Package: sicher (via r-universe)

June 4, 2026

Type Package

Title Runtime Type Checking

Version 0.1.1

Description Provides a lightweight runtime type system for 'R' that enables developers to declare and enforce variable types during execution. Inspired by 'TypeScript', the package introduces intuitive syntax for annotating variables and validating data structures, helping catch type-related errors early and making 'R' code more robust and easier to maintain.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL https://github.com/feddelegrand7/sicher

BugReports https://github.com/feddelegrand7/sicher/issues

RoxygenNote 7.3.2

Imports glue (>= 1.8.0)

Suggests testthat

Repository https://feddelegrand7.r-universe.dev

Date/Publication 2026-06-04 10:21:01 UTC

RemoteUrl https://github.com/feddelegrand7/sicher

RemoteRef HEAD

RemoteSha 7aac394840efbb207fc08ba5d8da3b6934364e35

Contents

`[.sicher_type` *Vector Size Operator for sicher_type*

Description

Creates a type that checks for a specific vector length.

Usage

```
## S3 method for class 'sicher_type'
type[size]
```

Arguments

type	A sicher_type object
size	The required length (non-negative integer)

Value

A new sicher_type that checks for the specified length

Examples

```
vec %:% Numeric[3] %<-% c(1, 2, 3)
try(vec <- c(1, 2)) # Error: wrong length
try(vec <- c("a", "b", "c"))  # Error: wrong type
```

%:% *Type annotation operator*

Description

Creates a typed variable annotation used together with '%<-%'.

Usage

```
name %:% type
```

Arguments

name	Variable name (unevaluated).
type	Type specification (e.g., Integer, String, Double).

Value

A typed annotation object.

Examples

```
x %:% Integer %<-% 5L
name %:% String %<-% "Alice"
id %:% (Integer | String) %<-% 42L
```

%<-% *Type-checked assignment operator*

Description

Completes the typed assignment started with '*%:%*'.

Usage

```
typed_annotation %<-% value
```

Arguments

typed_annotation	Result of '*%:%*'.
value	Value to assign.

Value

Invisibly returns the assigned value.

Examples

```
x %:% Integer %<-% 5L
y %:% Double %<-% 3.14
name %:% String %<-% "Bob"
flag %:% Bool %<-% TRUE
```

Any *Built-in Any Type*

Description

A type that accepts any value.

Usage

```
Any
```

Format

An object of class `sicher_type` of length 2.

Between — *Between — closed-interval numeric range constraint*

Description

Creates a type that accepts numeric values within the closed interval `[min, max]`. Every element of a vector must satisfy the bounds. NA values are always rejected.

Usage

```
Between(min, max)
```

Arguments

`min`	A single non-NA numeric scalar giving the lower bound (inclusive).
`max`	A single non-NA numeric scalar giving the upper bound (inclusive).

Value

A new `sicher_type` that checks the value is numeric, NA-free, and that all elements lie within `[min, max]`.

Examples

```
age   %:% Between(0, 150)   %<-% 30
score %:% Between(0.0, 1.0) %<-% 0.95
try(score <- 1.5)   # Error: 1.5 is outside [0, 1]
try(score <- NA_real_)  # Error: value contains NA(s)

# Works with integer storage (is.numeric(1L) is TRUE in R)
count %:% Between(0L, 100L) %<-% 42L
```

Bool — *Built-in Boolean Type*

Description

A type that checks for logical vectors.

Usage

```
Bool
```

Format

An object of class `sicher_type` of length 2.

check_type *Type Checking Function*

Description

Validates that a value conforms to a specified type. This is the core validation function used internally by the type system, but can also be called directly for manual type checking.

Usage

```
check_type(value, type, context = NULL)
```

Arguments

value	The value to check
type	A sicher_type, sicher_union, or sicher_readonly object
context	Optional character string describing where the check is occurring (used in error messages)

Details

This function:

- Checks if a value matches a type specification
- Handles union types (checks if value matches any type in the union)
- Handles readonly types (strips the readonly modifier before checking)
- Provides detailed error messages when checks fail

Value

Returns ‘TRUE‘ invisibly if the value matches the type, otherwise throws an error with a descriptive message.

See Also

create_type for creating custom types

Examples

```
# Direct type checking
check_type(5L, Integer)  # Returns TRUE
try(check_type("hello", Integer))  # Throws error

# With context for better error messages
try(check_type(5L, String, context = "user_name"))

# With union types
```

```
check_type(5L, Integer | String)  # Returns TRUE
try(check_type(5.5, Integer | String))  # Throws error
```

`create_dataframe_type` *Create a Data Frame Type with Column Specification*

Description

Builds a type that validates a data frame's column names and their types. Each column is treated as a vector and checked against the provided sicher_type (or union) specification. Optional columns may be declared with 'Optional()'.

Usage

```
create_dataframe_type(col_spec)
```

Arguments

`col_spec`	A named list where names are column names and values are sicher_type or sicher_union objects describing the expected column type.

Value

A sicher_type representing the data frame schema.

Examples

```
PersonDF <- create_dataframe_type(list(
  name = String,
  age = Numeric,
  height = Optional(Numeric)
))

df %:% PersonDF %<-% data.frame(
  name = c("Alice", "Bob"),
  age = c(25, 30)
)
```

`create_list_type` *Create a List Type with Specific Structure*

Description

Creates a type that checks for lists with specific named elements and their types. Similar to object types in TypeScript/JavaScript.

Usage

```
create_list_type(type_spec)
```

Arguments

`type_spec`	A named list where names are field names and values are sicher_type objects

Value

A sicher_type that validates list structure

Examples

```
# Define a User type
User <- create_list_type(list(
  name = String,
  age = Numeric,
  preferences = create_list_type(list(
    color = String,
    movie = String
  ))
))

# Use it
user %:% User %<-% list(
  name = "Alice",
  age = 25,
  preferences = list(color = "red", movie = "batman")
)
```

`create_type` *Create a Custom Type*

Description

Creates a new type object for use in the type checking system. A type consists of a name (for error messages) and a checker function (for validation).

Usage

```
create_type(name, checker)
```

Arguments

`name`	A single character string representing the type name. This name will be displayed in error messages when type checking fails.
`checker`	A function that takes a single argument and returns 'TRUE' if the value matches the type, 'FALSE' otherwise. The checker function should be a predicate (e.g., 'is.numeric', 'is.character').

Details

This is the fundamental building block of the type system. Built-in types like 'Integer', 'Double', and 'String' are all created using this function.

The checker function should:

- Accept a single argument (the value to check)
- Return 'TRUE' if the value is valid for this type
- Return 'FALSE' if the value is invalid
- Not throw errors (error handling is done by 'check_type')

Value

An object of class '"sicher_type"' containing:

name The type name as a character string

check The checker function

See Also

`check_type` for type validation, `Scalar` for creating scalar type variants, `Readonly` for creating readonly type variants

Examples

```
# Create a custom positive number
Positive <- create_type("positive", function(x) {
  is.numeric(x) && all(x > 0)
})

# Use it in type annotations
age %:% Positive %<-% 25
try(age <- -5)  # Error: Type error

# Create a custom email type
Email <- create_type("email", function(x) {
  is.character(x) &&
    length(x) == 1 &&
    grepl("^[^@]+@[^@]+\\.[^@]+$", x)
})

user_email %:% Email %<-% "user@example.com"

# Create a type for even integers
EvenInt <- create_type("even_int", function(x) {
  is.integer(x) && all(x %% 2 == 0)
})

value %:% EvenInt %<-% 4L
try(value <- 5L)  # Error: Type error

# Create a type that checks data frame structure
PersonDF <- create_type("person_df", function(x) {
  is.data.frame(x) &&
    all(c("name", "age") %in% names(x)) &&
    is.character(x$name) &&
    is.numeric(x$age)
})
```

DataFrame *Built-in DataFrame Type*

Description

A type that checks for data.frame objects.

Usage

```
DataFrame
```

Format

An object of class `sicher_type` of length 2.

Double *Built-in Double Type*

Description

A type that checks for double-precision numeric vectors.

Usage

```
Double
```

Format

An object of class `sicher_type` of length 2.

Enum *Enum Type Factory*

Description

Creates an enumeration type using regular function call syntax. The resulting type only accepts atomic vectors whose elements all belong to the declared set of allowed values.

Usage

```
Enum(...)
```

Arguments

`...`	Allowed scalar values or a single atomic vector of allowed values.

Value

A new sicher_type that checks all values belong to the enum.

Examples

```
status %:% Enum(1, 2, 3) %<-% 2
colors %:% Enum("red", "green", "blue") %<-% c("red", "blue")
try(colors <- c("yellow", "red"))
```

extend *Extend a structured list type with additional fields*

Description

Creates a new list type by merging the field specification of an existing `create_list_type()` type with a set of additional fields. Analogous to TypeScript interface extension (`interface Employee extends Person`).

Usage

```
extend(base, extra)
```

Arguments

base	A `sicher_type` produced by `create_list_type()`. Must carry a `sicher_spec` attribute (all types built with the patched `create_list_type()` above automatically do).
extra	A named list of additional fields in the same format accepted by `create_list_type()` – names are field names, values are `sicher_type` or `sicher_union` objects.

Value

A new `sicher_type` whose required and optional fields are the union of `base`'s fields and `extra`'s fields. Fields in `extra` that share a name with a field in `base` *override* the base field's type (with a warning so the shadowing is never silent).

Examples

```
Person <- create_list_type(list(
  name = String,
  age  = Numeric
))

# Basic extension
Employee <- extend(Person, list(
  role       = String,
  department = Optional(String)
))

emp %:% Employee %<-% list(name = "Alice", age = 30, role = "Engineer")

# Multi-level extension
Manager <- extend(Employee, list(
  reports = Numeric   # number of direct reports
))

mgr %:% Manager %<-% list(
  name = "Bob", age = 45, role = "VP", reports = 12
```

```
)

# Field override (emits a warning)
DetailedPerson <- extend(Person, list(
  age = Integer   # narrows Numeric -> Integer
))
```

`Function` *Built-in Function Type*

Description

A type that checks for function objects.

Usage

```
Function
```

Format

An object of class `sicher_type` of length 2.

`infer_type` *Infer a Type from an R Object*

Description

Infers the most appropriate sicher type constructor for a given R object. By default, inference focuses on the underlying type and does not lock in the observed length of vectors. Set ‘strict = TRUE‘ to also infer scalar and fixed-size vector constraints from the example value.

Usage

```
infer_type(obj, strict = FALSE)
```

Arguments

`obj`	Any R object (primitive, vector, list, data.frame, function, etc.)
`strict`	Logical scalar. When ‘FALSE‘ (default), infer only the base type shape, such as ‘Numeric‘, ‘String‘, ‘ListOf(Integer)‘, or a ‘create_dataframe_type()‘ schema without fixed lengths. When ‘TRUE‘, also infer ‘Scalar()‘ and ‘[n]‘ size constraints from the observed object.

Value

A sicher_type object (e.g., Numeric, String, create_list_type(...), ListOf(...), etc.)

Examples

```
infer_type(42L)               # Integer
infer_type(3.14)              # Double
infer_type(c(1, 2, 3))        # Double or Numeric, no length constraint
infer_type("abc")             # String
infer_type(c("a", "b"))       # String
infer_type(TRUE)              # Bool
infer_type(NULL)              # Null
infer_type(function(x) x + 1)  # Function
infer_type(list(a = 1, b = "x"))   # create_list_type(list(a = Double, b = String))
infer_type(list(1, 2, 3))           # ListOf(Double)
infer_type(data.frame(x = 1:3))     # create_dataframe_type(list(x = Integer))
infer_type(list(a = NULL, b = 1))  # create_list_type(list(a = Optional(Any), b = Double))

# Strict mode keeps observed length constraints
infer_type(42L, strict = TRUE)          # Scalar(Integer)
infer_type(c("a", "b"), strict = TRUE)  # String[2]
```

Integer *Built-in Integer Type*

Description

A type that checks for integer vectors.

Usage

```
Integer
```

Format

An object of class `sicher_type` of length 2.

List *Built-in List Type*

Description

A type that checks for list objects.

Usage

```
List
```

Format

An object of class `sicher_type` of length 2.

ListOf *Create a homogeneous list type*

Description

Produces a type that validates a list whose every element satisfies the provided element type. This is useful when you expect a list of similar records (e.g. parsed JSON array). You can further constrain the length with the size operator: ‘ListOf(User)[10]‘.

Usage

```
ListOf(element_type)
```

Arguments

element_type	A sicher_type or sicher_union describing each element.

Value

A sicher_type that checks the value is a list and that all elements conform to ‘element_type‘.

Examples

```
# Define an inner record type
Record <- create_list_type(list(id = Numeric, name = String))

# Now require a list of records
Records <- ListOf(Record)
records %:% Records %<-% list(
  list(id = 1, name = "a"),
  list(id = 2, name = "b")
)

# fixed-size list of ten records
TenRecs <- Records[10]
# will throw if length != 10
```

Literal *Literal Type Factory*

Description

Creates a literal type inspired by TypeScript literal unions. The resulting type only accepts scalar atomic values that exactly match one of the declared literals, including the underlying R storage mode. For example, ‘200‘ and ‘200L‘ are treated as different literals.

Usage

```
Literal(...)
```

Arguments

...	Allowed scalar atomic literal values.

Value

A new sicher_type that checks the value is exactly one of the declared literals.

Examples

```
direction %:% Literal("left", "right") %<-% "left"
direction <- "right"
direction <- "left"
try(direction <- c("right", "left"))
status_code %:% Literal(200, 404) %<-% 200
try(status_code <- 500)
```

Matches *Matches — regex-constrained string type*

Description

Creates a type that accepts only character vectors whose every element matches the given Perl-compatible regular expression. NA elements are always rejected. An empty character vector `character(0)` passes vacuously; combine with `NonEmpty()` if you need at least one element.

Usage

```
Matches(pattern)
```

Arguments

pattern	A single non-NA character string used as a PCRE regex (passed to `grepl(..., perl = TRUE)`).

Value

A new `sicher_type` that validates every element against `pattern`.

Examples

```
email %:% Matches("^[^@]+@[^@]+\\.[^@]+$") %<-% "user@example.com"
try(email <- "not-an-email")   # Error: does not match pattern

hex %:% Matches("^#[0-9A-Fa-f]{6}$") %<-% "#FF5733"

# Combine with NonEmpty to also require at least one element
tags %:% NonEmpty(Matches("^[a-z]+$")) %<-% c("foo", "bar")
```

`NonEmpty` *NonEmpty modifier — require a non-empty value*

Description

Creates a type that first validates the underlying type, then rejects empty values. For data frames "empty" means zero rows (`nrow == 0`); for all other objects it means `length == 0`.

Usage

```
NonEmpty(type)
```

Arguments

`type`	A `sicher_type` or `sicher_union` object.

Value

A new `sicher_type` that rejects zero-length (or zero-row) values after the base type check passes.

Examples

```
tags %:% NonEmpty(String) %<-% c("r", "types")
try(tags <- character(0))   # Error: value must be non-empty

items %:% NonEmpty(List) %<-% list(1, 2)
try(items <- list())        # Error: value must be non-empty
```

NonNA *NonNA modifier — reject values containing NAs*

Description

Creates a type that accepts only non-NA values of the underlying type. Any element equal to NA causes an immediate error, making silent NA propagation impossible in typed pipelines.

Usage

```
NonNA(type)
```

Arguments

type	A sicher_type or sicher_union object.

Value

A new sicher_type that first validates the underlying type, then rejects any value that contains at least one NA.

Examples

```
salary %:% NonNA(Numeric) %<-% c(1800, 2300, 4000)
try(salary <- c(1800, NA, 4000))  # Error: value contains NA(s)

# Composable with other modifiers
tag %:% NonNA(Scalar(String)) %<-% "admin"
```

Null *Built-in Null Type*

Description

A type that checks for NULL values.

Usage

```
Null
```

Format

An object of class sicher_type of length 2.

Numeric *Built-in Numeric Type*

Description

A type that checks for numeric vectors (integer or double).

Usage

```
Numeric
```

Format

An object of class `sicher_type` of length 2.

Optional *Create an optional (nullable) type variant*

Description

Creates a type that accepts NULL values in addition to the base type.

Usage

```
Optional(type)
```

Arguments

`type`	A sicher_type object

Value

A union type that includes Null

Examples

```
middle_name %:% Optional(String) %<-% NULL
middle_name <- "Marie"  # Also OK
```

print.sicher_type *Print method for sicher_type*

Description

Print method for sicher_type

Usage

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

Arguments

x	A sicher_type object
...	Additional arguments (ignored)

Value

Invisibly returns the input object

print.sicher_typed_annotation *Print method for sicher_typed_annotation*

Description

Print method for sicher_typed_annotation

Usage

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

Arguments

x	A sicher_typed_annotation object
...	Additional arguments (ignored)

Value

Invisibly returns the input object

print.sicher_typed_function *Print method for sicher_typed_function*

Description

Print method for sicher_typed_function

Usage

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

Arguments

x	A sicher_typed_function object
...	Additional arguments (ignored)

Value

Invisibly returns the input object

print.sicher_union *Print method for sicher_union*

Description

Print method for sicher_union

Usage

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

Arguments

x	A sicher_union object
...	Additional arguments (ignored)

Value

Invisibly returns the input object

Readonly *Create a readonly type variant*

Description

Creates a type that prevents reassignment after initial value is set.

Usage

```
Readonly(type)
```

Arguments

type	A sicher_type object

Value

A readonly type modifier

Examples

```
PI %:% Readonly(Double) %<-% 3.14159
try(PI <- 3.0)  # Error: cannot reassign readonly
```

Scalar *Create a scalar (length-1) type variant*

Description

Creates a type that only accepts single values (vectors of length 1).

Usage

```
Scalar(type)
```

Arguments

type	A sicher_type object

Value

A new sicher_type that checks for length 1

Examples

```
age %:% Scalar(Integer) %<-% 30L
try(age <- c(30L, 40L))  # Error: not scalar
```

String *Built-in String Type*

Description

A type that checks for character vectors.

Usage

```
String
```

Format

An object of class `sicher_type` of length 2.

typed_function *Create a type-checked function*

Description

Wraps a function with runtime type checking for its parameters and, optionally, its return value. This is the function counterpart to the typed variable operators (‘%:%‘ / ‘%<-%‘), providing a syntax analogous to typed function signatures:

```
add <- typed_function(
  function(x, y) x + y,
  params  = list(x = Numeric, y = Numeric),
  .return = Numeric
)
```

Usage

```
typed_function(fn, params = list(), .return = NULL)
```

Arguments

`fn`	The function to wrap. Its formals are preserved in the wrapper so callers use the exact same signature.
`params`	A named list mapping parameter names to their types (e.g. `list(x = Numeric, y = String)`). Only listed parameters are type-checked on each call; unlisted parameters pass through unchecked. Defaults to an empty list (no parameter checking).
`.return`	Optional return type. When `NULL` (the default), the return value is not checked. Accepts any `sicher_type` or `sicher_union`.

Value

A function with the same formals as `fn` and S3 class `"sicher_typed_function"` that:

- Validates each listed parameter on every call.
- Validates the return value when `.return` is specified.
- Delegates all argument passing to `fn` unchanged.

Examples

```
# Basic typed function
add <- typed_function(
  function(x, y) x + y,
  params  = list(x = Numeric, y = Numeric),
  .return = Numeric
)
add(1, 2)     # Returns 3
try(add("a", 2))   # Error: Type error in 'x': Expected numeric, got string

# Optional parameter
greet <- typed_function(
  function(name, title = NULL) {
    if (is.null(title)) paste("Hello,", name)
    else paste("Hello,", title, name)
  },
  params = list(name = String, title = Optional(String))
)
greet("Alice")                    # "Hello, Alice"
greet("Alice", title = "Dr.")     # "Hello, Dr. Alice"
try(greet("Alice", title = 42))  # Error: Type error in 'title'

# Union type in params
describe <- typed_function(
  function(id) paste("ID:", id),
  params  = list(id = String | Numeric),
  .return = String
)
describe("abc")  # "ID: abc"
describe(123)    # "ID: 123"
try(describe(TRUE)) # Error: Type error in 'id'
```

`union-operator` *Union Type Operator*

Description

S3 methods for the '|' operator to create union types.

Usage

```
## S3 method for class 'sicher_type'
type1 | type2

## S3 method for class 'sicher_union'
type1 | type2
```

Arguments

type1	First type (sicher_type or sicher_union object)
type2	Second type (sicher_type or sicher_union object)

Value

A union type (sicher_union object)

Index