Package: dataclass (via r-universe)

February 20, 2025

Type Package

Title Easily Create Structured Lists or Data Frames with Input Validation

Version 1.0.0

Description Easily define templates for lists and data frames that validate each element. Specify the expected type (i.e., character, numeric, etc), expected length, minimum and maximum values, allowable values, and more for each element in your data. Decide whether violations of these expectations should throw an error or a warning. This package is useful for validating data within R processes which pull from dynamic data sources such as databases and web APIs to provide an extra layer of validation around input and output data.

License MIT + file LICENSE

Encoding UTF-8

Imports purrr, rlang, glue, magrittr, tibble, cli, dplyr, lifecycle

RoxygenNote 7.3.2

NeedsCompilation no

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Author Chris Walker [aut, cre, cph]

Maintainer Chris Walker <walkerjameschris@gmail.com>

Date/Publication 2024-09-24 13:10:23 UTC

Additional_repositories https://cranhaven.r-universe.dev

Repository https://cranhaven.r-universe.dev

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/dataclass

RemoteSha deb25ecd051932515b82da48e290c67acd2397eb

RemoteSubdir dataclass

Contents

`any_obj` *Validator: Allow any object*

Description

This function is used to bypass dataclass checks for a given element. If you do not want dataclass to check a given element, set the element equal to any_obj() to allow any object. Keep in mind that while dataclass will bypass the check, the object must still be a valid R object. Furthermore, if you are using dataclass to create a tibble, then the object must be a valid tibble column type, even if additional checks are not considered. This can be dangerous because dataclass is designed to check objects, not bypass them. Use this validator sparingly and consider how you can write a stricter dataclass.

Usage

```
any_obj()
```

Value

A function with the following properties:

- Always returns TRUE
- Bypasses any dataclass checks

Examples

```
# Define a dataclass:
my_dataclass <-
  dataclass(
    date_val = dte_vec(),
    anything = any_obj()
  )

# While `date_val` must be a date, `anything` can be any value!
```

```
my_dataclass(
  date_val = as.Date("2022-01-01"),
  anything = lm(vs ~ am, mtcars)
)

my_dataclass(
  date_val = as.Date("2022-01-01"),
  anything = c(1, 2, 3, 4, 5)
)

my_dataclass(
  date_val = as.Date("2022-01-01"),
  anything = list(a = 1, b = 2)
)
```

`atm_vec` *Validator: Check if element is atomic*

Description

This function is used to check whether something is atomic. Atomic elements are represented by simple vectors, (i.e., numeric, logical, character) but also include special vectors like date vectors. You can use this function to check the length of a vector. You can also specify the level of a violation. If level is set to "warn" then invalid inputs will warn you. However, if level is set to "error" then invalid inputs will abort.

Usage

```
atm_vec(
  max_len = Inf,
  min_len = 1,
  level = "error",
  allow_na = FALSE,
  allow_dups = TRUE
)
```

Arguments

`max_len`	The maximum length of an atomic element
`min_len`	The minimum length of an atomic element
`level`	Setting "warn" throws a warning, setting "error" halts
`allow_na`	Should NA values be allowed?
`allow_dups`	Should duplicates be allowed?

Value

A function with the following properties:

- Checks whether something is atomic
- Determines whether the check will throw warning or error
- Optionally checks for element length

Examples

```
# Define a dataclass for testing atomic:
my_dataclass <-
  dataclass(
    num_val = num_vec(),
    # Setting warn means a warning will occur if violation is found
    # The default is "error" which is stricter and will halt upon violation
    atm_val = atm_vec(level = "warn")
  )

# While `num_val` must be a number, `atm_val` can be any atomic element!
my_dataclass(
  num_val = c(1, 2, 3),
  atm_val = Sys.Date()
)

my_dataclass(
  num_val = c(1, 2, 3),
  atm_val = c(TRUE, FALSE)
)

my_dataclass(
  num_val = c(1, 2, 3),
  atm_val = c("This is", "a character!")
)
```

---

chr_vec *Validator: Check if element is a character*

Description

This function is used to check whether something is a character. You can use this function to check the length and allowable values of character. You can also specify the level of a violation. If level is set to "warn" then invalid inputs will warn you. However, if level is set to "error" then invalid inputs will abort.

Usage

```
chr_vec(
  max_len = Inf,
  min_len = 1,
  allowed = NA,
  level = "error",
  allow_na = FALSE,
  allow_dups = TRUE
)
```

Arguments

max_len	The maximum length of a character element
min_len	The minimum length of a character element
allowed	A vector of allowable values
level	Setting "warn" throws a warning, setting "error" halts
allow_na	Should NA values be allowed?
allow_dups	Should duplicates be allowed?

Value

A function with the following properties:

- Checks whether something is a character vector
- Determines whether the check will throw warning or error
- Optionally checks for element length
- Optionally checks for allowable values

Examples

```
# Define a dataclass for testing characters:
my_dataclass <-
  dataclass(
    string = chr_vec(allowed = c("this", "or", "that")),
    other_string = chr_vec()
  )

# `string` must be one of these: `c("this", "or", "that")`
my_dataclass(
  string = "this",
  other_string = "I can be anything I want (as long as I am a string)"
)
```

`dataclass` *Construct a dataclass in R*

Description

Building a dataclass is easy! Provide names for each of the elements you want in your dataclass and an associated validator. The dataclass package comes with several built in validators, but you can define a custom validator as an anonymous function or named function to be bundled with your dataclass.

Usage

```
dataclass(...)
```

Arguments

...	Elements to validate (i.e., dte_vec() will validate a date vector)

Details

dataclass() will return a new function with named arguments for each of the elements you define here. If you want to use your dataclass on data frames or tibbles you must pass the dataclass to data_validator() to modify behavior.

Value

A function with the following properties:

- An argument for each element provided to dataclass()
- Each argument in the returned function will validate inputs
- An error occurs if new elements passed to the returned function are invalid
- List is returned if new elements passed to the returned function are valid

Examples

```
my_dataclass <- dataclass(
  min_date = dte_vec(1), # Ensures min_date is a date vector of length 1
  max_date = dte_vec(1), # Ensures max_date is a date vector of length 1
  run_data = df_like(), # Ensures run_date is a data object (i.e. tibble)
  run_note = chr_vec(1) # Ensures run_note is a character vector of length 1
)

# This returns a validated list!
my_dataclass(
  min_date = as.Date("2022-01-01"),
  max_date = as.Date("2023-01-01"),
  run_data = head(mtcars, 2),
  run_note = "A note!"
```

```
)

# An example with anonymous functions
a_new_dataclass <-
  dataclass(
    start_date = dte_vec(1),
    # Ensures calculation is a column in this data and is data like
    results_df = function(df) "calculation" %in% colnames(df)
  )

# Define a dataclass for creating data! Wrap in data_validator():
my_df_dataclass <-
  dataclass(
    dte_col = dte_vec(),
    chr_col = chr_vec(),
    # Custom column validator ensures values are positive!
    new_col = function(x) all(x > 0)
  ) |>
  data_validator()

# Validate a data frame or data frame like objects!
data.frame(
  dte_col = as.Date("2022-01-01"),
  chr_col = "String!",
  new_col = 100
) |>
  my_df_dataclass()
```

`data_validator` *Convert a dataclass to a data frame validator*

Description

If you intend to use your dataclass to validate data frame like object such as tibbles, data frames, or data tables, pass the dataclass into this function to modify behavior.

Usage

```
data_validator(x, strict_cols = TRUE)
```

Arguments

`x`	A dataclass object
`strict_cols`	Should additional columns be allowed in the output?

Value

A function with the following properties:

- A modified dataclass function designed to accept data frames

- A single argument to test new data frames
- Each column in a new data frame will be tested
- An error occurs if new data passed to the returned function are invalid
- Data is returned if new data passed to the returned function are valid

Examples

```
# Define a dataclass for creating data! Pass to data_validator():
my_df_dataclass <-
  dataclass(
    dte_col = dte_vec(),
    chr_col = chr_vec(),
    # Custom column validator ensures values are positive!
    new_col = function(x) all(x > 0)
  ) |>
  data_validator()

# Validate a data frame or data frame like objects!
data.frame(
  dte_col = as.Date("2022-01-01"),
  chr_col = "String!",
  new_col = 100
) |>
  my_df_dataclass()

# Allow additional columns in output
test_df_class <-
  dataclass(
    dte_col = dte_vec()
  ) |>
  data_validator(strict_cols = FALSE)

tibble::tibble(
  dte_col = as.Date("2022-01-01"),
  other_col = "a"
) |>
  test_df_class()
```

`df_like` *Validator: Check if element is a data like object*

Description

This function is used to check whether something is data like. You can use this function to check the data row count. You can also specify the level of a violation. If level is set to "warn" then invalid inputs will warn you. However, if level is set to "error" then invalid inputs will abort.

Usage

```
df_like(max_row = Inf, min_row = 1, level = "error")
```

Arguments

max_row	The maximum row count of a data element
min_row	The minimum row count of a data element
level	Setting "warn" throws a warning, setting "error" halts

Value

A function with the following properties:

- Checks whether something is a data frame like object
- Determines whether the check will throw warning or error
- Optionally checks for row count

Examples

```
# Define a dataclass for testing data:
my_dataclass <-
  dataclass(
    df = df_like(100)
  )

# `df` must be a data like object with at most 100 rows!
my_dataclass(
  df = mtcars
)
```

---

dte_vec *Validator: Check if element is a date*

Description

This function is used to check whether something is a date. You can use this function to check the length of a date vector. You can also specify the level of a violation. If level is set to "warn" then invalid inputs will warn you. However, if level is set to "error" then invalid inputs will abort.

Usage

```
dte_vec(
  max_len = Inf,
  min_len = 1,
  level = "error",
  allow_na = FALSE,
  allow_dups = TRUE
)
```

Arguments

max_len	The maximum length of a date element
min_len	The minimum length of a date element
level	Setting "warn" throws a warning, setting "error" halts
allow_na	Should NA values be allowed?
allow_dups	Should duplicates be allowed?

Value

A function with the following properties:

- Checks whether something is a date
- Determines whether the check will throw warning or error
- Optionally checks for element length

Examples

```
# Define a dataclass for testing dates:
my_dataclass <-
  dataclass(
    num_val = num_vec(),
    # Setting warn means a warning will occur if violation is found
    # The default is "error" which is stricter and will halt upon violation
    dte_val = dte_vec(level = "warn")
  )

# While `num_val` must be a number, `dte_val` must be a date!
my_dataclass(
  num_val = c(1, 2, 3),
  dte_val = Sys.Date()
)

my_dataclass(
  num_val = c(1, 2, 3),
  dte_val = as.Date("2022-01-01")
)

my_dataclass(
  num_val = c(1, 2, 3),
  dte_val = as.Date(c("2022-01-01", "2023-01-01"))
)
```

enforce_types *Enforced typing in R* **[Experimental]**

Description

This function allows for simple type enforcement in R inspired by C++ and other compiled languages. There are currently six primitive types which the function handles:

Usage

```
enforce_types(level = c("error", "warn", "none"))
```

Arguments

level	Should type failures error, warn, or be skipped (none)?

Details

- int(): An integer specified with the *L* syntax (i.e., '1L")
- chr(): A string or character
- lgl(): A boolean TRUE/FALSE
- dbl(): A double or numeric value
- tbl(): A data frame or tibble (types within the data frame are not checked)

You can also provide default arguments within the parenthesis of the type. This is shown in the example below. You can provide new arguments as well. The function has knowledge of the function declaration when it runs. Note: types are checked at runtime not when the function is declared.

Examples

```
foo <- function(
  x = int(1L),
  y = chr("Hello!"),
  z = lgl(TRUE),
  a = dbl(1.1),
  b = tbl(mtcars),
  c = NULL # This argument will not be checked
) {

  # Simply place enforce_types() in your function header!
  dataclass::enforce_types()

  # Function logic ...
}

# This run the function with the type defaults
foo()
```

```
# This will check types but for new arguments
foo(2L, "Hi!", FALSE, 1.2, mtcars)

# This would fail because types are incorrect!
# foo(1.1, FALSE, NULL, "Hi", list())

# This function will only warn when there are type failures
bar <- function(x = int(1)) {
  dataclass::enforce_types("warn")
}
```

`lgl_vec` *Validator: Check if element is a logical*

Description

This function is used to check whether something is a logical. You can use this function to check the length of a logical vector. You can also specify the level of a violation. If level is set to "warn" then invalid inputs will warn you. However, if level is set to "error" then invalid inputs will abort.

Usage

```
lgl_vec(max_len = Inf, min_len = 1, level = "error", allow_na = FALSE)
```

Arguments

`max_len`	The maximum length of a logical element
`min_len`	The minimum length of a logical element
`level`	Setting "warn" throws a warning, setting "error" halts
`allow_na`	Should NA values be allowed?

Value

A function with the following properties:

- Checks whether something is a logical vector
- Determines whether the check will throw warning or error
- Optionally checks for element length

Examples

```
# Define a dataclass for testing logicals:
my_dataclass <-
  dataclass(
    bool = lgl_vec()
  )
```

```
# `bool` must be a logical vector of any length!
my_dataclass(
  bool = TRUE
)
```

---

num_vec *Validator: Check if element is a number*

Description

This function is used to check whether something is a number. You can use this function to check the length and min-max of a number vector. You can also specify the level of a violation. If level is set to "warn" then invalid inputs will warn you. However, if level is set to "error" then invalid inputs will abort.

Usage

```
num_vec(
  max_len = Inf,
  min_len = 1,
  max_val = Inf,
  min_val = -Inf,
  allowed = NA,
  level = "error",
  allow_na = FALSE,
  allow_dups = TRUE
)
```

Arguments

max_len	The maximum length of a numeric element
min_len	The minimum length of a numeric element
max_val	The maximum value of a numeric element
min_val	The minimum value of a numeric element
allowed	A vector of allowable values
level	Setting "warn" throws a warning, setting "error" halts
allow_na	Should NA values be allowed?
allow_dups	Should duplicates be allowed?

Value

A function with the following properties:

- Checks whether something is a number vector
- Determines whether the check will throw warning or error
- Optionally checks for element length
- Optionally checks for allowable values
- Optionally checks for max/min

Examples

```
# Define a dataclass for testing numbers:
my_dataclass <-
  dataclass(
    dte_val = dte_vec(),
    # Setting warn means a warning will occur if violation is found
    # The default is "error" which is stricter and will halt upon violation
    # We also set allowed to 0 and 1 which means elements must be 0 or 1
    num_val = num_vec(level = "warn", allowed = c(0, 1))
  )

# While `dte_val` must be a date, `num_val` must be 0 or 1!
my_dataclass(
  dte_val = Sys.Date(),
  num_val = c(0, 1, 1, 0, 1)
)

my_dataclass(
  dte_val = Sys.Date(),
  num_val = 1
)

# Set min and max requirements!
test_dataclass <-
  dataclass(
    num = num_vec(min_val = 1, max_val = 100)
  )

# Value must be between 1 and 10 inclusive!
test_dataclass(num = 10.03012)
```

Index