple model tasks

Usage

```
check_tbl_spl_mt_unique(
  tbl_chr,
  round_id,
  file_path,
  hub_path,
  derived_task_ids = get_hub_derived_task_ids(hub_path, round_id)
)
```

Arguments

`tbl_chr`	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
`round_id`	character string. The round identifier.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.

`hub_path` Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

`derived_task_ids` Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore during validation. Defaults to extracting derived task IDs from hub `task.json`. See `get_hub_derived_task_ids()` for more details.

Details

Different model tasks can have different sample configurations (`compound_taskid_set`, `min/max_samples_per_task`, etc.), so samples should be entirely independent across model tasks. This check verifies that no sample `output_type_id` appears in more than one model task.

Output of the check includes an `errors` element, a list with the following structure:

- `mt_ids`: Integer vector of model task indices the overlapping samples span.
- `output_type_ids`: Character vector of sample `output_type_id`s that appear in multiple model tasks.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_tbl_spl_n` *Check model output data tbl samples contain the appropriate number of samples for a given compound idx.*

Description

Check model output data tbl samples contain the appropriate number of samples for a given compound idx.

Usage

```
check_tbl_spl_n(
  tbl_chr,
  round_id,
  file_path,
  hub_path,
  compound_taskid_set = NULL,
  derived_task_ids = get_hub_derived_task_ids(hub_path, round_id)
)
```

Arguments

`tbl_chr`	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
`round_id`	character string. The round identifier.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`compound_taskid_set`	a list of `compound_taskid_set`s (characters vector of compound task IDs), one for each modeling task. Used to override the compound task ID set in the config file, for example, when validating coarser samples.
`derived_task_ids`	Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore during validation. Defaults to extracting derived task IDs from hub `task.json`. See `get_hub_derived_task_ids()` for more details.

Details

Output of the check includes an `errors` element, a list of items, one for each compound_idx failing validation, with the following structure:

- `compound_idx`: the compound idx that failed validation of number of samples.
- `n`: the number of samples counted for the compound idx.
- `min_samples_per_task`: the minimum number of samples required for the compound idx.
- `max_samples_per_task`: the maximum number of samples required for the compound idx.
- `compound_idx_tbl`: a tibble of the expected structure for samples belonging to the compound idx. See hubverse documentation on samples for more details.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_tbl_spl_non_compound_tid`
Check model output data tbl samples contain single unique combination of non-compound task ID values across all samples

---

Description

Check model output data tbl samples contain single unique combination of non-compound task ID values across all samples

Usage

```
check_tbl_spl_non_compound_tid(
  tbl_chr,
  round_id,
  file_path,
  hub_path,
  compound_taskid_set = NULL,
  derived_task_ids = get_hub_derived_task_ids(hub_path, round_id)
)
```

Arguments

`tbl_chr`	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
`round_id`	character string. The round identifier.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`compound_taskid_set`	a list of `compound_taskid_set`s (characters vector of compound task IDs), one for each modeling task. Used to override the compound task ID set in the config file, for example, when validating coarser samples.

`derived_task_ids`
: Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore during validation. Defaults to extracting derived task IDs from hub `task.json`. See `get_hub_derived_task_ids()` for more details.

Details

Output of the check includes an `errors` element, a list of items, one for each modeling task containing samples failing validation, with the following structure:

- `mt_id`: Index identifying the config modeling task the samples are associated with.
- `output_type_ids`: The output type IDs of samples that do not match the most frequent non-compound task ID value combination across all samples in the modeling task.
- `frequent`: The most frequent non-compound task ID value combination across all samples in the modeling task to which all samples were compared. See hubverse documentation on samples for more details.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_tbl_unique_round_id`
: *Check model output data tbl contains a single unique round ID.*

Description

Check model output data tbl contains a single unique round ID.

Usage

```
check_tbl_unique_round_id(tbl, file_path, hub_path, round_id_col = NULL)
```

Arguments

`tbl`	a tibble/data.frame of the contents of the file being validated.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.

hub_path — Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.

round_id_col — Character string. The name of the column containing round_ids. Usually, the value of round property round_id in hub tasks.json config file. Defaults to NULL and determined from the config if applicable.

Details

This check only applies to files being submitted to rounds where round_id_from_variable: true or where a round_id_col name is explicitly provided. Skipped otherwise.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_error> condition class object.

If round_id_from_variable: false and no round_id_col name is provided, check is skipped and a <message/check_info> condition class object is returned. If no valid round_id_col name is provided or can extracted from config (check through check_valid_round_id_col), a <message/check_error> condition class object is returned and the rest of the check skipped.

---

check_tbl_value_col *Check output type values of model output data against config*

Description

Checks that values in the value column of a tibble/data.frame of data read in from the file being validated conform to the configuration for each output type of the appropriate model task.

Usage

```
check_tbl_value_col(
  tbl_chr,
  round_id,
  file_path,
  hub_path,
  derived_task_ids = get_hub_derived_task_ids(hub_path, round_id)
)
```

Arguments

`tbl_chr`	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
`round_id`	character string. The round identifier.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`derived_task_ids`	Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore during validation. Defaults to extracting derived task IDs from hub `task.json`. See `get_hub_derived_task_ids()` for more details.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_tbl_value_col_ascending`

Check that `quantile` *and* `cdf` *output type values of model output data are non-descending*

Description

Checks that values in the `value` column for `quantile` and `cdf` output type data for each unique task ID/output type combination are non-descending when arranged by increasing `output_type_id` order. Check only performed if `tbl_chr` contains `quantile` or `cdf` output type data. If not, the check is skipped and a `<message/check_info>` condition class object is returned.

Usage

```
check_tbl_value_col_ascending(
  tbl_chr,
  file_path,
  hub_path,
  round_id,
  derived_task_ids = get_hub_derived_task_ids(hub_path, round_id)
)
```

Arguments

`tbl_chr`	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`round_id`	character string. The round identifier.
`derived_task_ids`	Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore during validation. Defaults to extracting derived task IDs from hub `task.json`. See `get_hub_derived_task_ids()` for more details.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

---

check_tbl_value_col_sum1

Check that `pmf` *output type values of model output data sum to 1.*

Description

Checks that values in the `value` column of `pmf` output type data for each unique task ID combination sum to 1. Check only performed if `tbl` contains `pmf` output type data. If not, the check is skipped and a `<message/check_info>` condition class object is returned.

Usage

```
check_tbl_value_col_sum1(tbl, file_path)
```

Arguments

tbl	a tibble/data.frame of the contents of the file being validated.
file_path	character string. Path to the file being validated relative to the hub's model-output directory.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_failure> condition class object.

Returned object also inherits from subclass <hub_check>.

---

check_tbl_values *Check model output data tbl contains valid value combinations*

Description

Check model output data tbl contains valid value combinations

Usage

```
check_tbl_values(
  tbl_chr,
  round_id,
  file_path,
  hub_path,
  derived_task_ids = deprecated()
)
```

Arguments

tbl_chr	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
round_id	character string. The round identifier.
file_path	character string. Path to the file being validated relative to the hub's model-output directory.
hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.

derived_task_ids
: [**Deprecated**] Derived task ID columns are validated like any other task ID column, so the argument has no effect.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_error> condition class object.

Returned object also inherits from subclass <hub_check>.

---

check_tbl_values_required

Check all required task ID/output type/output type ID value combinations present in model data.

Description

Check all required task ID/output type/output type ID value combinations present in model data.

Usage

```
check_tbl_values_required(
  tbl_chr,
  round_id,
  file_path,
  hub_path,
  derived_task_ids = get_hub_derived_task_ids(hub_path, round_id)
)
```

Arguments

tbl_chr	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
round_id	character string. The round identifier.
file_path	character string. Path to the file being validated relative to the hub's model-output directory.
hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.

`derived_task_ids`
: Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore during validation. Defaults to extracting derived task IDs from hub `task.json`. See `get_hub_derived_task_ids()` for more details.

Details

Note that it is **necessary for `derived_task_ids` to be specified if any of the task IDs with `required` values have dependent derived task IDs**. If this is the case and derived task IDs are not specified, the dependent nature of derived task ID values will result in **false validation errors when validating required values**.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

---

check_valid_round_id *Check whether the* `round_id` *determined for the submission is valid*

Description

Check whether the `round_id` determined for the submission is valid

Usage

```
check_valid_round_id(round_id, file_path, hub_path = ".")
```

Arguments

`round_id`	character string. The round identifier.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_error>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`check_valid_round_id_col` *Check that any round_id_col name provided or extracted from the hub config is valid.*

Description

Check that any round_id_col name provided or extracted from the hub config is valid.

Usage

```
check_valid_round_id_col(tbl, file_path, hub_path, round_id_col = NULL)
```

Arguments

`tbl`	a tibble/data.frame of the contents of the file being validated.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`round_id_col`	Character string. The name of the column containing `round_id`s. Usually, the value of round property `round_id` in hub `tasks.json` config file. Defaults to `NULL` and determined from the config if applicable.

Details

This check only applies to files being submitted to rounds where `round_id_from_variable: true` or where a `round_id_col` name is explicitly provided. Skipped otherwise.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

If `round_id_from_variable: false` and no `round_id_col` name is provided, check is skipped and a `<message/check_info>` condition class object is returned. Returned object also inherits from subclass `<hub_check>`.

`combine` *Concatenate validation objects*

Description

Combines multiple validation objects of the same class into one. Works with both single-file validation objects (`hub_validations`, `target_validations`) and multi-file collection objects (`hub_validations_collection`, `target_validations_collection`). For more details on these classes, see article on `<hub_validations>` S3 class objects.

Usage

```
combine(...)

## S3 method for class 'hub_validations_collection'
combine(...)
```

Arguments

`...`	Validation objects to be concatenated. All objects must be of the same class. NULL values are ignored. Empty objects are filtered out when combining multiple inputs, but a single empty input is returned as-is.

Details

For `hub_validations` objects, all inputs must share the same `where` attribute (i.e., be validations for the same subject).

For `hub_validations_collection` objects, the individual `hub_validations` objects from all collections are extracted and grouped by their `where` attribute, combining validation results for the same subject.

Subclasses (e.g., `target_validations`, `target_validations_collection`) are preserved.

Value

An object of the same class as the inputs, or NULL if no valid inputs provided.

See Also

new_hub_validations(), new_hub_validations_collection(), new_target_validations(), new_target_validations_collection()

`create_custom_check` *Create a custom validation check function template file.*

Description

Create a custom validation check function template file.

Usage

```
create_custom_check(
  name,
  hub_path = ".",
  r_dir = "src/validations/R",
  error = FALSE,
  conditional = FALSE,
  error_object = FALSE,
  config = FALSE,
  extra_args = FALSE,
  overwrite = FALSE
)
```

Arguments

`name`	Character string. Name of the custom check function. We recommend following the hubValidations package naming convention. For more details, consult the article on writing custom check functions.
`hub_path`	Character string. Path to the hub directory. Default is the current working directory.
`r_dir`	Character string. Path (relative to `hub_path`) to the directory the custom check function file will be written to. Default is `src/validations/R` which is the recommended directory for storing custom check functions.
`error`	Logical. Defaults to `FALSE`, which will return a `<error/check_failure>` class object in the case of a failed check. Set this to `TRUE` if your custom check function is required to pass for other custom checks to be performed; in the case of a failed check, the custom check function will then return an `<error/check_error>` class object and cause custom validations to return early. Note that in the case of custom validations, executions errors in custom functions will also result in custom validations returning early.
`conditional`	Logical. If `TRUE`, the custom check function template will include a block of code to check a condition before running the check. This is useful when a check may need to be skipped based on a condition.

`error_object` Logical. If `TRUE`, the custom check function template will include an error object that can be used to store additional information about the properties of the object being checked that caused check failure. For example, it could store the index of rows in a `tbl` that caused a check failure.

`config` Logical. If `TRUE`, the custom check function template will include `hub_path` as a function argument and a block of code for reading in the hub `tasks.json` config file.

`extra_args` Logical. If `TRUE`, the custom check function template will include an `extra_arg` template function argument and template block of code to check the input arguments of the custom check function.

`overwrite` Logical. If `TRUE`, the function will overwrite an existing

Details

See the article on writing custom check functions for more.

Value

Invisible `TRUE` if the custom check function file is created successfully.

Examples

```
withr::with_tempdir({
  # Create the custom check file with default settings.
  create_custom_check("check_default")
  cat(readLines("src/validations/R/check_default.R"), sep = "\n")

  # Create fully featured custom check file.
  create_custom_check("check_full",
    error = TRUE, conditional = TRUE,
    error_object = TRUE, config = TRUE,
    extra_args = TRUE
  )
  cat(readLines("src/validations/R/check_full.R"), sep = "\n")
})
```

`expand_model_out_grid` *Create expanded grid of valid task ID and output type value combinations*

Description

Create expanded grid of valid task ID and output type value combinations

Usage

```
expand_model_out_grid(
  config_tasks,
  round_id,
  required_vals_only = FALSE,
  force_output_types = FALSE,
  all_character = FALSE,
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  as_arrow_table = FALSE,
  bind_model_tasks = TRUE,
  include_sample_ids = FALSE,
  compound_taskid_set = NULL,
  output_types = NULL,
  derived_task_ids = get_config_derived_task_ids(config_tasks, round_id)
)
```

Arguments

`config_tasks` a list version of the content's of a hub's `tasks.json` config file, accessed through the `"config_tasks"` attribute of a `<hub_connection>` object or function `hubUtils::read_config()`.

`round_id` Character string. Round identifier. If the round is set to `round_id_from_variable: true`, IDs are values of the task ID defined in the round's `round_id` property of `config_tasks`. Otherwise should match round's `round_id` value in config. Ignored if hub contains only a single round.

`required_vals_only` Logical. Whether to return only combinations of Task ID and related output type ID required values.

`force_output_types` Logical. Whether to force all output types to be required. If `TRUE`, all output type ID values are treated as required regardless of the value of the `is_required` property. Useful for creating grids of required values for optional output types.

`all_character` Logical. Whether to return all character column.

`output_type_id_datatype` character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json` config file if available. If the property is not set in the config, the argument falls back to `"auto"` which determines the `output_type_id` data type automatically from the `tasks.json` config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where `output_type_id`s are `NA`,) are being collected by a hub, the `output_type_id` column is assigned a `character` data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce `output_type_id` to a data type that is not valid for

the data (e.g. trying to coerce`"character"` values to `"double"`) will likely result in an error or potentially unexpected behaviour so use with care.

`as_arrow_table`
: Logical. Whether to return an arrow table. Defaults to `FALSE`.

`bind_model_tasks`
: Logical. Whether to bind expanded grids of values from multiple modeling tasks into a single tibble/arrow table or return a list.

`include_sample_ids`
: Logical. Whether to include sample identifiers in the `output_type_id` column.

`compound_taskid_set`
: List of character vectors, one for each modeling task in the round. Can be used to override the compound task ID set defined in the config. If `NULL` is provided for a given modeling task, a compound task ID set of all task IDs is used.

`output_types`
: Character vector of output type names to include. Use to subset for grids for specific output types.

`derived_task_ids`
: Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore. Columns for such task ids will contain `NA`s. Defaults to extracting derived task IDs from `config_tasks`. See `get_config_derived_task_ids()` for more details.

Details

When a round is set to `round_id_from_variable: true`, the value of the task ID from which round IDs are derived (i.e. the task ID specified in `round_id` property of `config_tasks`) is set to the value of the `round_id` argument in the returned output.

When sample output types are included in the output and `include_sample_ids = TRUE`, the `output_type_id` column contains example sample indexes which are useful for identifying the compound task ID structure of multivariate sampling distributions in particular, i.e. which combinations of task ID values represent individual samples.

Value

If `bind_model_tasks = TRUE` (default) a tibble or arrow table containing all possible task ID and related output type ID value combinations. If `bind_model_tasks = FALSE`, a list containing a tibble or arrow table for each round modeling task.

Columns are coerced to data types according to the hub schema, unless `all_character = TRUE`. If `all_character = TRUE`, all columns are returned as character which can be faster when large expanded grids are expected. If `required_vals_only = TRUE`, values are limited to the combinations of required values only.

Note that if `required_vals_only = TRUE` and an optional output type is requested through `output_types`, a zero row grid will be returned. If all output types are requested however (i.e. when `output_types = NULL`) and they are all optional, a grid of required task ID values only will be returned. However, whenever `force_output_types = TRUE`, all output types are treated as required.

Examples

```
hub_con <- hubData::connect_hub(
  system.file("testhubs/flusight", package = "hubUtils")
)
config_tasks <- attr(hub_con, "config_tasks")
expand_model_out_grid(config_tasks, round_id = "2023-01-02")
expand_model_out_grid(
  config_tasks,
  round_id = "2023-01-02",
  required_vals_only = TRUE
)
# Specifying a round in a hub with multiple round configurations.
hub_con <- hubData::connect_hub(
  system.file("testhubs/simple", package = "hubUtils")
)
config_tasks <- attr(hub_con, "config_tasks")
expand_model_out_grid(config_tasks, round_id = "2022-10-01")
# Later round_id maps to round config that includes additional task ID 'age_group'.
expand_model_out_grid(config_tasks, round_id = "2022-10-29")
# Coerce all columns to character
expand_model_out_grid(config_tasks,
  round_id = "2022-10-29",
  all_character = TRUE
)
# Return arrow table
expand_model_out_grid(config_tasks,
  round_id = "2022-10-29",
  all_character = TRUE,
  as_arrow_table = TRUE
)
# Hub with sample output type
config_tasks <- read_config_file(system.file("config", "tasks.json",
  package = "hubValidations"
))
expand_model_out_grid(config_tasks,
  round_id = "2022-12-26"
)
# Include sample IDS
expand_model_out_grid(config_tasks,
  round_id = "2022-12-26",
  include_sample_ids = TRUE
)
# Hub with sample output type and compound task ID structure
config_tasks <- read_config_file(
  system.file("config", "tasks-comp-tid.json", package = "hubValidations")
)
expand_model_out_grid(config_tasks,
  round_id = "2022-12-26",
  include_sample_ids = TRUE
)
# Override config compound task ID set
# Create coarser compound task ID set for the first modeling task which contains
```

```
# samples
expand_model_out_grid(config_tasks,
  round_id = "2022-12-26",
  include_sample_ids = TRUE,
  compound_taskid_set = list(
    c("forecast_date", "target"),
    NULL
  )
)
expand_model_out_grid(config_tasks,
  round_id = "2022-12-26",
  include_sample_ids = TRUE,
  compound_taskid_set = list(
    NULL,
    NULL
  )
)
# Subset output types
config_tasks <- read_config(
  system.file("testhubs", "samples", package = "hubValidations")
)
expand_model_out_grid(config_tasks,
  round_id = "2022-10-29",
  include_sample_ids = TRUE,
  bind_model_tasks = FALSE,
  output_types = c("sample", "pmf"),
)
expand_model_out_grid(config_tasks,
  round_id = "2022-10-29",
  include_sample_ids = TRUE,
  bind_model_tasks = TRUE,
  output_types = "sample",
)
# Ignore derived task IDs
expand_model_out_grid(config_tasks,
  round_id = "2022-10-29",
  include_sample_ids = TRUE,
  bind_model_tasks = FALSE,
  output_types = "sample",
  derived_task_ids = "target_end_date"
)
# Return only required values
hub_path <- system.file("testhubs", "v4", "simple", package = "hubUtils")
config_tasks <- read_config(hub_path)
# Return required output types and output_types_ids only
expand_model_out_grid(
  config_tasks = config_tasks,
  round_id = "2022-10-22",
  required_vals_only = TRUE
)
# Force all output types to be required
expand_model_out_grid(
  config_tasks = config_tasks,
```

```
  round_id = "2022-10-22",
  required_vals_only = TRUE,
  force_output_types = TRUE
)
# Sub-setting for an optional output type returns an empty data frame
expand_model_out_grid(
  config_tasks = config_tasks,
  round_id = "2022-10-22",
  output_types = "mean",
  required_vals_only = TRUE
)
# force_output_types on an optional output type forces all output_type_id values
# to be required
expand_model_out_grid(
  config_tasks = config_tasks,
  round_id = "2022-10-22",
  output_types = "mean",
  required_vals_only = TRUE,
  force_output_types = TRUE
)
# Ignore derived task IDs
hub_path <- system.file("testhubs", "v4", "flusight", package = "hubUtils")
config_tasks <- read_config(hub_path)
# Defaults to using derived_task_ids from config
expand_model_out_grid(config_tasks, round_id = "2023-05-08")
# Can be overridden by argument derived_task_ids
expand_model_out_grid(config_tasks,
  round_id = "2023-05-08",
  derived_task_ids = NULL
)
```

`get_config_derived_task_ids` *Get hub configuration fields from a* `<config>` *class object*

Description

Get hub configuration fields from a `<config>` class object

Usage

```
get_config_derived_task_ids(config_tasks, round_id = NULL)
```

Arguments

`config_tasks`	a list version of the content's of a hub's `tasks.json` config file, accessed through the `"config_tasks"` attribute of a `<hub_connection>` object or function `hubUtils::read_config()`.

round_id Character string. Round identifier. If the round is set to `round_id_from_variable: true`, IDs are values of the task ID defined in the round's `round_id` property of `config_tasks`. Otherwise should match round's `round_id` value in config. Ignored if hub contains only a single round.

Value

- `get_config_derived_task_ids`: character vector of hub or round level derived task ID names. If `round_id` is `NULL` or the round does not have a round level `derived_tasks_ids` setting, returns the hub level `derived_tasks_ids` setting.

Functions

- `get_config_derived_task_ids()`: Get the hub or round level `derived_tasks_ids`

Examples

```
hub_path <- system.file("testhubs/v4/flusight", package = "hubUtils")
config_tasks <- read_config(hub_path)
get_config_derived_task_ids(config_tasks)
get_config_derived_task_ids(config_tasks, round_id = "2023-05-08")
```

---

get_target_task_id *Get Unique Target Task ID*

Description

Retrieves the unique target task ID by extracting all target metadata and extracting the names of their `target_keys`. For valid config files these should be the same across all rounds and model tasks.

Usage

```
get_target_task_id(config_tasks)
```

Arguments

config_tasks a list representation of the `tasks.json` config file.

Value

A character vector of unique target task ID names. Post v5.0.0 this should be a single task ID name.

`get_tbl_compound_taskid_set` *Detect the compound_taskid_set for a tbl for each modeling task in a given round.*

Description

Detect the compound_taskid_set for a tbl for each modeling task in a given round.

Usage

```
get_tbl_compound_taskid_set(
  tbl_chr,
  config_tasks,
  round_id,
  compact = TRUE,
  error = TRUE,
  derived_task_ids = get_config_derived_task_ids(config_tasks, round_id)
)
```

Arguments

`tbl_chr`	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**.
`config_tasks`	a list representantion of the `tasks.json` config file.
`round_id`	Character string. The round ID.
`compact`	Logical. If TRUE, the output will be compacted to remove NULL elements.
`error`	Logical. If TRUE, an error will be thrown if the compound task ID set is not valid. If FALSE and an error is detected, the detected compound task ID set will be returned with error attributes attached.
`derived_task_ids`	Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore. Columns for such task ids will contain `NA`s. Defaults to extracting derived task IDs from `config_tasks`. See `get_config_derived_task_ids()` for more details.

Value

A list of vectors of compound task IDs detected in the tbl, one for each modeling task in the round. If `compact` is TRUE, modeling tasks returning NULL elements will be removed.

Examples

```
hub_path <- system.file("testhubs/samples", package = "hubValidations")
file_path <- "flu-base/2022-10-22-flu-base.csv"
round_id <- "2022-10-22"
tbl_chr <- read_model_out_file(
  file_path = file_path,
  hub_path = hub_path,
  coerce_types = "chr"
)
config_tasks <- read_config(hub_path, "tasks")
get_tbl_compound_taskid_set(tbl_chr, config_tasks, round_id)
get_tbl_compound_taskid_set(tbl_chr, config_tasks, round_id,
  compact = FALSE
)
```

---

is_success *Get status of a hub check*

Description

Get status of a hub check

Usage

```
is_success(x)

is_failure(x)

is_error(x)

is_info(x)

not_pass(x)

is_exec_error(x)

is_exec_warn(x)

is_any_error(x)
```

Arguments

x	an object that inherits from class `<hub_check>` to test.

Value

Logical. Is given status of check TRUE?

Functions

- `is_success()`: Is check success?
- `is_failure()`: Is check failure?
- `is_error()`: Is check error?
- `is_info()`: Is check info?
- `not_pass()`: Did check not pass?
- `is_exec_error()`: Is exec error?
- `is_exec_warn()`: Is exec warning?
- `is_any_error()`: Is error or exec error?

`match_tbl_to_model_task` *Match model output data to their model tasks in* `config_tasks`.

Description

Useful for performing model task specific checks on model output.

Usage

```
match_tbl_to_model_task(
  tbl_chr,
  config_tasks,
  round_id,
  output_types = NULL,
  derived_task_ids = get_config_derived_task_ids(config_tasks, round_id),
  order_by_config = FALSE
)
```

Arguments

`tbl_chr`	a tibble/data.frame of the contents of the file being validated. Column types must **all be character**: the config's values are converted to character when they are extracted, and are compared against this table without further conversion. Every task ID column the round defines must be present.
`config_tasks`	a list version of the content's of a hub's `tasks.json` config file, accessed through the `"config_tasks"` attribute of a `<hub_connection>` object or function `hubUtils::read_config()`.
`round_id`	Character string. Round identifier. If the round is set to `round_id_from_variable: true`, IDs are values of the task ID defined in the round's `round_id` property of `config_tasks`. Otherwise should match round's `round_id` value in config. Ignored if hub contains only a single round.

`output_types` Character vector of output type names to include. Use to subset for grids for specific output types.

`derived_task_ids`
Character vector of derived task ID names, or `NULL` for none. A derived task ID's value follows from the values of other task IDs. A derived task ID cannot therefore further distinguish a row beyond the values of the task IDs it is derived from. Derived task ID columns are skipped, and returned unchanged.

`order_by_config`
Logical. How to order each modeling task's rows. `FALSE`, the default, leaves them in the order they were submitted in. `TRUE` sorts them into the order the config lists their values in: on `output_type` first, so rows of one output type sit together, then on `output_type_id`, so they ascend within each, then on the task IDs to break ties.

What is sorted on is each value's position in the config, not the value itself. `pmf` categories show why that matters: `"low"`, `"moderate"` and `"high"` have no useful alphabetical order, but the config lists them in the order they belong in. `check_tbl_value_col_ascending()` asks for this order, because it reads values in the order the rows arrive in.

In the config a task ID's values are split into `required` and `optional`. Extracting them collapses the two into a single order, `required` values first, then `optional` ones, and that is the order sorted on.

Sample rows are ordered by their task ID values only.

Details

Sample `output_type_id` values are returned as submitted. The submitter chooses them, so this function does not check them against the config.

Value

A list with one element per model task in the round, each a `tbl_df` of the model output rows matched to that model task. A model task that offers none of the requested `output_types` gets `NULL`. Rows that match no model task are not returned.

Examples

```
hub_path <- system.file("testhubs/samples", package = "hubValidations")
tbl_chr <- read_model_out_file(
  file_path = "flu-base/2022-10-22-flu-base.csv",
  hub_path, coerce_types = "chr"
)
config_tasks <- read_config(hub_path, "tasks")
match_tbl_to_model_task(tbl_chr, config_tasks, round_id = "2022-10-22")
match_tbl_to_model_task(tbl_chr, config_tasks,
  round_id = "2022-10-22",
  output_types = "sample"
)
```

`new_hub_validations` *Create new or convert list to* `hub_validations` *S3 class object*

Description

A `hub_validations` object contains validation results for a single validation subject. Depending on context, this could be a file, a configuration directory (`hub-config`), or a target dataset type (`time-series`, `oracle-output`). All checks must have the same `$where` value, which is extracted and stored as the `where` attribute.

Usage

```
new_hub_validations(...)

as_hub_validations(x)
```

Arguments

`...`	named elements to be included. Each element must be an object which inherits from class `<hub_check>`. All checks must have the same `$where` value.
`x`	a list of named elements. Each element must be an object which inherits from class `<hub_check>`. All checks must have the same `$where` value.

Value

an S3 object of class `<hub_validations>` with a `where` attribute.

Functions

- `new_hub_validations()`: Create new `<hub_validations>` S3 class object
- `as_hub_validations()`: Convert list to `<hub_validations>` S3 class object

Examples

```
new_hub_validations()

hub_path <- system.file("testhubs/simple", package = "hubValidations")
file_path <- "team1-goodmodel/2022-10-08-team1-goodmodel.csv"
new_hub_validations(
  file_exists = check_file_exists(file_path, hub_path),
  file_name = check_file_name(file_path)
)
x <- list(
  file_exists = check_file_exists(file_path, hub_path),
  file_name = check_file_name(file_path)
)
as_hub_validations(x)
```

`new_hub_validations_collection` *Create new or convert list to* `hub_validations_collection` *S3 class object*

Description

A `hub_validations_collection` is a container for validation results from multiple validation subjects. It is a named list where each element is a `hub_validations` object. Names are automatically extracted from the `where` attribute of each `hub_validations` object. If multiple `hub_validations` objects have the same `where` value, they are merged using `combine()`. Empty `hub_validations` objects are ignored.

Usage

```
new_hub_validations_collection(...)

as_hub_validations_collection(x)
```

Arguments

`...`	`hub_validations` objects to be included.
`x`	a list where each element is a `hub_validations` object.

Value

an S3 object of class `<hub_validations_collection>`. Elements are named by their `where` attribute (e.g., `collection[["path/to/file.csv"]]`).

Functions

- `new_hub_validations_collection()`: Create new `<hub_validations_collection>` S3 class object
- `as_hub_validations_collection()`: Convert list to `<hub_validations_collection>` S3 class object

Examples

```
new_hub_validations_collection()

hub_path <- system.file("testhubs/simple", package = "hubValidations")

# Create validations for two different files
file_path_1 <- "team1-goodmodel/2022-10-08-team1-goodmodel.csv"
validations_1 <- new_hub_validations(
  file_exists = check_file_exists(file_path_1, hub_path),
  file_name = check_file_name(file_path_1)
)
```

```
file_path_2 <- "team1-goodmodel/2022-10-15-team1-goodmodel.csv"
validations_2 <- new_hub_validations(
  file_exists = check_file_exists(file_path_2, hub_path),
  file_name = check_file_name(file_path_2)
)

# Combine into a collection
collection <- new_hub_validations_collection(validations_1, validations_2)

# Print the collection
collection

# Get file paths (element names)
names(collection)

# Access validations for a specific file
collection[[file_path_1]]

# Access validations for a specific file and check
collection$`team1-goodmodel/2022-10-08-team1-goodmodel.csv`$file_exists
```

`new_target_validations` *Create new or convert list to* `target_validations` *S3 class object*

Description

A `target_validations` object contains validation results for a single validation subject. Depending on context, this could be a target data file, the hub configuration directory (`hub-config`), or a target dataset type (`time-series`, `oracle-output`). All checks must have the same `$where` value, which is extracted and stored as the `where` attribute.

Usage

```
new_target_validations(...)

as_target_validations(x)
```

Arguments

`...`	named elements to be included. Each element must be an object which inherits from class `<hub_check>`. All checks must have the same `$where` value.
`x`	a list of named elements. Each element must be an object which inherits from class `<hub_check>`. All checks must have the same `$where` value.

Value

an S3 object of class `<target_validations>` with a `where` attribute.

Functions

- `new_target_validations()`: Create new `<target_validations>` S3 class object
- `as_target_validations()`: Convert list to `<target_validations>` S3 class object

Examples

```
new_target_validations()

hub_path <- system.file("testhubs/v5/target_file", package = "hubUtils")
file_path <- "time-series.csv"
new_target_validations(
  target_file_name = check_target_file_name(file_path),
  target_file_ext_valid = check_target_file_ext_valid(file_path)
)
x <- list(
  target_file_name = check_target_file_name(file_path),
  target_file_ext_valid = check_target_file_ext_valid(file_path)
)
as_target_validations(x)
file_path <- "time-series/target=flu_hosp_rate/part-0.parquet"
new_target_validations(
  target_file_name = check_target_file_name(file_path),
  target_file_ext_valid = check_target_file_ext_valid(file_path)
)
```

---

new_target_validations_collection

Create new or convert list to `target_validations_collection` *S3 class object*

Description

A `target_validations_collection` is a container for target validation results from multiple validation subjects. It is a named list where each element is a `target_validations` object. Names are automatically extracted from the `where` attribute of each `target_validations` object. If multiple `target_validations` objects have the same `where` value, they are merged using `combine()`. Empty `target_validations` objects are ignored.

Usage

```
new_target_validations_collection(...)

as_target_validations_collection(x)
```

Arguments

...	`target_validations` objects to be included.
x	a list where each element is a `target_validations` object.

Value

an S3 object of class `<target_validations_collection>`. Elements are named by their `where` attribute (e.g., `collection[["path/to/file.csv"]]`).

Functions

- `new_target_validations_collection()`: Create new `<target_validations_collection>` S3 class object
- `as_target_validations_collection()`: Convert list to `<target_validations_collection>` S3 class object

Examples

```
new_target_validations_collection()

# Create validations for two different files
file_path_1 <- "time-series.csv"
validations_1 <- new_target_validations(
  target_file_name = check_target_file_name(file_path_1),
  target_file_ext_valid = check_target_file_ext_valid(file_path_1)
)

file_path_2 <- "other-data.csv"
validations_2 <- new_target_validations(
  target_file_name = check_target_file_name(file_path_2),
  target_file_ext_valid = check_target_file_ext_valid(file_path_2)
)

# Combine into a collection
collection <- new_target_validations_collection(validations_1, validations_2)

# Print the collection
collection

# Get file paths (element names)
names(collection)

# Access validations for a specific file
collection[[file_path_1]]

# Access a specific check within a file's validations
collection[["time-series.csv"]]$target_file_name
```

`opt_check_metadata_team_max_model_n`
Check that submitting team does not exceed maximum number of allowed models per team

---

Description

Check that submitting team does not exceed maximum number of allowed models per team

Usage

```
opt_check_metadata_team_max_model_n(file_path, hub_path, n_max = 2L)
```

Arguments

`file_path`	character string. Path to the file being validated relative to the hub's model-metadata directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`n_max`	Integer. Number of maximum allowed models per team.

Details

Should be deployed as part of `validate_model_metadata` optional checks.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`opt_check_tbl_col_timediff` *Check time difference between values in two date columns equal a defined period.*

Description

Check time difference between values in two date columns equal a defined period.

Usage

```
opt_check_tbl_col_timediff(
  tbl,
  file_path,
  hub_path,
  t0_colname,
  t1_colname,
  timediff = lubridate::weeks(2),
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date")
)
```

Arguments

`tbl`	a tibble/data.frame of the contents of the file being validated.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`t0_colname`	Character string. The name of the time zero date column.
`t1_colname`	Character string. The name of the time zero + 1 time step date column.
`timediff`	an object of class `lubridate::Period` and length 1.
`output_type_id_datatype`	character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json` config file if available. If the property is not set in the config, the argument falls back to `"auto"` which determines the `output_type_id` data type automatically from the `tasks.json` config file as the simplest data type required to represent all output type ID values across all output types in the

hub. When only point estimate output types (where `output_type_id`s are `NA`,) are being collected by a hub, the `output_type_id` column is assigned a `character` data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce `output_type_id` to a data type that is not valid for the data (e.g. trying to coerce`"character"` values to `"double"`) will likely result in an error or potentially unexpected behaviour so use with care.

Details

Should be deployed as part of `validate_model_data` optional checks.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

---

opt_check_tbl_counts_lt_popn

Check that predicted values per location are less than total location population.

Description

Check that predicted values per location are less than total location population.

Usage

```
opt_check_tbl_counts_lt_popn(
  tbl,
  file_path,
  hub_path,
  targets = NULL,
  popn_file_path = "auxiliary-data/locations.csv",
  popn_col = "population",
  location_col = "location"
)
```

Arguments

`tbl`	a tibble/data.frame of the contents of the file being validated.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
targets	Either a single target key list or a list of multiple target key lists.
popn_file_path	Character string. Path to population data relative to the hub root. Defaults to auxiliary-data/locations.csv.
popn_col	Character string. The name of the population size column in the population data set.
location_col	Character string. The name of the location column. Used to join population data to submission file data. Must be shared by both files.

Details

Should only be applied to rows containing count predictions. Use argument targets to filter tbl data to appropriate count target rows.

Should be deployed as part of validate_model_data optional checks.

Value

Depending on whether validation has succeeded, one of:

- <message/check_success> condition class object.
- <error/check_failure> condition class object.

Returned object also inherits from subclass <hub_check>.

Examples

```
hub_path <- system.file("testhubs/flusight", package = "hubValidations")
file_path <- "hub-ensemble/2023-05-08-hub-ensemble.parquet"
tbl <- hubValidations::read_model_out_file(file_path, hub_path)
# Single target key list
targets <- list("target" = "wk ahead inc flu hosp")
opt_check_tbl_counts_lt_popn(tbl, file_path, hub_path, targets = targets)
```

---

opt_check_tbl_horizon_timediff

Check time difference between values in two date columns equals a defined time period defined by values in a horizon column

---

Description

Check time difference between values in two date columns equals a defined time period defined by values in a horizon column

Usage

```
opt_check_tbl_horizon_timediff(
  tbl,
  file_path,
  hub_path,
  t0_colname,
  t1_colname,
  horizon_colname = "horizon",
  timediff = lubridate::weeks(),
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date")
)
```

Arguments

`tbl`	a tibble/data.frame of the contents of the file being validated.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`t0_colname`	Character string. The name of the time zero date column.
`t1_colname`	Character string. The name of the time zero + 1 time step date column.
`horizon_colname`	Character string. The name of the horizon column. Defaults to `"horizon"`.
`timediff`	an object of class `lubridate::Period` and length 1. The period of a single horizon. Default to 1 week.
`output_type_id_datatype`	character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json` config file if available. If the property is not set in the config, the argument falls back to `"auto"` which determines the `output_type_id` data type automatically from the `tasks.json` config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where `output_type_ids` are `NA`,) are being collected by a hub, the `output_type_id` column is assigned a `character` data type when auto-determined. Other data type

values can be used to override automatic determination. Note that attempting to coerce `output_type_id` to a data type that is not valid for the data (e.g. trying to coerce`"character"` values to `"double"`) will likely result in an error or potentially unexpected behaviour so use with care.

Details

Should be deployed as part of `validate_model_data` optional checks.

Value

Depending on whether validation has succeeded, one of:

- `<message/check_success>` condition class object.
- `<error/check_failure>` condition class object.

Returned object also inherits from subclass `<hub_check>`.

`parse_file_name` *Parse model output file metadata from file name*

Description

Parse model output file metadata from file name

Usage

```
parse_file_name(file_path, file_type = c("model_output", "model_metadata"))
```

Arguments

`file_path`	Character string. A model output file name. Can include parent directories which are ignored.
`file_type`	Character string. Type of file name being parsed. One of `"model_output"` or `"model_metadata"`.

Details

File names are allowed to contain the following compression extension prefixes: .snappy, .gzip, .gz, .brotli, .zstd, .lz4, .lzo, .bz2. These extension prefixes are now extracted when parsing the file name and returned as `compression_ext` element if present.

Value

A list with the following elements:

- `round_id`: The round ID the model output is associated with (`NA` for model metadata files.)
- `team_abbr`: The team responsible for the model.
- `model_abbr`: The name of the model.
- `model_id`: The unique model ID derived from the concatenation of `<team_abbr>-<model_abbr>`.
- `ext`: The file extension.
- `compression_ext`: optional. The compression extension if present.

Examples

```
parse_file_name("hub-baseline/2022-10-15-hub-baseline.csv")
parse_file_name("hub-baseline/2022-10-15-hub-baseline.gzip.parquet")
```

---

print.hub_validations

Print results of `validate_...()` *function as a bullet list*

Description

Prints a formatted summary of validation results. Validation-level warnings (attached to the `hub_validations` object) are always displayed prominently in a box at the top. Check-level warnings (attached to individual checks) are only shown when `show_check_warnings = TRUE`.

Usage

```
## S3 method for class 'hub_validations'
print(x, show_check_warnings = FALSE, ...)
```

Arguments

`x`	An object of class `hub_validations` or any of its subclasses.
`show_check_warnings`	Logical. If `TRUE`, prints check-level warnings inline with their checks. Validation-level warnings are always printed. Default `FALSE`.
`...`	Unused argument present for class consistency

Value

Returns `x` invisibly.

Examples

```
hub_path <- system.file("testhubs/simple", package = "hubValidations")
v <- validate_submission(
  hub_path,
  file_path = "team1-goodmodel/2022-10-08-team1-goodmodel.csv"
)

# Default print
print(v)

# Show check-level warnings (if any)
print(v, show_check_warnings = TRUE)

# Example with validation-level warning
v_with_warning <- v
attr(v_with_warning, "warnings") <- list(
  capture_validation_warning(
    msg = "Example validation-level warning message.",
    where = "example"
  )
)
print(v_with_warning)
```

---

print.hub_validations_collection

Print results of multi-file validation as a hierarchical bullet list

---

Description

Prints a formatted summary of validation results from multiple files. Each file's validations are printed under a header showing the file path. Validation-level warnings (attached to the collection object) are displayed prominently in a box at the top.

Usage

```
## S3 method for class 'hub_validations_collection'
print(x, show_check_warnings = FALSE, ...)
```

Arguments

`x`	An object of class `hub_validations_collection` or its subclasses.
`show_check_warnings`	Logical. If `TRUE`, prints check-level warnings inline with their checks. Validation-level warnings are always printed. Default `FALSE`.
`...`	Unused argument present for class consistency

Value

Returns `x` invisibly.

`read_model_out_file` *Read a model output file*

Description

Read a model output file

Usage

```
read_model_out_file(
  file_path,
  hub_path = ".",
  coerce_types = c("hub", "chr", "none"),
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date")
)
```

Arguments

`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`coerce_types`	character. What to coerce column types to on read. • `hub`: (default) read in (`csv`) or coerce (`parquet`, `arrow`) to hub schema. When coercing data types using the `hub` schema, the `output_type_id_datatype` can also be used to set the `output_type_id` column data type manually. • `chr`: read in (`csv`) or coerce (`parquet`, `arrow`) all columns to character. • `none`: No coercion. Use `arrow` `read_*` function defaults.
`output_type_id_datatype`	character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json` config file if available. If the property is not set in the config, the argument falls back to `"auto"` which determines the `output_type_id` data type automatically from the `tasks.json` config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where `output_type_ids`

are NA,) are being collected by a hub, the output_type_id column is assigned a character data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce output_type_id to a data type that is not valid for the data (e.g. trying to coerce"character" values to "double") will likely result in an error or potentially unexpected behaviour so use with care.

Value

a tibble of contents of the model output file.

`read_target_file` *Read a single target data file*

Description

Read a single target data file

Usage

```
read_target_file(
  target_file_path,
  hub_path,
  coerce_types = c("target", "chr", "none"),
  date_col = NULL,
  na = c("NA", "")
)
```

Arguments

`target_file_path`
: Character string. Path to the target data file being validated relative to the hub's target-data directory.

`hub_path`
: Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.

`coerce_types`
: character string. What to coerce column types to on read.

- target: (default) read in (csv) or coerce (parquet) to expected schema by target type (See hubData::create_timeseries_schema() and hubData::create_oracle_output_schema() for details). When coercing data types using the target schema, the output_type_id_datatype can also be used to set the output_type_id column data type manually.

- `chr`: read in (`csv`) or coerce (`parquet`) all columns to character.
- `none`: No coercion. Use `arrow` `read_*` function defaults.

`date_col` Optional column name to be interpreted as date. Default is `NULL`. Useful when the required date column is a partitioning column in the target data and does not have the same name as a date typed task ID variable in the config.

`na` A character vector of strings to interpret as missing values. Only applies to CSV files. The default is `c("NA", "")`. Useful when actual character string `"NA"` values are used in the data. In such a case, use empty cells to indicate missing values in your files and set `na = ""`.

Value

a tibble of contents of the target data file.

Examples

```
# download example hub
hub_path <- system.file("testhubs/v5/target_file",
  package = "hubUtils"
)
# read in time-series file
read_target_file("time-series.csv", hub_path)
read_target_file("time-series.csv", hub_path, coerce_types = "chr")
# read in oracle-output file
read_target_file("oracle-output.csv", hub_path)
read_target_file("oracle-output.csv", hub_path, coerce_types = "chr")
```

`submission_tmpl` *Create a model output submission file template*

Description

Create a model output submission file template

Usage

```
submission_tmpl(
  path,
  round_id,
  required_vals_only = FALSE,
  force_output_types = FALSE,
  complete_cases_only = TRUE,
  compound_taskid_set = NULL,
  output_types = NULL,
  derived_task_ids = NULL,
```

```
  hub_con = deprecated(),
  config_tasks = deprecated()
)
```

Arguments

`path`	Character string. Can be one of: • a path to a local fully configured hub directory • a path to a local `tasks.json` file. • a URL to the repository of a fully configured hub on GitHub. • a URL to the **raw contents** of a `tasks.json` file on GitHub. • a `<SubTreeFileSystem>` class object pointing to the root of an S3 cloud hub. • a `<SubTreeFileSystem>` class object pointing to a `tasks.json` config file in an S3 cloud hub, relative to the hub's root directory. See examples for more details.
`round_id`	Character string. Round identifier. If the round is set to `round_id_from_variable: true`, IDs are values of the task ID defined in the round's `round_id` property of `config_tasks`. Otherwise should match round's `round_id` value in config. Ignored if hub contains only a single round.
`required_vals_only`	Logical. Whether to return only combinations of Task ID and related output type ID required values.
`force_output_types`	Logical. Whether to force all output types to be required. If `TRUE`, all output type ID values are treated as required regardless of the value of the `is_required` property. Useful for creating grids of required values for optional output types.
`complete_cases_only`	Logical. If `TRUE` (default) and `required_vals_only = TRUE`, only rows with complete cases of combinations of required values are returned. If `FALSE`, rows with incomplete cases of combinations of required values are included in the output.
`compound_taskid_set`	List of character vectors, one for each modeling task in the round. Can be used to override the compound task ID set defined in the config. If `NULL` is provided for a given modeling task, a compound task ID set of all task IDs is used.
`output_types`	Character vector of output type names to include. Use to subset for grids for specific output types.
`derived_task_ids`	Character vector of derived task ID names (task IDs whose values depend on other task IDs) to ignore. Columns for such task ids will contain `NA`s. If `NULL`, defaults to extracting derived task IDs from `config_tasks` or the `config_tasks` attribute of `hub_con`. See `get_config_derived_task_ids()` for more details.

`hub_con` [**Deprecated**] Use `path` instead. A `<hub_connection>` class object.

`config_tasks` [**Deprecated**] Use `path` instead. A list version of the content's of a hub's `tasks.json` config file, accessed through the `"config_tasks"` attribute of a `<hub_connection>` object or function `read_config()`.

Details

For task IDs where all values are optional, by default, columns are created as columns of `NA`s when `required_vals_only = TRUE`. When such columns exist, the function returns a tibble with zero rows, as no complete cases of required value combinations exists. *(Note that determination of complete cases does excludes valid* `NA` `output_type_id` *values in* `"mean"` *and* `"median"` *output types).* To return a template of incomplete required cases, which includes `NA` columns, use `complete_cases_only = FALSE`.

To include output types that are optional in the submission template when `required_vals_only = TRUE` and `complete_cases_only = FALSE`, use `force_output_types = TRUE`. Use this in combination with sub-setting for output types you plan to submit via argument `output_types` to create a submission template customised to your submission plans. *Tip: to ensure you create a template with all required output types, it's a good idea to first run the functions without subsetting or forcing output types and examing the unique values in* `output_type` *to check which output types are required.*

When sample output types are included in the output, the `output_type_id` column contains example sample indexes which are useful for identifying the compound task ID structure of multivariate sampling distributions in particular, i.e. which combinations of task ID values represent individual samples.

When a round is set to `round_id_from_variable: true`, the value of the task ID from which round IDs are derived (i.e. the task ID specified in `round_id` property of `config_tasks`) is set to the value of the `round_id` argument in the returned output.

Value

a tibble template containing an expanded grid of valid task ID and output type ID value combinations for a given submission round and output type. If `required_vals_only = TRUE`, values are limited to the combination of required values only.

Examples

```
hub_path <- system.file("testhubs/flusight", package = "hubUtils")
submission_tmpl(hub_path, round_id = "2023-01-02")
# Return required values only
submission_tmpl(
  hub_path,
  round_id = "2023-01-02",
  required_vals_only = TRUE
)
submission_tmpl(
  hub_path,
  round_id = "2023-01-02",
  required_vals_only = TRUE,
  complete_cases_only = FALSE
```

```
)
# Specify a round in a hub with multiple rounds
hub_path <- system.file("testhubs/simple", package = "hubUtils")
submission_tmpl(hub_path, round_id = "2022-10-01")
submission_tmpl(hub_path, round_id = "2022-10-29")
# Subset for a specific output type
hub_path <- system.file("testhubs", "samples", package = "hubValidations")
submission_tmpl(
  hub_path,
  round_id = "2022-12-17",
  output_types = "sample"
)
# Create a template from the path to a tasks config file
config_path <- system.file("config", "tasks.json",
  package = "hubValidations"
)
submission_tmpl(
  config_path,
  round_id = "2022-12-26"
)
# Hub with sample output type and compound task ID structure
config_path <- system.file("config", "tasks-comp-tid.json",
  package = "hubValidations"
)
submission_tmpl(
  config_path,
  round_id = "2022-12-26",
  output_types = "sample"
)
# Override config compound task ID set
# Create coarser compound task ID set for the first modeling task which contains
# samples
submission_tmpl(
  config_path,
  round_id = "2022-12-26",
  output_types = "sample",
  compound_taskid_set = list(
    c("forecast_date", "target"),
    NULL
  )
)
# Derive a template with ignored derived task ID. Useful to avoid creating
# a template with invalid derived task ID value combinations.
hub_path <- system.file("testhubs", "flusight", package = "hubValidations")
submission_tmpl(
  hub_path,
  round_id = "2022-12-12",
  output_types = "pmf",
  derived_task_ids = "target_end_date",
  complete_cases_only = FALSE
)
# Force optional output type, in this case "mean".
submission_tmpl(
```

```
  hub_path,
  round_id = "2022-12-12",
  required_vals_only = TRUE,
  output_types = c("pmf", "quantile", "mean"),
  force_output_types = TRUE,
  derived_task_ids = "target_end_date",
  complete_cases_only = FALSE
)
# Create a template from a URL to fully configured hub repository on GitHub
submission_tmpl(
  path = "https://github.com/hubverse-org/example-simple-forecast-hub",
  round_id = "2022-11-28",
  output_types = "quantile"
)
# Create a template from a URL to the raw contents of a tasks.json file on
# GitHub
config_raw_url <- paste0(
  "https://raw.githubusercontent.com/hubverse-org/",
  "example-simple-forecast-hub/refs/heads/main/hub-config/tasks.json"
)
submission_tmpl(
  path = config_raw_url,
  round_id = "2022-11-28",
  output_types = "quantile"
)

# Create submission file using config file from AWS S3 bucket hub
# Use `s3_bucket()` to create a path to the hub's root directory
s3_hub_path <- arrow::s3_bucket("hubverse/hubutils/testhubs/simple/")
submission_tmpl(
  path = s3_hub_path,
  round_id = "2022-10-01",
  output_types = "quantile"
)
# Use `path()` method to create a path to the tasks.json file relative to the
# the S3 cloud hub's root directory
s3_config_path <- s3_hub_path$path("hub-config/tasks.json")
submission_tmpl(
  path = s3_config_path,
  round_id = "2022-10-01",
  output_types = "quantile"
)
```

`try_check` *Wrap check expression in try to capture check execution errors*

Description

Wrap check expression in try to capture check execution errors

Usage

```
try_check(expr, file_path)
```

Arguments

expr	check function expression to run.
file_path	character string. Path to the file being validated relative to the hub's model-output directory.

Value

If `expr` executes correctly, the output of `expr` is returned. If execution fails, and object of class `<error/check_exec_error>` is returned. The execution error message is attached as attribute `msg`.

---

validate_model_data *Validate the contents of a submitted model data file*

Description

Validate the contents of a submitted model data file

Usage

```
validate_model_data(
  hub_path,
  file_path,
  round_id_col = NULL,
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  validations_cfg_path = NULL,
  derived_task_ids = NULL
)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
file_path	character string. Path to the file being validated relative to the hub's model-output directory.

round_id_col Character string. The name of the column containing round_ids. Usually, the value of round property round_id in hub tasks.json config file. Defaults to NULL and determined from the config if applicable.

output_type_id_datatype
character string. One of "from_config", "auto", "character", "double", "integer", "logical", "Date". Defaults to "from_config" which uses the setting in the output_type_id_datatype property in the tasks.json config file if available. If the property is not set in the config, the argument falls back to "auto" which determines the output_type_id data type automatically from the tasks.json config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where output_type_ids are NA,) are being collected by a hub, the output_type_id column is assigned a character data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce output_type_id to a data type that is not valid for the data (e.g. trying to coerce"character" values to "double") will likely result in an error or potentially unexpected behaviour so use with care.

validations_cfg_path
Path to validations.yml file. If NULL defaults to hub-config/validations.yml.

derived_task_ids
Character vector of derived task ID names (task IDs whose values depend on other task IDs). The required values and sample checks leave these columns out. If NULL, defaults to extracting derived task IDs from hub task.json. See get_hub_derived_task_ids() for more details.

Details

Note that it is **necessary for derived_task_ids to be specified if any task IDs with required values have dependent derived task IDs**. If this is the case and derived task IDs are not specified, the dependent nature of derived task ID values will result in **false validation errors when validating required values**.

Details of checks performed by validate_model_data()

Value

An object of class hub_validations. Each named element contains a hub_check class object reflecting the result of a given check. Function will return early if a check returns an error. The where attribute is set to file_path.

For more details on the structure of <hub_validations> objects, including how to access more information on individual checks, see article on <hub_validations> S3 class objects.

Examples

```
hub_path <- system.file("testhubs/simple", package = "hubValidations")
file_path <- "team1-goodmodel/2022-10-08-team1-goodmodel.csv"
validate_model_data(hub_path, file_path)
```

validate_model_file *Valid file level properties of a submitted model output file.*

Description

Valid file level properties of a submitted model output file.

Usage

```
validate_model_file(hub_path, file_path, validations_cfg_path = NULL)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
file_path	character string. Path to the file being validated relative to the hub's model-output directory.
validations_cfg_path	Path to validations.yml file. If NULL defaults to hub-config/validations.yml.

Details

Details of checks performed by validate_model_file()

Value

An object of class hub_validations. Each named element contains a hub_check class object reflecting the result of a given check. Function will return early if a check returns an error. The where attribute is set to file_path.

For more details on the structure of <hub_validations> objects, including how to access more information on individual checks, see article on <hub_validations> S3 class objects.

Examples

```
hub_path <- system.file("testhubs/simple", package = "hubValidations")
validate_model_file(hub_path,
  file_path = "team1-goodmodel/2022-10-08-team1-goodmodel.csv"
)
validate_model_file(hub_path,
  file_path = "team1-goodmodel/2022-10-15-team1-goodmodel.csv"
)
```

validate_model_metadata *Valid properties of a metadata file.*

Description

Valid properties of a metadata file.

Usage

```
validate_model_metadata(
  hub_path,
  file_path,
  round_id = "default",
  validations_cfg_path = NULL
)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
file_path	character string. Path to the file being validated relative to the hub's model-output directory.
round_id	character string. The round identifier. Used primarily to indicate whether the "default" or a round specific configuration should be used for custom validations.
validations_cfg_path	Path to validations.yml file. If NULL defaults to hub-config/validations.yml.

Details

Details of checks performed by validate_model_metadata()

Value

An object of class hub_validations. Each named element contains a hub_check class object reflecting the result of a given check. Function will return early if a check returns an error. The where attribute is set to file_path.

For more details on the structure of <hub_validations> objects, including how to access more information on individual checks, see article on <hub_validations> S3 class objects.

Examples

```
hub_path <- system.file("testhubs/simple", package = "hubValidations")
validate_model_metadata(hub_path,
  file_path = "hub-baseline.yml"
)
validate_model_metadata(hub_path,
  file_path = "team1-goodmodel.yaml"
)
```

---

validate_pr *Validate Pull Request*

Description

Validates model output and model metadata files in a Pull Request.

Usage

```
validate_pr(
  hub_path = ".",
  gh_repo,
  pr_number,
  round_id_col = NULL,
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  validations_cfg_path = NULL,
  skip_submit_window_check = FALSE,
  file_modification_check = c("error", "failure", "warn", "message", "none"),
  allow_submit_window_mods = TRUE,
  submit_window_ref_date_from = c("file", "file_path"),
  derived_task_ids = NULL
)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
gh_repo	GitHub repository address in the format username/repo
pr_number	Number of the pull request to validate
round_id_col	Character string. The name of the column containing round_ids. Only required if files contain a column that contains round_id details but has not been configured via round_id_from_variable: true and round_id: in in hub tasks.json config file.

output_type_id_datatype
: character string. One of "from_config", "auto", "character", "double", "integer", "logical", "Date". Defaults to "from_config" which uses the setting in the output_type_id_datatype property in the tasks.json config file if available. If the property is not set in the config, the argument falls back to "auto" which determines the output_type_id data type automatically from the tasks.json config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where output_type_ids are NA,) are being collected by a hub, the output_type_id column is assigned a character data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce output_type_id to a data type that is not valid for the data (e.g. trying to coerce"character" values to "double") will likely result in an error or potentially unexpected behaviour so use with care.

validations_cfg_path
: Path to validations.yml file. If NULL defaults to hub-config/validations.yml.

skip_submit_window_check
: Logical. Whether to skip the submission window check.

file_modification_check
: Character string. Whether to perform check and what to return when modification/deletion of a previously submitted model output file or deletion of a previously submitted model metadata file is detected in PR:

  - "error": Appends a <error/check_error> condition class object for each applicable modified/deleted file.
  - "warning": Appends a <error/check_failure> condition class object for each applicable modified/deleted file.
  - "message": Appends a <message/check_info> condition class object for each applicable modified/deleted file.
  - "none": No modification/deletion checks performed.

allow_submit_window_mods
: Logical. Whether to allow modifications/deletions of model output files within their submission windows. Defaults to TRUE.

submit_window_ref_date_from
: whether to get the reference date around which relative submission windows will be determined from the file's file_path round ID or the file contents themselves. file requires that the file can be read. Only applicable when a round is configured to determine the submission windows relative to the value in a date column in model output files. Not applicable when explicit submission window start and end dates are provided in the hub's config.

derived_task_ids
: Character vector of derived task ID names (task IDs whose values depend on other task IDs). The required values and sample checks leave these columns out. If NULL, defaults to extracting derived task IDs from hub task.json. See get_hub_derived_task_ids() for more details.

Details

Only model output and model metadata files are individually validated using `validate_submission()` or `validate_model_metadata()` respectively although as part of checks, hub config files are also validated. Any other files included in the PR are ignored but flagged in a message.

By default, modifications (which include renaming) and deletions of previously submitted model output files and deletions or renaming of previously submitted model metadata files are not allowed and return a `<error/check_error>` condition class object for each applicable modified/deleted file. This behaviour can be modified through arguments `file_modification_check`, which controls whether modification/deletion checks are performed and what is returned if modifications/deletions are detected, and `allow_submit_window_mods`, which controls whether modifications/deletions of model output files are allowed within their submission windows.

When modification/deletion checks are enabled, each affected file creates an entry in the returned collection named by the file's path. The check within each entry is named `valid_file_status` (reflecting that we validate the file's git status). For example, to access the check for a deleted file: `collection[["team1-goodmodel/2022-10-15-team1-goodmodel.csv"]][["valid_f`

Note that to establish **relative** submission windows when performing modification/deletion checks and `allow_submit_window_mods` is `TRUE`, the reference date is taken as the `round_id` extracted from the file path (i.e. `submit_window_ref_date_from` is always set to `"file_path"`). This is because we cannot extract dates from columns of deleted files. If hub submission window reference dates do not match round IDs in file paths, currently `allow_submit_window_mods` will not work correctly and is best set to `FALSE`. This only relates to hubs/rounds where submission windows are determined relative to a reference date and not when explicit submission window start and end dates are provided in the config.

Finally, note that it is **necessary for `derived_task_ids` to be specified if any task IDs with `required` values have dependent derived task IDs**. If this is the case and derived task IDs are not specified, the dependent nature of derived task ID values will result in **false validation errors when validating required values**.

Checks on model output files:

Details of checks performed by `validate_submission()`

Checks on model metadata files:

Details of checks performed by `validate_model_metadata()`

Value

An object of class `hub_validations_collection`, a collection of validation results. The collection includes entries for hub config validation (`"hub-config"`) and file-specific validations (named by file path).

Examples

```
## Not run:
validate_pr(
  hub_path = ".",
  gh_repo = "hubverse-org/ci-testhub-simple",
  pr_number = 3
```

```
)

## End(Not run)
```

`validate_submission` *Validate a submitted model data file.*

Description

Checks both file level properties like file name, extension, location etc as well as model output data, i.e. the contents of the file.

Usage

```
validate_submission(
  hub_path,
  file_path,
  round_id_col = NULL,
  validations_cfg_path = NULL,
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  skip_submit_window_check = FALSE,
  skip_check_config = FALSE,
  submit_window_ref_date_from = c("file", "file_path"),
  derived_task_ids = NULL
)
```

Arguments

`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`file_path`	character string. Path to the file being validated relative to the hub's model-output directory.
`round_id_col`	Character string. The name of the column containing `round_id`s. Usually, the value of round property `round_id` in hub `tasks.json` config file. Defaults to `NULL` and determined from the config if applicable.
`validations_cfg_path`	Path to `validations.yml` file. If `NULL` defaults to `hub-config/validations.yml`.
`output_type_id_datatype`	character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json`

config file if available. If the property is not set in the config, the argument falls back to "auto" which determines the output_type_id data type automatically from the tasks.json config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where output_type_ids are NA,) are being collected by a hub, the output_type_id column is assigned a character data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce output_type_id to a data type that is not valid for the data (e.g. trying to coerce"character" values to "double") will likely result in an error or potentially unexpected behaviour so use with care.

skip_submit_window_check
: Logical. Whether to skip the submission window check.

skip_check_config
: Logical. Whether to skip the hub config validation check.

submit_window_ref_date_from
: whether to get the reference date around which relative submission windows will be determined from the file's file_path round ID or the file contents themselves. file requires that the file can be read. Only applicable when a round is configured to determine the submission windows relative to the value in a date column in model output files. Not applicable when explicit submission window start and end dates are provided in the hub's config.

derived_task_ids
: Character vector of derived task ID names (task IDs whose values depend on other task IDs). The required values and sample checks leave these columns out. If NULL, defaults to extracting derived task IDs from hub task.json. See get_hub_derived_task_ids() for more details.

Details

Note that it is **necessary for derived_task_ids to be specified if any task IDs with required values have dependent derived task IDs**. If this is the case and derived task IDs are not specified, the dependent nature of derived task ID values will result in **false validation errors when validating required values**.

Details of checks performed by validate_submission()

Value

A hub_validations_collection object containing validation results organized by file. The collection includes separate entries for hub config validation (keyed by "hub-config") and file-specific validations (keyed by file path).

Examples

```
hub_path <- system.file("testhubs/simple", package = "hubValidations")
file_path <- "team1-goodmodel/2022-10-08-team1-goodmodel.csv"
validate_submission(hub_path, file_path)
```

validate_submission_time *Validate a submitted model data file submission time.*

Description

Validate a submitted model data file submission time.

Usage

```
validate_submission_time(
  hub_path,
  file_path,
  ref_date_from = c("file_path", "file")
)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
file_path	character string. Path to the file being validated relative to the hub's model-output directory.
ref_date_from	whether to get the reference date around which relative submission windows will be determined from the file's file_path round ID or the file contents themselves. file requires that the file can be read. Only applicable when a round is configured to determine the submission windows relative to the value in a date column in model output files. Not applicable when explicit submission window start and end dates are provided in the hub's config.

Value

An object of class hub_validations. Each named element contains a hub_check class object reflecting the result of a given check. Function will return early if a check returns an error. The where attribute is set to file_path.

For more details on the structure of <hub_validations> objects, including how to access more information on individual checks, see article on <hub_validations> S3 class objects.

Examples

```
hub_path <- system.file("testhubs/simple", package = "hubValidations")
file_path <- "team1-goodmodel/2022-10-08-team1-goodmodel.csv"
validate_submission_time(hub_path, file_path)
```

`validate_target_data` *Validate the contents of a submitted target data file.*

Description

Validate the contents of a submitted target data file.

Usage

```
validate_target_data(
  hub_path,
  file_path,
  target_type = c("time-series", "oracle-output"),
  date_col = NULL,
  allow_extra_dates = FALSE,
  na = c("NA", ""),
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  validations_cfg_path = NULL,
  round_id = "default"
)
```

Arguments

`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`file_path`	A character string representing the path to the target data file relative to the `target-data` directory.
`target_type`	Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".
`date_col`	Optional name of the column containing the date observations actually occurred (e.g., `"target_end_date"`) to be interpreted as date. Useful when this column does not correspond to a valid task ID (e.g., calculated from other task IDs like `origin_date + horizon`) for: (1) correct schema creation, particularly when it's also a partitioning column, and (2) more robust column name validation when `target-data.json` config does not exist. Ignored when `target-data.json` exists.
`allow_extra_dates`	Logical. If TRUE and target_type is "time-series", allows date values not in tasks.json. Other task ID columns are still strictly validated. Ignored for oracle-output (always strict).

`na` A character vector of strings to interpret as missing values. Only applies to CSV files. The default is `c("NA", "")`. Useful when actual character string `"NA"` values are used in the data. In such a case, use empty cells to indicate missing values in your files and set `na = ""`.

`output_type_id_datatype`
character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json` config file if available. If the property is not set in the config, the argument falls back to `"auto"` which determines the `output_type_id` data type automatically from the `tasks.json` config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where `output_type_id`s are `NA`,) are being collected by a hub, the `output_type_id` column is assigned a `character` data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce `output_type_id` to a data type that is not valid for the data (e.g. trying to coerce`"character"` values to `"double"`) will likely result in an error or potentially unexpected behaviour so use with care.

`validations_cfg_path`
Path to YAML file configuring custom validation checks. If `NULL` defaults to standard `hub-config/validations.yml` path. For more details see article on custom validation checks.

`round_id` Character string. Not generally relevant to target datasets but can be used to specify a specific block of custom validation checks. Otherwise best set to `"default"` which will deploy the default custom validation checks.

Details

Details of checks performed by `validate_target_data()`

Value

An object of class `hub_validations`. Each named element contains a `hub_check` class object reflecting the result of a given check. Function will return early if a check returns an error. The `where` attribute is set to `file_path`.

For more details on the structure of `<hub_validations>` objects, including how to access more information on individual checks, see article on `<hub_validations>` S3 class objects.

Examples

```
hub_path <- system.file("testhubs/v5/target_file", package = "hubUtils")
validate_target_data(hub_path,
  file_path = "time-series.csv",
  target_type = "time-series"
)
validate_target_data(hub_path,
  file_path = "oracle-output.csv",
```

```
  target_type = "oracle-output"
)
hub_path <- system.file("testhubs/v5/target_dir", package = "hubUtils")
validate_target_data(hub_path,
  file_path = "time-series/target=flu_hosp_rate/part-0.parquet",
  target_type = "time-series"
)
validate_target_data(hub_path,
  file_path = "oracle-output/output_type=pmf/part-0.parquet",
  target_type = "oracle-output"
)
```

`validate_target_dataset` *Validate dataset level properties of a given target type*

Description

Validate dataset level properties of a given target type

Usage

```
validate_target_dataset(
  hub_path,
  target_type = c("time-series", "oracle-output"),
  validations_cfg_path = NULL,
  round_id = "default"
)
```

Arguments

`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`target_type`	Type of target data to retrieve matching files. One of "time-series" or "oracle-output". Defaults to "time-series".
`validations_cfg_path`	Path to YAML file configuring custom validation checks. If `NULL` defaults to standard `hub-config/validations.yml` path. For more details see article on custom validation checks.
`round_id`	Character string. Not generally relevant to target datasets but can be used to specify a specific block of custom validation checks. Otherwise best set to `"default"` which will deploy the default custom validation checks.

Details

Details of checks performed by `validate_target_dataset()`

Value

An object of class `hub_validations`. Each named element contains a `hub_check` class object reflecting the result of a given check. Function will return early if a check returns an error. The `where` attribute is set to `target_type` (e.g. `"oracle-output"`, `"time-series"`).

For more details on the structure of `<hub_validations>` objects, including how to access more information on individual checks, see article on `<hub_validations>` S3 class objects.

Examples

```
# Validate single file target datasets
hub_path <- system.file("testhubs/v5/target_file", package = "hubUtils")
validate_target_dataset(hub_path,
  target_type = "time-series"
)
validate_target_dataset(hub_path,
  target_type = "oracle-output"
)
# Validate multi-file partitioned target datasets
hub_path <- system.file("testhubs/v5/target_dir", package = "hubUtils")
validate_target_dataset(hub_path,
  target_type = "time-series"
)
validate_target_dataset(hub_path,
  target_type = "oracle-output"
)
```

---

validate_target_file *Validate file level properties of a target data file.*

Description

Validate file level properties of a target data file.

Usage

```
validate_target_file(
  hub_path,
  file_path,
  validations_cfg_path = NULL,
  round_id = "default"
)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
file_path	A character string representing the path to the target data file relative to the target-data directory.
validations_cfg_path	Path to YAML file configuring custom validation checks. If NULL defaults to standard hub-config/validations.yml path. For more details see article on custom validation checks.
round_id	Character string. Not generally relevant to target datasets but can be used to specify a specific block of custom validation checks. Otherwise best set to "default" which will deploy the default custom validation checks.

Details

Details of checks performed by validate_target_file()

Value

An object of class hub_validations. Each named element contains a hub_check class object reflecting the result of a given check. Function will return early if a check returns an error. The where attribute is set to file_path.

For more details on the structure of <hub_validations> objects, including how to access more information on individual checks, see article on <hub_validations> S3 class objects.

Examples

```
hub_path <- system.file("testhubs/v5/target_file", package = "hubUtils")
validate_target_file(hub_path,
  file_path = "time-series.csv"
)
validate_target_file(hub_path,
  file_path = "oracle-output.csv"
)
hub_path <- system.file("testhubs/v5/target_dir", package = "hubUtils")
validate_target_file(hub_path,
  file_path = "time-series/target=flu_hosp_rate/part-0.parquet"
)
validate_target_file(hub_path,
  file_path = "oracle-output/output_type=pmf/part-0.parquet"
)
```

validate_target_pr *Validate Target Data Pull Request*

Description

Validates target data files in a Pull Request.

Usage

```
validate_target_pr(
  hub_path = ".",
  gh_repo,
  pr_number,
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  date_col = NULL,
  allow_extra_dates = FALSE,
  na = c("NA", ""),
  round_id = "default",
  validations_cfg_path = NULL,
  file_modification_check = c("none", "message", "failure", "error"),
  allow_target_type_deletion = FALSE
)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
gh_repo	GitHub repository address in the format username/repo
pr_number	Number of the pull request to validate
output_type_id_datatype	character string. One of "from_config", "auto", "character", "double", "integer", "logical", "Date". Defaults to "from_config" which uses the setting in the output_type_id_datatype property in the tasks.json config file if available. If the property is not set in the config, the argument falls back to "auto" which determines the output_type_id data type automatically from the tasks.json config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where output_type_ids are NA,) are being collected by a hub, the output_type_id column is assigned a character data type when auto-determined. Other data type

values can be used to override automatic determination. Note that attempting to coerce `output_type_id` to a data type that is not valid for the data (e.g. trying to coerce`"character"` values to `"double"`) will likely result in an error or potentially unexpected behaviour so use with care.

`date_col` Optional name of the column containing the date observations actually occurred (e.g., `"target_end_date"`) to be interpreted as date. Useful when this column does not correspond to a valid task ID (e.g., calculated from other task IDs like `origin_date + horizon`) for: (1) correct schema creation, particularly when it's also a partitioning column, and (2) more robust column name validation when `target-data.json` config does not exist. Ignored when `target-data.json` exists.

`allow_extra_dates`
Logical. If TRUE and target_type is "time-series", allows date values not in tasks.json. Other task ID columns are still strictly validated. Ignored for oracle-output (always strict).

`na` Character vector of strings to interpret as missing values when reading data files. Passed to the underlying file reader.

`round_id` Character string. Not generally relevant to target datasets but can be used to specify a specific block of custom validation checks. Otherwise best set to `"default"` which will deploy the default custom validation checks.

`validations_cfg_path`
Path to YAML file configuring custom validation checks. If `NULL` defaults to standard `hub-config/validations.yml` path. For more details see article on custom validation checks.

`file_modification_check`
Character string. Whether to perform check and what to return when modification/deletion of a previously submitted target data file is detected in PR:

- `"none"`: No modification/deletion checks performed.
- `"message"`: Appends a `<message/check_info>` condition class object for each applicable modified/deleted file.
- `"failure"`: Appends a `<error/check_failure>` condition class object for each applicable modified/deleted file.
- `"error"`: Appends a `<error/check_error>` condition class object for each applicable modified/deleted file.

Defaults to `"none"`.

`allow_target_type_deletion`
Logical. Whether to allow deletion of an entire target type dataset (i.e. all files of a target type) in the PR. Defaults to `FALSE`.

Details

Only target data files are individually validated using `validate_target_submission()` although as part of checks, hub config files and any affected target type datasets as a whole are also validated via `validate_target_dataset()`. Any other files included in the PR are ignored but flagged in a message.

By default, modifications (which include renaming) and deletions of previously submitted target data files are allowed. This behaviour can be modified through arguments `file_modification_check`, which controls whether modification/deletion checks are performed and what is returned if modifications/deletions are detected.

When modification/deletion checks are enabled, each affected file creates an entry in the returned collection named by the file's path. The check within each entry is named `valid_file_status` (reflecting that we validate the file's git status). For example, to access the check for a deleted file: `collection[["oracle-output/output_type=sample/part-0.parquet"]][["valid_`

Checks on target dataset:

Details of checks performed by `validate_target_dataset()`

Checks on individual target files:

Details of checks performed by `validate_target_submission()`

Value

An object of class `target_validations_collection`, a collection of validation results. The collection includes entries for hub config validation (`"hub-config"`), target dataset type validation (`"time-series"`, `"oracle-output"`), and individual file validations (named by file path).

Examples

```
## Not run:
tmp_dir <- withr::local_tempdir()
ci_target_hub_path <- fs::path(tmp_dir, "target")
gert::git_clone(
  url = "https://github.com/hubverse-org/ci-testhub-target.git",
  path = ci_target_hub_path
)
# Validate addition of single file in single file target dataset
gert::git_branch_checkout(
  "add-file-oracle-output",
  repo = ci_target_hub_path
)
validate_target_pr(
  hub_path = ci_target_hub_path,
  gh_repo = "hubverse-org/ci-testhub-target",
  pr_number = 1
)
# Validate addition of multiple files in partitioned target dataset
gert::git_branch_checkout(
  "add-target-dir-files-v5",
  repo = ci_target_hub_path
)
validate_target_pr(
  hub_path = ci_target_hub_path,
  gh_repo = "hubverse-org/ci-testhub-target",
  pr_number = 2
)
```

```
## End(Not run)
```

`validate_target_submission` *Validate a submitted target data file.*

Description

Checks both file level properties like file name, extension, location etc as well as target data, i.e. the contents of the file.

Usage

```
validate_target_submission(
  hub_path,
  file_path,
  target_type = c("time-series", "oracle-output"),
  date_col = NULL,
  allow_extra_dates = FALSE,
  round_id = "default",
  na = c("NA", ""),
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  validations_cfg_path = NULL,
  skip_check_config = FALSE
)
```

Arguments

`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package. The hub must be fully configured with valid `admin.json` and `tasks.json` files within the `hub-config` directory.
`file_path`	A character string representing the path to the target data file relative to the `target-data` directory.
`target_type`	Character string. The type of target data, either `"time-series"` or `"oracle-output"`.
`date_col`	Optional name of the column containing the date observations actually occurred (e.g., `"target_end_date"`) to be interpreted as date. Useful when this column does not correspond to a valid task ID (e.g., calculated from other task IDs like `origin_date + horizon`) for: (1) correct schema creation, particularly when it's also a partitioning column, and (2) more robust column name validation when `target-data.json` config does not exist. Ignored when `target-data.json` exists.

`allow_extra_dates`
: Logical. If TRUE and target_type is "time-series", allows date values not in tasks.json. Other task ID columns are still strictly validated. Ignored for oracle-output (always strict).

`round_id`
: Character string. Not generally relevant to target datasets but can be used to specify a specific block of custom validation checks. Otherwise best set to `"default"` which will deploy the default custom validation checks.

`na`
: Character vector of strings to interpret as missing values when reading data files. Passed to the underlying file reader.

`output_type_id_datatype`
: character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json` config file if available. If the property is not set in the config, the argument falls back to `"auto"` which determines the `output_type_id` data type automatically from the `tasks.json` config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where `output_type_id`s are `NA`,) are being collected by a hub, the `output_type_id` column is assigned a `character` data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce `output_type_id` to a data type that is not valid for the data (e.g. trying to coerce`"character"` values to `"double"`) will likely result in an error or potentially unexpected behaviour so use with care.

`validations_cfg_path`
: Path to YAML file configuring custom validation checks. If `NULL` defaults to standard `hub-config/validations.yml` path. For more details see article on custom validation checks.

`skip_check_config`
: Logical. Whether to skip the hub config validation check.

Details

Details of checks performed by `validate_target_submission()`

Value

A `target_validations_collection` object containing validation results organized by file. The collection includes separate entries for hub config validation (keyed by `"hub-config"`) and file-specific validations (keyed by file path).

Examples

```
hub_path <- system.file("testhubs/v5/target_file", package = "hubUtils")
validate_target_submission(
  hub_path,
  file_path = "time-series.csv",
  target_type = "time-series"
```

```
)
# Example with partitioned data
hub_path <- system.file("testhubs/v5/target_dir", package = "hubUtils")
validate_target_submission(
  hub_path,
  file_path = "time-series/target=flu_hosp_rate/part-0.parquet",
  target_type = "time-series"
)
```

Index

as_hub_validations
(*new_hub_validations*), 68
as_hub_validations_collection
(*new_hub_validations_collection*), 69
as_target_validations
(*new_target_validations*), 70
as_target_validations_collection
(*new_target_validations_collection*), 71

capture_check_cnd, 4
capture_check_info, 5
capture_exec_error, 6
capture_exec_warning, 6
capture_validation_warning, 7
check_config_hub_valid, 8
check_file_exists, 9
check_file_format, 9
check_file_location, 10
check_file_n, 11
check_file_name, 11
check_file_read, 12
check_for_errors, 13
check_for_errors(), *7*
check_metadata_file_exists, 13
check_metadata_file_ext, 14
check_metadata_file_location, 15
check_metadata_file_name, 15
check_metadata_matches_schema, 16
check_metadata_schema_exists, 17
check_submission_metadata_file_exists, 18
check_submission_time, 18
check_target_dataset_exists, 19
check_target_dataset_file_ext_unique, 20
check_target_dataset_rows_unique, 21
check_target_dataset_unique, 22
check_target_file_ext_valid, 23
check_target_file_name, 24
check_target_file_read, 24
check_target_tbl_colnames, 25
check_target_tbl_coltypes, 27
check_target_tbl_oracle_value, 28
check_target_tbl_output_type_ids, 30
check_target_tbl_rows_unique, 31
check_target_tbl_ts_targets, 33
check_target_tbl_values, 34
check_tbl_col_types, 35
check_tbl_colnames, 36
check_tbl_derived_task_id_vals, 37
check_tbl_match_round_id, 38
check_tbl_rows_unique, 39
check_tbl_spl_compound_taskid_set, 40
check_tbl_spl_compound_tid, 42
check_tbl_spl_mt_unique, 43
check_tbl_spl_n, 44
check_tbl_spl_non_compound_tid, 46
check_tbl_unique_round_id, 47
check_tbl_value_col, 48
check_tbl_value_col_ascending, 49
check_tbl_value_col_sum1, 50
check_tbl_values, 51
check_tbl_values_required, 52
check_valid_round_id, 53
check_valid_round_id_col, 54
cli::format_inline(), *4*, *5*, *7*
combine, 55
combine(), *69*, *71*
create_custom_check, 56

dynamic, *4*

expand_model_out_grid, 57

get_config_derived_task_ids, 62
get_config_derived_task_ids(), *59*, *64*, *84*

get_hub_derived_task_ids(), *38*, *41*, *42*, *44*, *45*, *47*, *49*, *50*, *53*, *89*, *93*, *96*
get_target_task_id, 63
get_tbl_compound_taskid_set, 64
gs_bucket(), *8–12*, *14*, *16–20*, *22*, *23*, *25*, *26*, *28–30*, *32–35*, *37*, *39–42*, *44–46*, *48–54*, *73*, *74*, *76*, *77*, *81*, *82*, *88*, *90–92*, *95*, *97*, *98*, *100*, *102*, *103*, *106*

hubData::create_oracle_output_schema(), *82*
hubData::create_timeseries_schema(), *82*
hubUtils::read_config(), *58*, *62*, *66*

is_any_error *(is_success)*, 65
is_error *(is_success)*, 65
is_exec_error *(is_success)*, 65
is_exec_warn *(is_success)*, 65
is_failure *(is_success)*, 65
is_info *(is_success)*, 65
is_success, 65

lubridate::Period, *74*, *77*

match_tbl_to_model_task, 66

new_hub_validations, 68
new_hub_validations(), *56*
new_hub_validations_collection, 69
new_hub_validations_collection(), *56*
new_target_validations, 70
new_target_validations(), *56*
new_target_validations_collection, 71
new_target_validations_collection(), *56*
not_pass *(is_success)*, 65

opt_check_metadata_team_max_model_n, 73
opt_check_tbl_col_timediff, 74
opt_check_tbl_counts_lt_popn, 75
opt_check_tbl_horizon_timediff, 76

parse_file_name, 78
print.hub_validations, 79
print.hub_validations_collection, 80

read_config(), *85*
read_model_out_file, 81
read_target_file, 82
rlang::stack, *5*, *7*

s3_bucket(), *8–12*, *14*, *16–20*, *22*, *23*, *25*, *26*, *28–30*, *32–35*, *37*, *39–42*, *44–46*, *48–54*, *73*, *74*, *76*, *77*, *81*, *82*, *88*, *90–92*, *95*, *97*, *98*, *100*, *102*, *103*, *106*
submission_tmpl, 83

try, *6*
try_check, 87

validate_model_data, 88
validate_model_file, 90
validate_model_metadata, 91
validate_pr, 92
validate_pr(), *13*
validate_submission, 95
validate_submission(), *13*
validate_submission_time, 97
validate_target_data, 98
validate_target_dataset, 100
validate_target_file, 101
validate_target_pr, 103
validate_target_pr(), *13*
validate_target_submission, 106
validate_target_submission(), *13*