Package: hubAdmin (via r-universe)

March 23, 2025

Title Utilities for Administering Hubverse Hubs

Version 1.5.0

Description A set of utility functions for administering 'hubverse' Hubs.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/testthat/edition 3

URL https://github.com/hubverse-org/hubAdmin

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

Imports checkmate, cli, fs, glue, gt (>= 0.11.1), hubUtils (>= 0.3.0), jsonlite, jsonvalidate, magrittr, purrr, rlang, stringr, utils, tibble, lifecycle

Suggests covr, curl, digest, gh, hubData, knitr, withr, rmarkdown, testthat (>= 3.2.0), waldo

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

Config/Needs/website hubverse-org/hubStyle

Depends R (>= 4.1)

LazyData true

VignetteBuilder knitr

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

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

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

RemoteRef v1.5.0

RemoteSha 7d9c2030c4a40b08dde503c367dead6e397f0a6e

Contents

`append_round` *Append one or more rounds to a config class object.*

Description

Append one or more <round>s sequentially to the end of the `rounds` property of a <config> class object.

Usage

```
append_round(config, ...)
```

Arguments

`config`	an object of class <config>.
`...`	one or more objects of class <round>.

Value

an object of class <config> with the rounds appended sequentially to the end of the `rounds` property.

Examples

```
config <- create_config(
  create_rounds(
    create_round(
      round_id_from_variable = TRUE,
      round_id = "origin_date",
      model_tasks = create_model_tasks(
        create_model_task(
          task_ids = create_task_ids(
            create_task_id("origin_date",
              required = NULL,
              optional = c(
                "2023-01-02",
                "2023-01-09",
                "2023-01-16"
              )
            ),
            create_task_id("location",
              required = "US",
              optional = c("01", "02", "04", "05", "06")
            ),
            create_task_id("horizon",
              required = 1L,
              optional = 2:4
            )
          ),
          output_type = create_output_type(
            create_output_type_mean(
              is_required = TRUE,
              value_type = "double",
              value_minimum = 0L
            )
          ),
          target_metadata = create_target_metadata(
            create_target_metadata_item(
              target_id = "inc hosp",
              target_name = "Weekly incident influenza hospitalizations",
              target_units = "rate per 100,000 population",
              target_keys = NULL,
              target_type = "discrete",
              is_step_ahead = TRUE,
              time_unit = "week"
            )
          )
        )
      ),
      submissions_due = list(
        relative_to = "origin_date",
        start = -4L,
        end = 2L
      )
    )
```

```
  )
)
# Add a new round with an age_group task_id
new_round <- create_round(
  round_id_from_variable = TRUE,
  round_id = "origin_date",
  model_tasks = create_model_tasks(
    create_model_task(
      task_ids = create_task_ids(
        create_task_id("origin_date",
          required = NULL,
          optional = c(
            "2023-01-23"
          )
        ),
        create_task_id("location",
          required = "US",
          optional = c("01", "02", "04", "05", "06")
        ),
        create_task_id("horizon",
          required = 1L,
          optional = 2:4
        ),
        create_task_id("age_group",
          required = NULL,
          optional = c("1", "2", "3", "4", "5")
        )
      ),
      output_type = create_output_type(
        create_output_type_mean(
          is_required = TRUE,
          value_type = "double",
          value_minimum = 0L
        )
      ),
      target_metadata = create_target_metadata(
        create_target_metadata_item(
          target_id = "inc hosp",
          target_name = "Weekly incident influenza hospitalizations",
          target_units = "rate per 100,000 population",
          target_keys = NULL,
          target_type = "discrete",
          is_step_ahead = TRUE,
          time_unit = "week"
        )
      )
    )
  ),
  submissions_due = list(
    relative_to = "origin_date",
    start = -4L,
    end = 2L
  )
```

```
)
append_round(config, new_round)
# Append in existing config file using an older schema version
options(hubAdmin.schema_version = "v4.0.0")
config <- hubUtils::read_config_file(
  system.file("v4-tasks.json", package = "hubAdmin")
)
# Create new round using version defined through
# hubAdmin.schema_version option
new_round <- create_round(
  round_id_from_variable = TRUE,
  round_id = "origin_date",
  model_tasks = create_model_tasks(
    create_model_task(
      task_ids = create_task_ids(
        create_task_id("origin_date",
          required = NULL,
          optional = c(
            "2023-01-23"
          )
        ),
        create_task_id("location",
          required = "US",
          optional = c("01", "02", "04", "05", "06")
        ),
        create_task_id("horizon",
          required = 1L,
          optional = 2:4
        ),
        create_task_id("age_group",
          required = NULL,
          optional = c("1", "2", "3", "4", "5")
        )
      ),
      output_type = create_output_type(
        create_output_type_mean(
          is_required = TRUE,
          value_type = "double",
          value_minimum = 0L
        )
      ),
      target_metadata = create_target_metadata(
        create_target_metadata_item(
          target_id = "inc hosp",
          target_name = "Weekly incident influenza hospitalizations",
          target_units = "rate per 100,000 population",
          target_keys = NULL,
          target_type = "discrete",
          is_step_ahead = TRUE,
          time_unit = "week"
        )
      )
    )
```

```
  ),
  submissions_due = list(
    relative_to = "origin_date",
    start = -4L,
    end = 2L
  )
)
append_round(config, new_round)
# Reset option to latest schema version
options(hubAdmin.schema_version = "latest")
```

`create_config` *Create a* `config` *class object.*

Description

Create a representation of a complete `"tasks"` config file as a list object of class `config`. This can be written out to a `tasks.json` file.

Usage

```
create_config(rounds, output_type_id_datatype = NULL, derived_task_ids = NULL)
```

Arguments

`rounds`	An object of class `rounds` created using function `create_rounds()`
`output_type_id_datatype`	A character string specifying the value of the `output_type_id_datatype` property. Only available since hubverse schema v3.0.1. Ignored if NULL (default) and throws a warning if a value is provided and the schema version used for the config is <= v3.0.1. This property is used to set the data type of the `output_type_id` column in model outputs and can take values of `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, or `"Date"`. For more details consult the hubDocs documentation on model output datatypes
`derived_task_ids`	character vector of derived task id names (i.e. task IDs whose values are depended on the values of other task IDs). Only available for schema version v4.0.0 and later.

Details

For more details consult the documentation on `tasks.json` Hub config files.

Value

a named list of class `config`.

See Also

create_rounds()

Examples

```
rounds <- create_rounds(
  create_round(
    round_id_from_variable = TRUE,
    round_id = "origin_date",
    model_tasks = create_model_tasks(
      create_model_task(
        task_ids = create_task_ids(
          create_task_id("origin_date",
            required = NULL,
            optional = c(
              "2023-01-02",
              "2023-01-09",
              "2023-01-16"
            )
          ),
          create_task_id("location",
            required = "US",
            optional = c("01", "02", "04", "05", "06")
          ),
          create_task_id("horizon",
            required = 1L,
            optional = 2:4
          )
        ),
        output_type = create_output_type(
          create_output_type_mean(
            is_required = TRUE,
            value_type = "double",
            value_minimum = 0L
          )
        ),
        target_metadata = create_target_metadata(
          create_target_metadata_item(
            target_id = "inc hosp",
            target_name = "Weekly incident influenza hospitalizations",
            target_units = "rate per 100,000 population",
            target_keys = NULL,
            target_type = "discrete",
            is_step_ahead = TRUE,
            time_unit = "week"
          )
        )
      )
    ),
    submissions_due = list(
      relative_to = "origin_date",
      start = -4L,
```

```
      end = 2L
    )
  )
)
create_config(rounds)
```

`create_model_task` *Create an object of class* `model_task`

Description

Create an object of class `model_task` representing a model task. Multiple model tasks can be combined using function `create_model_tasks()`.

Usage

```
create_model_task(task_ids, output_type, target_metadata)
```

Arguments

`task_ids`	object of class `model_task`.
`output_type`	object of class `output_type`.
`target_metadata`	object of class `target_metadata`.

Value

a named list of class `model_task`.

See Also

`create_task_ids()`, `create_output_type()`, `create_target_metadata()`, `create_model_tasks()`

Examples

```
create_model_task(
  task_ids = create_task_ids(
    create_task_id("origin_date",
      required = NULL,
      optional = c(
        "2023-01-02",
        "2023-01-09",
        "2023-01-16"
      )
    ),
    create_task_id("location",
      required = "US",
      optional = c("01", "02", "04", "05", "06")
    ),
```

```
    create_task_id("horizon",
      required = 1L,
      optional = 2:4
    )
  ),
  output_type = create_output_type(
    create_output_type_mean(
      is_required = TRUE,
      value_type = "double",
      value_minimum = 0L
    )
  ),
  target_metadata = create_target_metadata(
    create_target_metadata_item(
      target_id = "inc hosp",
      target_name = "Weekly incident influenza hospitalizations",
      target_units = "rate per 100,000 population",
      target_keys = NULL,
      target_type = "discrete",
      is_step_ahead = TRUE,
      time_unit = "week"
    )
  )
)
```

`create_model_tasks` *Create a* `model_tasks` *class object.*

Description

Create a `model_tasks` class object.

Usage

```
create_model_tasks(...)
```

Arguments

`...`	objects of class `model_tasks` created using function `create_model_task()`

Value

a named list of class `model_tasks`.

See Also

`create_model_task()`

Examples

```
create_model_tasks(
  create_model_task(
    task_ids = create_task_ids(
      create_task_id("origin_date",
        required = NULL,
        optional = c(
          "2023-01-02",
          "2023-01-09",
          "2023-01-16"
        )
      ),
      create_task_id("location",
        required = "US",
        optional = c("01", "02", "04", "05", "06")
      ),
      create_task_id("horizon",
        required = 1L,
        optional = 2:4
      )
    ),
    output_type = create_output_type(
      create_output_type_mean(
        is_required = TRUE,
        value_type = "double",
        value_minimum = 0L
      )
    ),
    target_metadata = create_target_metadata(
      create_target_metadata_item(
        target_id = "inc hosp",
        target_name = "Weekly incident influenza hospitalizations",
        target_units = "rate per 100,000 population",
        target_keys = NULL,
        target_type = "discrete",
        is_step_ahead = TRUE,
        time_unit = "week"
      )
    )
  )
)
create_model_tasks(
  create_model_task(
    task_ids = create_task_ids(
      create_task_id("origin_date",
        required = NULL,
        optional = c(
          "2023-01-02",
          "2023-01-09",
          "2023-01-16"
        )
      ),
```

```
  create_task_id("location",
    required = "US",
    optional = c("01", "02", "04", "05", "06")
  ),
  create_task_id("target",
    required = NULL,
    optional = c("inc death", "inc hosp")
  ),
  create_task_id("horizon",
    required = 1L,
    optional = 2:4
  )
),
output_type = create_output_type(
  create_output_type_mean(
    is_required = TRUE,
    value_type = "double",
    value_minimum = 0L
  ),
  create_output_type_median(
    is_required = FALSE,
    value_type = "double"
  ),
  create_output_type_quantile(
    required = c(
      0.1, 0.2, 0.3, 0.4, 0.6,
      0.7, 0.8, 0.9
    ),
    is_required = TRUE,
    value_type = "double",
    value_minimum = 0
  )
),
target_metadata = create_target_metadata(
  create_target_metadata_item(
    target_id = "inc hosp",
    target_name = "Weekly incident influenza hospitalizations",
    target_units = "rate per 100,000 population",
    target_keys = list(target = "inc hosp"),
    target_type = "discrete",
    is_step_ahead = TRUE,
    time_unit = "week"
  ),
  create_target_metadata_item(
    target_id = "inc death",
    target_name = "Weekly incident influenza deaths",
    target_units = "rate per 100,000 population",
    target_keys = list(target = "inc death"),
    target_type = "discrete",
    is_step_ahead = TRUE,
    time_unit = "week"
  )
)
```

```
  ),
  create_model_task(
    task_ids = create_task_ids(
      create_task_id("origin_date",
        required = NULL,
        optional = c(
          "2023-01-02",
          "2023-01-09",
          "2023-01-16"
        )
      ),
      create_task_id("location",
        required = "US",
        optional = c("01", "02", "04", "05", "06")
      ),
      create_task_id("target",
        required = "flu hosp rt chng",
        optional = NULL
      ),
      create_task_id("horizon",
        required = 1L,
        optional = 2:4
      )
    ),
    output_type = create_output_type(
      create_output_type_pmf(
        required = c(
          "large_decrease",
          "decrease",
          "stable",
          "increase",
          "large_increase"
        ),
        is_required = TRUE,
        value_type = "double"
      )
    ),
    target_metadata = create_target_metadata(
      create_target_metadata_item(
        target_id = "flu hosp rt chng",
        target_name = "Weekly influenza hospitalization rate change",
        target_units = "rate per 100,000 population",
        target_keys = list(target = "flu hosp rt chng"),
        target_type = "nominal",
        is_step_ahead = TRUE,
        time_unit = "week"
      )
    )
  )
)
```

`create_output_type` *Create an* `output_type` *class object.*

Description

Create an `output_type` class object.

Usage

```
create_output_type(...)
```

Arguments

`...`	objects of class `output_type_item` created using functions from the `create_output_type_*()` family of functions.

Details

For more details consult the documentation on `tasks.json` Hub config files.

Value

a named list of class `output_type`.

See Also

`create_output_type_mean()`, `create_output_type_median()`, `create_output_type_quantile()`, `create_output_type_cdf()`, `create_output_type_pmf()`, `create_output_type_sample()`

Examples

```
create_output_type(
  create_output_type_mean(
    is_required = TRUE,
    value_type = "double",
    value_minimum = 0L
  ),
  create_output_type_median(
    is_required = FALSE,
    value_type = "double"
  ),
  create_output_type_quantile(
    required = c(
      0.1, 0.2, 0.3, 0.4, 0.6,
      0.7, 0.8, 0.9
    ),
    is_required = TRUE,
    value_type = "double",
    value_minimum = 0
```

```
  )
)
```

`create_output_type_mean` *Create a point estimate output type object of class* `output_type_item`

Description

Create a representation of a `mean` or `median` output type as a list object of class `output_type_item`. This can be combined with additional `output_type_item` objects using function `create_output_type()` to create an `output_type` object for a given model_task. This can be combined with other building blocks which can then be written as or appended to `tasks.json` Hub config files.

Usage

```
create_output_type_mean(
  is_required,
  value_type,
  value_minimum = NULL,
  value_maximum = NULL,
  schema_version = getOption("hubAdmin.schema_version", default = "latest"),
  branch = getOption("hubAdmin.branch", default = "main")
)

create_output_type_median(
  is_required,
  value_type,
  value_minimum = NULL,
  value_maximum = NULL,
  schema_version = getOption("hubAdmin.schema_version", default = "latest"),
  branch = getOption("hubAdmin.branch", default = "main")
)
```

Arguments

`is_required`	Logical. Is the output type required?
`value_type`	Character string. The data type of the output_type values.
`value_minimum`	Numeric. The inclusive minimum of output_type values.
`value_maximum`	Numeric. The inclusive maximum of output_type values.
`schema_version`	Character string specifying the json schema version to be used for validation. The default value `"latest"` will use the latest version available in the hubverse schemas repository. Alternatively, a specific version of a schema (e.g. `"v0.0.1"`) can be specified. Can be set through global option "hubAdmin.schema_version".
`branch`	The branch of the hubverse schemas repository from which to fetch schema. Defaults to `"main"`. Can be set through global option "hubAdmin.branch".

Details

For more details consult the documentation on `tasks.json` Hub config files.

Value

a named list of class `output_type_item` representing a `mean` or `median` output type.

Functions

- `create_output_type_mean()`: Create a list representation of a `mean` output type.
- `create_output_type_median()`: Create a list representation of a `median` output type.

See Also

`create_output_type()`

Examples

```
create_output_type_mean(
  is_required = TRUE,
  value_type = "double",
  value_minimum = 0L
)
create_output_type_median(
  is_required = FALSE,
  value_type = "integer"
)
# Pre v4.0.0 schema output
create_output_type_mean(
  is_required = TRUE,
  value_type = "double",
  value_minimum = 0L,
  schema_version = "v3.0.1"
)
# Set schema version for all subsequent calls
options(hubAdmin.schema_version = "v3.0.1")
create_output_type_mean(
  is_required = TRUE,
  value_type = "double",
  value_minimum = 0L,
  schema_version = "v3.0.1"
)
options(hubAdmin.schema_version = "latest")
```

create_output_type_quantile *Create a distribution output type object of class* output_type_item

Description

Create a representation of a quantile, cdf, pmf or sample output type as a list object of class output_type_item. This can be combined with additional output_type_items using function create_output_type() to create an output_type object for a given model_task. This can be combined with other building blocks which can then be written as or appended to tasks.json Hub config files.

Usage

```
create_output_type_quantile(
  required,
  optional,
  is_required,
  value_type,
  value_minimum = NULL,
  value_maximum = NULL,
  schema_version = getOption("hubAdmin.schema_version", default = "latest"),
  branch = getOption("hubAdmin.branch", default = "main")
)

create_output_type_cdf(
  required,
  optional,
  is_required,
  value_type,
  schema_version = getOption("hubAdmin.schema_version", default = "latest"),
  branch = getOption("hubAdmin.branch", default = "main")
)

create_output_type_pmf(
  required,
  optional,
  is_required,
  value_type,
  schema_version = getOption("hubAdmin.schema_version", default = "latest"),
  branch = getOption("hubAdmin.branch", default = "main")
)

create_output_type_sample(
  is_required,
  output_type_id_type,
  max_length = NULL,
```

```
  min_samples_per_task,
  max_samples_per_task,
  compound_taskid_set = NULL,
  value_type,
  value_minimum = NULL,
  value_maximum = NULL,
  schema_version = getOption("hubAdmin.schema_version", default = "latest"),
  branch = getOption("hubAdmin.branch", default = "main")
)
```

Arguments

`required`	Atomic vector of required `output_type_id` values. Can be NULL if all values are optional.
`optional`	**[Deprecated] (schema >= v4.0.0)**. Atomic vector of optional `output_type_id` values. Can be NULL if all values are required.
`is_required`	Logical. Is this sample output type required?
`value_type`	Character string. The data type of the output_type values.
`value_minimum`	Numeric. The inclusive minimum of output_type values.
`value_maximum`	Numeric. The inclusive maximum of output_type values.
`schema_version`	Character string specifying the json schema version to be used for validation. The default value `"latest"` will use the latest version available in the hubverse schemas repository. Alternatively, a specific version of a schema (e.g. `"v0.0.1"`) can be specified. Can be set through global option "hubAdmin.schema_version".
`branch`	The branch of the hubverse schemas repository from which to fetch schema. Defaults to `"main"`. Can be set through global option "hubAdmin.branch".
`output_type_id_type`	Character string. The data type of the output_type_id. One of "integer" or "character".
`max_length`	Integer. Optional. The maximum length of the output_type_id value if type is `"character"`.
`min_samples_per_task`	Integer. The minimum number of samples per task. Must be equal to or less than `max_samples_per_task`.
`max_samples_per_task`	Integer. The maximum number of samples per task. Must be equal to or greater than `min_samples_per_task`.
`compound_taskid_set`	Character vector. Optional. The set of compound task IDs that the sample each output type can be associated with.

Details

For more details consult the documentation on `tasks.json` Hub config files.

Value

a named list of class `output_type_item` representing a `quantile`, `cdf`, `pmf` or `sample` output type.

Functions

- `create_output_type_quantile()`: Create a list representation of a `quantile` output type.
- `create_output_type_cdf()`: Create a list representation of a `cdf` output type.
- `create_output_type_pmf()`: Create a list representation of a `pmf` output type.
- `create_output_type_sample()`: Create a list representation of a `sample` output type.

See Also

`create_output_type()`

Examples

```
create_output_type_quantile(
  required = c(0.25, 0.5, 0.75),
  is_required = TRUE,
  value_type = "double",
  value_minimum = 0
)
create_output_type_cdf(
  required = c(10, 20),
  is_required = FALSE,
  value_type = "double"
)
create_output_type_cdf(
  required = c("EW202240", "EW202241", "EW202242"),
  is_required = TRUE,
  value_type = "double"
)
create_output_type_pmf(
  required = c("low", "moderate", "high", "extreme"),
  is_required = FALSE,
  value_type = "double"
)
create_output_type_sample(
  is_required = TRUE,
  output_type_id_type = "integer",
  min_samples_per_task = 70L, max_samples_per_task = 100L,
  value_type = "double",
  value_minimum = 0,
  value_maximum = 1
)
# Pre v4.0.0 schema output
create_output_type_quantile(
  required = c(0.25, 0.5, 0.75),
  optional = c(
    0.1, 0.2, 0.3, 0.4, 0.6,
    0.7, 0.8, 0.9
```

```
  ),
  value_type = "double",
  value_minimum = 0,
  schema_version = "v3.0.1"
)
# Set schema version for all subsequent calls
options(hubAdmin.schema_version = "v3.0.1")
create_output_type_quantile(
  required = c(0.25, 0.5, 0.75),
  optional = c(
    0.1, 0.2, 0.3, 0.4, 0.6,
    0.7, 0.8, 0.9
  ),
  value_type = "double",
  value_minimum = 0
)
create_output_type_cdf(
  required = c(10, 20),
  optional = NULL,
  value_type = "double"
)
create_output_type_cdf(
  required = NULL,
  optional = c("EW202240", "EW202241", "EW202242"),
  value_type = "double"
)
create_output_type_pmf(
  required = NULL,
  optional = c("low", "moderate", "high", "extreme"),
  value_type = "double"
)
options(hubAdmin.schema_version = "latest")
```

`create_round` *Create an object of class* `round`

Description

Create a representation of a round item as a list object of class `round`. This can be combined with additional `round` objects using function `create_rounds()`. Such building blocks can ultimately be combined and then written out as or appended to `tasks.json` Hub config files.

Usage

```
create_round(
  round_id_from_variable,
  round_id,
  round_name = NULL,
  model_tasks,
  submissions_due,
```

```
  last_data_date = NULL,
  file_format = NULL,
  derived_task_ids = NULL
)
```

Arguments

`round_id_from_variable`
: logical. Whether `round_id` is inferred from the values of a `task_id` variable within the `model_tasks` `model_task` items.

`round_id`
: character string. The round identifier. If `round_id_from_variable = TRUE`, `round_id` should be the name of a `task_id` variable present in all `model_tasks` `model_task` items.

`round_name`
: character string. An optional round name. This can be useful for internal referencing of rounds, for examples, when a date is used as `round_id` but hub maintainers and teams also refer to rounds as round-1, round-2 etc.

`model_tasks`
: an object of class `model_tasks` created with function `create_model_tasks()`.

`submissions_due`
: named list conforming to one of the two following structures:

  1. Submission dates for round is determined relative to an origin date.
     - `relative_to`: character string of the name of the `task_id` variable containing origin dates in relation to which submission start and end dates are determined.
     - `start`: integer. Difference in days between start and origin date.
     - `end`: integer. Difference in days between end and origin date.
  2. Submission dates for round are provided explicitly.
     - `start`: character. Submission start date in ISO 8601 format (YYYY-MM-DD).
     - `end`: character. Submission end date in ISO 8601 format (YYYY-MM-DD).

`last_data_date`
: character date in ISO 8601 format (YYYY-MM-DD). The last date with recorded data in the data set used as input to a model. Optional.

`file_format`
: character string. An optional specification of a file format for the round. This option in only available for some versions of the schema and is ignored if not allowed in the version of the schema used. It also overrides any specification of file format in `admin.json`. For more details on whether this argument can be used as well as available formats, please consult the documentation on `tasks.json` Hub config files.

`derived_task_ids`
: character vector of derived task id names (i.e. task IDs whose values are depended on the values of other task IDs). Only available for schema version v4.0.0 and later.

Details

For more details consult the documentation on `tasks.json` Hub config files.

Value

a named list of class round.

See Also

create_rounds()

Examples

```
model_tasks <- create_model_tasks(
  create_model_task(
    task_ids = create_task_ids(
      create_task_id("origin_date",
        required = NULL,
        optional = c(
          "2023-01-02",
          "2023-01-09",
          "2023-01-16"
        )
      ),
      create_task_id("location",
        required = "US",
        optional = c("01", "02", "04", "05", "06")
      ),
      create_task_id("horizon",
        required = 1L,
        optional = 2:4
      )
    ),
    output_type = create_output_type(
      create_output_type_mean(
        is_required = TRUE,
        value_type = "double",
        value_minimum = 0L
      )
    ),
    target_metadata = create_target_metadata(
      create_target_metadata_item(
        target_id = "inc hosp",
        target_name = "Weekly incident influenza hospitalizations",
        target_units = "rate per 100,000 population",
        target_keys = NULL,
        target_type = "discrete",
        is_step_ahead = TRUE,
        time_unit = "week"
      )
    )
  )
)
create_round(
  round_id_from_variable = TRUE,
  round_id = "origin_date",
```

```
  model_tasks = model_tasks,
  submissions_due = list(
    relative_to = "origin_date",
    start = -4L,
    end = 2L
  ),
  last_data_date = "2023-01-02"
)
```

`create_rounds` *Create a* `rounds` *class object.*

Description

Create a `rounds` class object.

Usage

```
create_rounds(...)
```

Arguments

`...`	objects of class `round` created using function `create_round()`

Details

For more details consult the documentation on `tasks.json` Hub config files.

Value

a named list of class `rounds`.

See Also

`create_round()`

Examples

```
model_tasks <- create_model_tasks(
  create_model_task(
    task_ids = create_task_ids(
      create_task_id("origin_date",
        required = NULL,
        optional = c(
          "2023-01-02",
          "2023-01-09",
          "2023-01-16"
        )
      ),
      create_task_id("location",
```

```
          required = "US",
          optional = c("01", "02", "04", "05", "06")
        ),
        create_task_id("horizon",
          required = 1L,
          optional = 2:4
        )
      ),
      output_type = create_output_type(
        create_output_type_mean(
          is_required = TRUE,
          value_type = "double",
          value_minimum = 0L
        )
      ),
      target_metadata = create_target_metadata(
        create_target_metadata_item(
          target_id = "inc hosp",
          target_name = "Weekly incident influenza hospitalizations",
          target_units = "rate per 100,000 population",
          target_keys = NULL,
          target_type = "discrete",
          is_step_ahead = TRUE,
          time_unit = "week"
        )
      )
    )
  )
  # Create a rounds object with a single rounds where round_id is defined through
  # the value of a task_id variable.
  create_rounds(
    create_round(
      round_id_from_variable = TRUE,
      round_id = "origin_date",
      model_tasks = model_tasks,
      submissions_due = list(
        relative_to = "origin_date",
        start = -4L,
        end = 2L
      )
    )
  )
  # Create a rounds object with two rounds and user defined round_ids
  create_rounds(
    create_round(
      round_id_from_variable = FALSE,
      round_id = "round_1",
      model_tasks =
        create_model_tasks(
          create_model_task(
            task_ids = create_task_ids(
              create_task_id("origin_date",
                required = NULL,
```

```
            optional = c(
              "2023-01-09"
            )
          ),
          create_task_id("location",
            required = "US",
            optional = c("01", "02", "04", "05", "06")
          ),
          create_task_id("horizon",
            required = 1L,
            optional = 2:4
          )
        ),
        output_type = create_output_type(
          create_output_type_mean(
            is_required = TRUE,
            value_type = "double",
            value_minimum = 0L
          )
        ),
        target_metadata = create_target_metadata(
          create_target_metadata_item(
            target_id = "inc hosp",
            target_name = "Weekly incident influenza hospitalizations",
            target_units = "rate per 100,000 population",
            target_keys = NULL,
            target_type = "discrete",
            is_step_ahead = TRUE,
            time_unit = "week"
          )
        )
      )
    ),
  submissions_due = list(
    start = "2023-01-05",
    end = "2023-01-11"
  ),
  last_data_date = "2023-01-06"
),
create_round(
  round_id_from_variable = FALSE,
  round_id = "round_2",
  model_tasks =
    create_model_tasks(
      create_model_task(
        task_ids = create_task_ids(
          create_task_id("origin_date",
            required = NULL,
            optional = c(
              "2023-01-16"
            )
          ),
          create_task_id("location",
```

```
            required = "US",
            optional = c("01", "02", "04", "05", "06")
          ),
          create_task_id("horizon",
            required = 1L,
            optional = 2:4
          )
        ),
        output_type = create_output_type(
          create_output_type_mean(
            is_required = TRUE,
            value_type = "double",
            value_minimum = 0L
          )
        ),
        target_metadata = create_target_metadata(
          create_target_metadata_item(
            target_id = "inc hosp",
            target_name = "Weekly incident influenza hospitalizations",
            target_units = "rate per 100,000 population",
            target_keys = NULL,
            target_type = "discrete",
            is_step_ahead = TRUE,
            time_unit = "week"
          )
        )
      )
    ),
    submissions_due = list(
      start = "2023-01-12",
      end = "2023-01-18"
    ),
    last_data_date = "2023-01-13"
  )
)
```

`create_target_metadata` *Create a* `target_metadata` *class object.*

Description

Create a `target_metadata` class object.

Usage

```
create_target_metadata(...)
```

Arguments

`...`	objects of class `target_metadata_item` created using function `create_target_metadata_item()`

Details

For more details consult the documentation on `tasks.json` Hub config files.

Value

a named list of class `target_metadata`.

See Also

`create_target_metadata_item()`

Examples

```
create_target_metadata(
  create_target_metadata_item(
    target_id = "inc hosp",
    target_name = "Weekly incident influenza hospitalizations",
    target_units = "rate per 100,000 population",
    target_keys = list(target = "inc hosp"),
    target_type = "discrete",
    is_step_ahead = TRUE,
    time_unit = "week"
  ),
  create_target_metadata_item(
    target_id = "inc death",
    target_name = "Weekly incident influenza deaths",
    target_units = "rate per 100,000 population",
    target_keys = list(target = "inc death"),
    target_type = "discrete",
    is_step_ahead = TRUE,
    time_unit = "week"
  )
)
```

`create_target_metadata_item` *Create an object of class* `target_metadata_item`

Description

Create a representation of a target_metadata item as a list object of class `target_metadata_item`. This can be combined with additional target_metadata items using function `create_target_metadata()` to create a target_metadata object for a given model_task. Such building blocks can ultimately be combined and then written out as or appended to `tasks.json` Hub config files.

Usage

```
create_target_metadata_item(
  target_id,
  target_name,
  target_units,
  target_keys = NULL,
  description = NULL,
  target_type,
  is_step_ahead,
  time_unit = NULL,
  schema_version = getOption("hubAdmin.schema_version", default = "latest"),
  branch = getOption("hubAdmin.branch", default = "main")
)
```

Arguments

target_id	character string. Short description that uniquely identifies the target.
target_name	character string. A longer human readable target description that could be used, for example, as a visualisation axis label.
target_units	character string. Unit of observation of the target.
target_keys	named list or NULL. The target_keys value defines a single target. Should be a named list containing a single character string element. The name of the element should match a task_id variable name within the same model_tasks object and the value should match a single value of that variable as described in target metadata section of the official hubverse documentation. # nolint: line_length_linter Otherwise, NULL in the case where the target is not specified as a task_id but is specified solely through the target_id argument.
description	character string (optional). An optional verbose description of the target that might include information such as definitions of a 'rate' or similar.
target_type	character string. Target statistical data type. Consult the appropriate version of the hub schema for potential values.
is_step_ahead	logical. Whether the target is part of a sequence of values
time_unit	character string. If is_step_ahead is TRUE, then this argument is required and defines the unit of time steps. if is_step_ahead is FALSE, then this argument is not required and will be ignored if given.
schema_version	Character string specifying the json schema version to be used for validation. The default value "latest" will use the latest version available in the hubverse schemas repository. Alternatively, a specific version of a schema (e.g. "v0.0.1") can be specified. Can be set through global option "hubAdmin.schema_version".
branch	The branch of the hubverse schemas repository from which to fetch schema. Defaults to "main". Can be set through global option "hubAdmin.branch".

Details

For more details consult the documentation on tasks.json Hub config files.

Value

a named list of class `target_metadata_item`.

See Also

`create_target_metadata()`

Examples

```
create_target_metadata_item(
  target_id = "inc hosp",
  target_name = "Weekly incident influenza hospitalizations",
  target_units = "rate per 100,000 population",
  target_keys = list(target = "inc hosp"),
  target_type = "discrete",
  is_step_ahead = TRUE,
  time_unit = "week"
)
options(hubAdmin.schema_version = "v3.0.1")
create_target_metadata_item(
  target_id = "inc hosp",
  target_name = "Weekly incident influenza hospitalizations",
  target_units = "rate per 100,000 population",
  target_keys = list(target = "inc hosp"),
  target_type = "discrete",
  is_step_ahead = TRUE,
  time_unit = "week"
)
options(hubAdmin.schema_version = "latest")
```

---

create_task_id *Create an object of class* task_id

Description

Create a representation of a task ID item as a list object of class `task_id`. This can be combined with additional `task_id` objects using function `create_task_ids()` to create a `task_ids` class object for a given model_task. Such building blocks can ultimately be combined and then written out as or appended to `tasks.json` Hub config files.

Usage

```
create_task_id(
  name,
  required,
  optional,
  schema_version = getOption("hubAdmin.schema_version", default = "latest"),
  branch = getOption("hubAdmin.branch", default = "main")
)
```

Arguments

name	character string, Name of task_id to create.
required	Atomic vector of required task_id values. Can be NULL if all values are optional.
optional	Atomic vector of optional task_id values. Can be NULL if all values are required.
schema_version	Character string specifying the json schema version to be used for validation. The default value "latest" will use the latest version available in the hubverse schemas repository. Alternatively, a specific version of a schema (e.g. "v0.0.1") can be specified. Can be set through global option "hubAdmin.schema_version".
branch	The branch of the hubverse schemas repository from which to fetch schema. Defaults to "main". Can be set through global option "hubAdmin.branch".

Details

required and optional vectors for standard task_ids defined in a Hub schema must match data types and formats specified in the schema. For more details consult the documentation on tasks.json Hub config files

JSON schema data type names differ to those in R. Use the following mappings to create vectors of appropriate data types which will correspond to correct JSON schema data types during config file validation.

json	R
string	character
boolean	logical
integer	integer
number	double

Values across required and optional arguments must be unique. required and optional must be of the same type (unless NULL). Task_ids that represent dates must be supplied as character strings in ISO 8601 date format (YYYY-MM-DD). If working with date objects, please convert to character (e.g. using as.character()) before supplying as arguments.

Task_ids not present in the schema are allowed as additional properties but the user is responsible for providing values of the correct data type.

Value

a named list of class task_id representing a task ID.

See Also

create_task_ids()

Examples

```
create_task_id("horizon", required = 1L, optional = 2:4)
# Set schema version to "v3.0.1" for all subsequent calls
options(hubAdmin.schema_version = "v3.0.1")
```

```
create_task_id("horizon", required = 1L, optional = 2:4)
create_task_id("location", required = "US", optional = c("01", "02"))
options(hubAdmin.schema_version = "latest")
```

`create_task_ids` *Create a* `task_ids` *class object.*

Description

Create a `task_ids` class object.

Usage

```
create_task_ids(...)
```

Arguments

`...`	objects of class `task_id` created using function `create_task_id()`

Details

For more details consult the documentation on `tasks.json` Hub config files.

Value

a named list of class `task_ids`.

See Also

`create_task_id()`

Examples

```
create_task_ids(
  create_task_id("origin_date",
    required = NULL,
    optional = c(
      "2023-01-02",
      "2023-01-09",
      "2023-01-16"
    )
  ),
  create_task_id("scenario_id",
    required = NULL,
    optional = c(
      "A-2021-03-28",
      "B-2021-03-28"
    )
  ),
  create_task_id("location",
```

```
    required = "US",
    optional = c("01", "02", "04", "05", "06")
  ),
  create_task_id("target",
    required = "inc hosp",
    optional = NULL
  ),
  create_task_id("horizon",
    required = 1L,
    optional = 2:4
  )
)
```

`download_tasks_schema` *Download hubverse tasks schema from the hubverse schema repository.*

Description

Download hubverse tasks schema from the hubverse schema repository.

Usage

```
download_tasks_schema(
  schema_version = getOption("hubAdmin.schema_version", default = "latest"),
  branch = getOption("hubAdmin.branch", default = "main"),
  format = c("list", "json")
)
```

Arguments

`schema_version`	the version required. Defaults to "latest". Can be set through global option "hubAdmin.schema_version".
`branch`	the branch to download the schema from. Defaults to "main". Can be set through global option "hubAdmin.branch".
`format`	the format to return the schema in. Defaults to "list". Can be "list" or "json".

Value

The requested version of the tasks hubverse schema in the specified format.

Examples

```
download_tasks_schema()
download_tasks_schema(format = "json")
download_tasks_schema(schema_version = "v2.0.1")
options(hubAdmin.schema_version = "v3.0.1")
download_tasks_schema()
options(hubAdmin.schema_version = "latest")
```

get_array_schema_paths

Get potential paths to properties in a config file that can be arrays from a hubverse schema

Description

The functions identifies properties in a hubverse schema that can be arrays of vectors and returns the paths of such elements in a config. Properties that can be arrays of objects are ignored. Useful to determine elements in a config object that may need to be boxed. Note that any `"items"` elements indicate that the property is an array of objects and will be expanded with element index when applied to config objects.

Usage

```
get_array_schema_paths(schema)
```

Arguments

schema	a list representation of a hubverse schema

Value

a list where each element is character vector of a path to a property in the schema that can be an array of vectors. Elements returned as `"items"`

Examples

```
schema <- download_tasks_schema("v3.0.1")
get_array_schema_paths(schema)
```

schema_autobox

Box elements of a `<config>` *class object that can be arrays*

Description

Due to inconsistencies between R and JSON data types, in particular the fact that R has no concept of a scalar, when writing R list objects to JSON with `write_config()`, some properties in the output file may not conform to schema expectations. In particular, list elements that are vectors of length `1L` will be written as scalars, regardless of whether the schema expects an array. This function uses the hubverse schema to identify elements that can be arrays and "box" any such elements that exist in the `<config>` object and have a length of 1. This ensures that they are written out as arrays instead of scalars in JSON output files. The transformation is also applied to any properties that should be arrays covered by `additionalProperties` in the schema (e.g. custom task IDs).

Usage

```
schema_autobox(config, box_extra_paths = NULL)
```

Arguments

config	a <config> class object.
box_extra_paths	a list of character vectors of paths to elements in the <config> that can be arrays of vectors but are not covered by the schema. Elements in a path where arrays of objects are expected should be encoded as "items". See output of get_array_schema_paths() for more details, especially the examples.

Value

a <config> class object with list elements that can be arrays boxed.

Examples

```
config <- create_config(
  create_rounds(
    create_round(
      round_id_from_variable = TRUE,
      round_id = "origin_date",
      model_tasks = create_model_tasks(
        create_model_task(
          task_ids = create_task_ids(
            create_task_id("origin_date",
              required = NULL,
              optional = c(
                "2023-01-02",
                "2023-01-09",
                "2023-01-16"
              )
            ),
            create_task_id("location",
              required = "US",
              optional = c("01", "02", "04", "05", "06")
            ),
            create_task_id("horizon",
              required = 1L,
              optional = 2:4
            )
          ),
          output_type = create_output_type(
            create_output_type_mean(
              is_required = TRUE,
              value_type = "double",
              value_minimum = 0L
            )
          ),
          target_metadata = create_target_metadata(
```

```
            create_target_metadata_item(
              target_id = "inc hosp",
              target_name = "Weekly incident influenza hospitalizations",
              target_units = "rate per 100,000 population",
              target_keys = NULL,
              target_type = "discrete",
              is_step_ahead = TRUE,
              time_unit = "week"
            )
          )
        )
      ),
      submissions_due = list(
        relative_to = "origin_date",
        start = -4L,
        end = 2L
      )
    )
  )
)
schema_autobox(config)
schema_autobox(config) |>
  jsonlite::toJSON(
    auto_unbox = TRUE, na = "string",
    null = "null", pretty = TRUE
  )
```

`validate_config` *Validate a hub config file against a hubverse schema*

Description

This function validates a single hub config file against it's corresponding schema. Note that, for `tasks.json` config files, validation is performed in two stages:

1. Initial validation against the schema is performed using the `jsonvalidate` package which uses the `"ajv"` (Another JSON Schema Validator) validation engine.
2. If the initial validation is successful, additional dynamic validations are performed. This means that only after the initial validation passes, will any dynamic validation errors be detected.

Usage

```
validate_config(
  hub_path = ".",
  config = c("tasks", "admin"),
  config_path = NULL,
  schema_version = "from_config",
  branch = getOption("hubAdmin.branch", default = "main")
)
```

Arguments

hub_path	Path to a local hub directory.
config	Name of config file to validate. One of "tasks" or "admin".
config_path	Defaults to NULL which assumes all config files are in the hub-config directory in the root of hub directory. Argument config_path can be used to override default by providing a path to the config file to be validated.
schema_version	Character string specifying the json schema version to be used for validation. The default value "from_config" will use the version specified in the schema_version property of the config file. "latest" will use the latest version available in the hubverse schemas repository. Alternatively, a specific version of a schema (e.g. "v0.0.1") can be specified.
branch	The branch of the hubverse schemas repository from which to fetch schema. Defaults to "main". Can be set through global option "hubAdmin.branch".

Value

Returns the result of validation. If validation is successful, will return TRUE. If any validation errors are detected, returns FALSE with details of errors appended as a data.frame to an errors attribute. To access the errors table use attr(x, "errors") where x is the output of the function.

You can print a more concise and easier to view version of an errors table with view_config_val_errors().

See Also

view_config_val_errors()

Other functions supporting config file validation: validate_hub_config(), view_config_val_errors()

Examples

```
# Valid config file
validate_config(
  hub_path = system.file(
    "testhubs/simple/",
    package = "hubUtils"
  ),
  config = "tasks"
)
# Config file with errors
config_path <- system.file("error-schema/tasks-errors.json",
  package = "hubUtils"
)
validate_config(config_path = config_path, config = "tasks")
```

validate_hub_config *Validate Hub config files against hubverse schema.*

Description

Validate the `admin.json`, `tasks.json` and `model-metadata-schema.json` Hub config files in a single call. Note that, for `tasks.json` and `model-metadata-schema.json` config files, validation is performed in two stages:

1. Initial validation against the schema is performed using the `jsonvalidate` package which uses the `"ajv"` (Another JSON Schema Validator) validation engine. In the case of `model-metadata-schema.json`, `jsonvalidate` just checks that the file is valid JSON and can be parsed correctly.
2. If the initial validation is successful, additional dynamic validations are performed. This means that only after the initial validation passes, will any dynamic validation errors be detected.

Usage

```
validate_hub_config(
  hub_path = ".",
  schema_version = "from_config",
  branch = getOption("hubAdmin.branch", default = "main")
)
```

Arguments

`hub_path`	Path to a local hub directory.
`schema_version`	Character string specifying the json schema version to be used for validation. The default value `"from_config"` will use the version specified in the `schema_version` property of the config file. `"latest"` will use the latest version available in the hubverse schemas repository. Alternatively, a specific version of a schema (e.g. `"v0.0.1"`) can be specified.
`branch`	The branch of the hubverse schemas repository from which to fetch schema. Defaults to `"main"`. Can be set through global option "hubAdmin.branch".

Value

Returns a list of the results of validation, one for each `hub-config` file validated. A value of `TRUE` for a given file indicates that validation was successful. A value of `FALSE` for a given file indicates that validation errors were detected. Details of errors will be appended as a data.frame to an `errors` attribute. To access the errors table for a given element use `attr(x, "errors")` where `x` is the any element of the output of the function that is `FALSE`. You can print a more concise and easier to view version of an errors table with `view_config_val_errors()`.

See Also

`view_config_val_errors()`

Other functions supporting config file validation: `validate_config()`, `view_config_val_errors()`

Examples

```
validate_hub_config(
  hub_path = system.file(
    "testhubs/simple/",
    package = "hubUtils"
  )
)
```

`validate_model_metadata_schema` *Validate model-metadata-schema config file*

Description

Validate model-metadata-schema config file

Usage

```
validate_model_metadata_schema(hub_path = ".")
```

Arguments

`hub_path`	Path to a local hub directory.

Details

Checks that a `model-metadata-schema.json` config file exists in `hub-config`, can be successfully parsed and contains at least either a `model_id` property or both `team_abbr` and `model_abbr` properties.

Value

Returns the result of validation. If validation is successful, will return `TRUE`. If any validation errors are detected, returns `FALSE` with details of errors appended as a data.frame to an `errors` attribute. To access the errors table use `attr(x, "errors")` where `x` is the output of the function.

You can print a more concise and easier to view version of an errors table with `view_config_val_errors()`.

Examples

```
validate_model_metadata_schema(
  hub_path = system.file(
    "testhubs/simple/",
    package = "hubUtils"
  )
)
```

view_config_val_errors *Print a concise and informative version of validation errors table.*

Description

Print a concise and informative version of validation errors table.

Usage

```
view_config_val_errors(x)
```

Arguments

x	output of validate_config().

Value

prints the errors attribute of x in an informative format to the viewer. Only available in interactive mode.

See Also

validate_config()

Other functions supporting config file validation: validate_config(), validate_hub_config()

Examples

```
## Not run:
config_path <- system.file("error-schema/tasks-errors.json",
  package = "hubUtils"
)
validate_config(config_path = config_path, config = "tasks") |>
  view_config_val_errors()

## End(Not run)
```

write_config *Write config class object to a JSON file.*

Description

Write a **tasks** <config> class object to a tasks.json JSON file.

Usage

```
write_config(
  config,
  hub_path = ".",
  config_path = NULL,
  autobox = TRUE,
  box_extra_paths = NULL,
  overwrite = FALSE,
  silent = FALSE
)
```

Arguments

config	Object of class <config> to write to a JSON file.
hub_path	Path to the hub directory. Defaults to the current working directory. Ignored if config_path is specified.
config_path	Path to write the config object to. If NULL defaults to hub-config/tasks.json within hub_path. If specified, overrides hub_path.
autobox	Logical. Whether to automatically box vectors of length 1L that should be arrays in the JSON output according to the hubverse schema. See schema_autobox() for more details.
box_extra_paths	a list of character vectors of paths to elements in the <config> that can be arrays of vectors but are not covered by the schema. Elements in a path where arrays of objects are expected should be encoded as "items". See output of get_array_schema_paths() for more details, especially the examples.
overwrite	Logical. Whether to overwrite the file if it already exists.
silent	Logical. Whether to suppress informational messages.

Value

TRUE invisibly.

Examples

```
# Create rounds object
rounds <- create_rounds(
  create_round(
    round_id_from_variable = TRUE,
    round_id = "origin_date",
    model_tasks = create_model_tasks(
      create_model_task(
        task_ids = create_task_ids(
          create_task_id("origin_date",
            required = NULL,
            optional = c(
              "2023-01-02",
              "2023-01-09",
```

```
            "2023-01-16"
          )
        ),
        create_task_id("location",
          required = "US",
          optional = c("01", "02", "04", "05", "06")
        ),
        create_task_id("horizon",
          required = 1L,
          optional = 2:4
        )
      ),
      output_type = create_output_type(
        create_output_type_mean(
          is_required = TRUE,
          value_type = "double",
          value_minimum = 0L
        )
      ),
      target_metadata = create_target_metadata(
        create_target_metadata_item(
          target_id = "inc hosp",
          target_name = "Weekly incident influenza hospitalizations",
          target_units = "rate per 100,000 population",
          target_keys = NULL,
          target_type = "discrete",
          is_step_ahead = TRUE,
          time_unit = "week"
        )
      )
    )
  ),
  submissions_due = list(
    relative_to = "origin_date",
    start = -4L,
    end = 2L
  )
)

# Create config object
config <- create_config(rounds)

# Create temporary hub and write config
withr::with_tempdir({
  dir.create("hub-config")

  # Write config
  write_config(config, hub_path = ".")

  # Read and print tasks.json
  cat(readLines(file.path("hub-config", "tasks.json")), sep = "\n")
```

```
  # Validate config
  validate_config(hub_path = ".")

  # Add a custom additional property to the first round of the config
  rounds[[1]][[1]]$extra_array_property <- "length_1L_property"
  config <- create_config(rounds)

  write_config(
    config = config, hub_path = ".", overwrite = TRUE,
    box_extra_paths = list(c("rounds", "items", "extra_array_property"))
  )

  # Read and print tasks.json again
  cat(readLines(file.path("hub-config", "tasks.json")), sep = "\n")
})
```

Index