Package 'interface'

July 22, 2025

Type Package

Title Runtime Type System

Version 0.1.2

URL https://github.com/dereckmezquita/interface

Author Dereck Mezquita [aut, cre] (ORCID: <https://orcid.org/0000-0002-9307-6762>)

Maintainer Dereck Mezquita <dereck@mezquita.io>

Description Provides a runtime type system, allowing users to define and implement interfaces, enums, typed data.frame/data.table, as well as typed functions. This package enables stricter type checking and validation, improving code structure, robustness and reliability.

License MIT + file LICENSE

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Suggests testthat (>= 3.0.0), knitr, rmarkdown, box, rcmdcheck, data.table

NeedsCompilation no

Repository CRAN

Date/Publication 2024-09-10 09:10:05 UTC

Contents

`==.enum` *Equality comparison for enum objects*

Description

Compares two enum objects or an enum object with a character value.

Usage

```
## S3 method for class 'enum'
e1 == e2
```

Arguments

`e1`	First enum object
`e2`	Second enum object or a character value

Value

Logical value indicating whether the two objects are equal

enum *Create an enumerated type*

Description

Creates an enumerated type with a fixed set of possible values. This function returns an enum generator, which can be used to create enum objects with values restricted to the specified set.

Usage

```
enum(...)
```

Arguments

`...`	The possible values for the enumerated type. These should be unique character strings.

Value

A function (enum generator) of class 'enum_generator' that creates enum objects of the defined type. The returned function takes a single argument and returns an object of class 'enum'.

See Also

`interface` for using enums in interfaces

Examples

```
# Create an enum type for colors
Colors <- enum("red", "green", "blue")

# Create enum objects
my_color <- Colors("red")
print(my_color)  # Output: Enum: red

# Trying to create an enum with an invalid value will raise an error
try(Colors("yellow"))

# Enums can be used in interfaces
ColoredShape <- interface(
  shape = character,
  color = Colors
)

my_shape <- ColoredShape(shape = "circle", color = "red")

# Modifying enum values
my_shape$color$value <- "blue"  # This is valid
try(my_shape$color$value <- "yellow")  # This will raise an error
```

fun *Create a typed function*

Description

Defines a function with specified parameter types and return type. Ensures that the function's arguments and return value adhere to the specified types.

Usage

```
fun(...)
```

Arguments

...	Named arguments defining the function parameters and their types, including 'return' for the expected return type(s) and 'impl' for the function implementation.

Details

The 'fun' function allows you to define a function with strict type checking for its parameters and return value. This ensures that the function receives arguments of the correct types and returns a value of the expected type. The 'return' and 'impl' arguments should be included in the ... parameter list.

Value

A function of class 'typed_function' that enforces type constraints on its parameters and return value. The returned function has the same signature as the implementation function provided in the 'impl' argument.

Examples

```
# Define a typed function that adds two numbers
add_numbers <- fun(
  x = numeric,
  y = numeric,
  return = numeric,
  impl = function(x, y) {
    return(x + y)
  }
)

# Valid call
print(add_numbers(1, 2))  # [1] 3

# Invalid call (throws error)
try(add_numbers("a", 2))
```

```
# Define a typed function with multiple return types
concat_or_add <- fun(
  x = c(numeric, character),
  y = numeric,
  return = c(numeric, character),
  impl = function(x, y) {
    if (is.numeric(x)) {
      return(x + y)
    } else {
      return(paste(x, y))
    }
  }
)

# Valid calls
print(concat_or_add(1, 2))     # [1] 3
print(concat_or_add("a", 2))   # [1] "a 2"
```

`handle_violation` *Handle violations based on the specified action*

Description

Handles violations by either throwing an error, issuing a warning, or doing nothing, depending on the specified action.

Usage

```
handle_violation(message, action)
```

Arguments

`message`	The error message to be handled.
`action`	The action to take: "error", "warning", or "silent".

`interface` *Define an interface*

Description

An interface defines a structure with specified properties and their types or validation functions. This is useful for ensuring that objects adhere to a particular format and type constraints.

Usage

```
interface(..., validate_on_access = FALSE, extends = list())
```

Arguments

`...`	Named arguments defining the properties and their types or validation functions.
`validate_on_access`	Logical, whether to validate properties on access (default: FALSE).
`extends`	A list of interfaces that this interface extends.

Value

A function of class 'interface' that creates objects implementing the defined interface. The returned function takes named arguments corresponding to the interface properties and returns an object of class 'interface_object'.

Examples

```
# Define an interface for a person
Person <- interface(
  name = character,
  age = numeric,
  email = character
)

# Create an object that implements the Person interface
john <- Person(
  name = "John Doe",
  age = 30,
  email = "john@example.com"
)

# Using enum in an interface
Colors <- enum("red", "green", "blue")
ColoredShape <- interface(
  shape = character,
  color = Colors
)

my_shape <- ColoredShape(shape = "circle", color = "red")

# In-place enum declaration
Car <- interface(
  make = character,
  model = character,
  color = enum("red", "green", "blue")
)

my_car <- Car(make = "Toyota", model = "Corolla", color = "red")
```

print.enum *Print method for enum objects*

Description

Prints a human-readable representation of an enum object.

Usage

```
## S3 method for class 'enum'
print(x, ...)
```

Arguments

x	An enum object
...	Additional arguments (not used)

Value

No return value, called for side effects. Prints a string representation of the enum object to the console.

print.enum_generator *Print method for enum generators*

Description

Prints a human-readable representation of an enum generator, showing all possible values.

Usage

```
## S3 method for class 'enum_generator'
print(x, ...)
```

Arguments

x	An enum generator function
...	Additional arguments (not used)

Value

No return value, called for side effects. Prints a string representation of the enum generator to the console.

print.interface_object *Print method for interface objects*

Description

Print method for interface objects

Usage

```
## S3 method for class 'interface_object'
print(x, ...)
```

Arguments

x	An object implementing an interface
...	Additional arguments (not used)

Value

No return value, called for side effects. Prints a human-readable representation of the interface object to the console.

print.typed_frame *Print method for typed data frames*

Description

Provides a custom print method for typed data frames, displaying their properties and validation status.

Usage

```
## S3 method for class 'typed_frame'
print(x, ...)
```

Arguments

x	A typed data frame.
...	Additional arguments passed to print.

Value

No return value, called for side effects. Prints a summary of the typed data frame to the console, including its dimensions, column specifications, frame properties, and a preview of the data.

print.typed_function *Print method for typed functions*

Description

Provides a custom print method for typed functions, displaying their parameter types and return type.

Usage

```
## S3 method for class 'typed_function'
print(x, ...)
```

Arguments

x	A typed function.
...	Additional arguments (not used).

Value

No return value, called for side effects. Prints a human-readable representation of the typed function to the console, showing the argument types and return type.

rbind.typed_frame *Combine typed data frames row-wise*

Description

This function combines multiple typed data frames row-wise, ensuring type consistency and applying row validation rules. It extends the base rbind function by adding type checks and row validation based on the specified rules for typed data frames.

Usage

```
## S3 method for class 'typed_frame'
rbind(..., deparse.level = 1)
```

Arguments

...	Typed data frames to combine.
deparse.level	See rbind.

Details

This version of rbind for typed_frame performs extra type checking and row validation to ensure consistency and adherence to specified rules. Refer to the base rbind documentation for additional details on combining data frames: rbind.

Value

The combined typed data frame. The returned object is of class 'typed_frame' and inherits all properties (column types, validation rules, etc.) from the first data frame in the list.

`type.frame` *Create a typed data frame*

Description

Creates a data frame with specified column types and validation rules. Ensures that the data frame adheres to the specified structure and validation rules during creation and modification.

Usage

```
type.frame(
  frame,
  col_types,
  freeze_n_cols = TRUE,
  row_callback = NULL,
  allow_na = TRUE,
  on_violation = c("error", "warning", "silent")
)
```

Arguments

`frame`	The base data structure (e.g., data.frame, data.table).
`col_types`	A list of column types and validators.
`freeze_n_cols`	Logical, whether to freeze the number of columns (default: TRUE).
`row_callback`	A function to validate and process each row (optional).
`allow_na`	Logical, whether to allow NA values (default: TRUE).
`on_violation`	Action to take on violation: "error", "warning", or "silent" (default: "error").

Details

The 'type.frame' function defines a blueprint for a data frame, specifying the types of its columns and optional validation rules for its rows. When a data frame is created or modified using this blueprint, it ensures that all data adheres to the specified rules.

Value

A function that creates typed data frames. When called, this function returns an object of class 'typed_frame' (which also inherits from the base frame class used, i.e. data.frame, data.table).

Examples

```
# Define a typed data frame
PersonFrame <- type.frame(
    frame = data.frame,
    col_types = list(
        id = integer,
        name = character,
        age = numeric,
        is_student = logical
    )
)

# Create a data frame
persons <- PersonFrame(
    id = 1:3,
    name = c("Alice", "Bob", "Charlie"),
    age = c(25, 30, 35),
    is_student = c(TRUE, FALSE, TRUE)
)

print(persons)

# Invalid modification (throws error)
try(persons$id <- letters[1:3])

# Adding a column (throws error if freeze_n_cols is TRUE)
try(persons$yeet <- letters[1:3])
```

---

validate_property *Validate a property against a given type or validation function*

Description

Validates a property to ensure it matches the expected type or satisfies the given validation function.

Usage

```
validate_property(name, value, validator)
```

Arguments

name	The name of the property being validated.
value	The value of the property.
validator	The expected type or a custom validation function.

Details

This function supports various types of validators: - Enum generators - Lists of multiple allowed types - Interface objects - Built-in R types (character, numeric, logical, integer, double, complex) - data.table and data.frame types - Custom validation functions

Value

Returns NULL if the validation passes, otherwise returns a character string containing an error message describing why the validation failed.

---

wrap_fun_in_all *Modify a user-defined function to return a single logical value*

Description

Modifies a user-defined function to wrap its body in an all() call, ensuring that it returns a single logical value instead of a vector.

It uses bquote() to create a new body for the function. The .() inside bquote() inserts the original body of the function. The all() function wraps around the original body.

Usage

```
wrap_fun_in_all(user_fun)
```

Arguments

user_fun	A user-defined function.

Value

The modified function.

[<-.typed_frame *Modify a typed data frame using [ ]*

Description

Allows modifying a typed data frame using the [ ] operator, with validation checks.

Usage

```
## S3 replacement method for class 'typed_frame'
x[i, j] <- value
```

Arguments

x	A typed data frame.
i	Row index.
j	Column index or name.
value	The new value to assign.

Value

The modified typed data frame.

$.enum *Get value from enum object*

Description

Retrieves the value of an enum object.

Usage

```
## S3 method for class 'enum'
x$name
```

Arguments

x	An enum object
name	The name of the field to access (should be "value")

Value

The value of the enum object

$.interface_object *Get a property from an interface object*

Description

Get a property from an interface object

Usage

```
## S3 method for class 'interface_object'
x$name
```

Arguments

x	An interface object
name	The name of the property to get

Value

The value of the specified property. The class of the returned value depends on the property's type as defined in the interface.

$<-.enum *Set value of enum object*

Description

Sets the value of an enum object. The new value must be one of the valid enum values.

Usage

```
## S3 replacement method for class 'enum'
x$name <- value
```

Arguments

x	An enum object
name	The name of the field to set (should be "value")
value	The new value to set

Value

The updated enum object

$<-.interface_object *Set a property in an interface object*

Description

Set a property in an interface object

Usage

```
## S3 replacement method for class 'interface_object'
x$name <- value
```

Arguments

x	An interface object
name	The name of the property to set
value	The new value for the property

Value

The modified interface object, invisibly.

`$<-.typed_frame` *Modify a typed data frame using $*

Description

Allows modifying a typed data frame using the $ operator, with validation checks.

Usage

```
## S3 replacement method for class 'typed_frame'
x$col_name <- value
```

Arguments

`x`	A typed data frame.
`col_name`	The name of the column to modify or add.
`value`	The new value to assign.

Value

The modified typed data frame.

Index