# Valkyrie

v0.1.2

2024-05-06

GPL-3.0-only

Type safe type validation

James R SWIFT tinger <ME@TINGER.DEV>

https://github.com/JamesxX/valkyrie

This package implements type validation, and is targetted mainly at package and template developers. The desired outcome is that it becomes easier for the programmer to quickly put a package together without spending a long time on type safety, but also to make the usage of those packages by end-users less painful by generating useful error messages.

# Table of contents

I. Example usage	IV. Index
II. Documentation	
II.1. Terminology 3	
II.2. Specifig language 3	
II.3. Parsing functions 4	
II.4. Schema definition functions 6	
II.4.1. Any 6	
II.4.2. Array 7	
II.4.3. Boolean 9	
II.4.4. Choice 10	
II.4.5. Color 11	
II.4.6. Content 12	
II.4.7. Dictionary 13	
II.4.8. Logical 14	
II.4.9. Number 15	
II.4.10. String 17	
II.4.11. Tuple 19	
III. Advanced Documentation	
III.1. Internal functions 20	
III.2. Type specialization22	
III.2.1. Novice	
III.2.2. Intermediate 22	
III.2.3. Advanced 22	
III 2 4 Wizard 22	

### Part I.

# Example usage

```
(
#let template-schema = z.dictionary(
                                                             title: [],
 title: z.content(),
 abstract: z.content(default: []),
                                                             paper: "a3",
 dates: z.array(z.dictionary(
                                                             disable: (footer: true, header-journal: false),
  type: z.content(),
  date: z.string()
                                                              (name: "Example", corresponding: false, orcid: none),
                                                             ),
 paper: z.papersize(default: "a4"),
                                                             abstract: [],
 authors: z.array(z.dictionary(
                                                             dates: (),
  name: z.string(),
  corresponding: z.boolean(default: false),
                                                             header: (
  orcid: z.optional(z.string())
                                                              journal: [Journal Name],
                                                              article-type: [Article],
 header: z.dictionary(
                                                              article-color: rgb("#a7c3d4"),
  journal: z.content(default: [Journal Name]),
                                                              article-meta: [],
  article-type: z.content(default: "Article"),
  article-color: z.color(default: rgb(167,195,212)),
                                                             keywords: (),
  article-meta: z.content(default: [])
                                                             doi: none,
 keywords: z.array(z.string()),
                                                             citation: ∏,
 doi: z.optional(z.string()),
                                                             fonts: (header: "Century Gothic", body: "CMU Sans Serif"),
 citation: z.content(default: []),
 disable: z.dictionary(
  header-journal: z.boolean(default: false),
  footer: z.boolean(default: false)
 fonts: z.dictionary(
  header: z.string(default: "Century Gothic"),
  body: z.string(default: "CMU Sans Serif")
 )
);
#z.parse(
  title: [],
paper: "a3",
  disable: (footer: true),
  authors: ( (name: "Example"),)
 template-schema,
```

# Part II.

### **Documentation**

# II.1. Terminology

As this package introduces several type-like objects, the Tidy style has had these added for clarity. At present, these are schema (to represent type-validating objects), z-ctx (to represent the current state of the parsing heuristic), and scope (an array of strings that represents the parent object of values being parsed). internal represents arguments that, while settable by the end-user, should be reserved for internal or advanced usage.

Generally, users of this package will only need to be aware of the schema type.

# II.2. Specifig language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## II.3. Parsing functions

#parse()

#parse(⟨object⟩, ⟨schema⟩, ⟨ctx⟩: "z-ctx()", ⟨scope⟩: "(\"argument\",)") → any | none

This is the main function for validating an object against a schema. **WILL** return the given object after validation if successful, or none and **MAY** throw a failed assertion error.

Object to validate against provided schema. Object **SHOULD** statisfy the schema requirements. An error **MAY** be produced if not.

Argument — schema schema

Schema against which object is validated. MUST be a valid valkyrie schema type.

ctx passed to schema validator function containing flags that MAY alter behaviour

ctx passed to schema validator function, containing flags that MAY alter behaviour. See #z-ctx().

An array of strings used to generate the string representing the location of a failed requirement within object. **MUST** be an array of strings of length greater than or equal to 1.

#z-ctx()

 $\#z\text{-ctx}(\langle parent \rangle : "(:)", ..\langle args \rangle) \longrightarrow z\text{-ctx}$ 

This is a utility function for setting contextual flags that are used during validation of objects against schemas.

Currently, the following flags are described within the API:

**strict** If set, this flag adds the requirement that there are no entries in dictionary types that are not described by the validation schema.

**soft-error** If set, this flag silences errors from failed validation parses. It is used internally for types that should not error on validation failures. See #either()

Current context (if present), to which contextual flags passed in variadic arguments are appended.

# 2.3 Parsing functions

Argument ...⟨args⟩

Variadic contextual flags to set. Positional arguments are discarded.

#### II.4. Schema definition functions

#### II.4.1. Any

```
#any()
#any(
  ⟨name⟩: "any",
  ⟨default⟩: none,
  ⟨custom⟩: none,
  ⟨custom-error⟩: auto,
  ⟨transform⟩: it => it
) → schema
  This function yields a validation schema that should be satisfied by all inputs. It can be
  further specialized by providing a custom validation function and custom validation error,
  for the rapid implementation of novel types.
                                                                                         internal
  ⟨name⟩: "any"
   Used internally to generate error messages.
  ⟨default⟩: none
                                                                                      any none
   OPTIONAL default value to validate if none is provided.
                                                                                         function
  ⟨custom⟩: none
   OPTIONAL function that maps an input to an output. If the function returns none, then
   an error WILL be generated using custom-error.
  ⟨custom-error⟩: auto
                                                                                             str
   OPTIONAL error to return if custom function returns none.
 \langle transform \rangle: it => it
                                                                                         function
   OPTIONAL function that maps an input to an output, called after validation.
```

#### II.4.2. Array

```
#array()
#array(
  ⟨name⟩: "array",
  ⟨default⟩: "()",
  ⟨min⟩: none,
  \langle max \rangle: none,
   ⟨length⟩: auto,
  ⟨custom⟩: none,
  ⟨custom-error⟩: auto,
  ⟨transform⟩: it=>it,
  .. (args)
) → schema
  This function yields a validation schema that is satisfied by an array of entries than them-
  selves satisfy the schema defined in the sink argument. Array entries are validated by a
  single schema. For arrays with positional requirements, see #tuple(). If no schema for child
  entries is provided, entries are validated against #any().
  ⟨name⟩: "array"
                                                                                       internal
   Used internally to generate error messages.
                                                                                   array none
  ⟨default⟩: "()"
   OPTIONAL default value to validate if none is provided. MUST itself pass validation.
                                                                                     int none
  ⟨min⟩: none
   OPTIONAL minimum array length that satisfies the validation. MUST be a positive
   integer. The program is ILL-FORMED if min is greater than max.
                                                                                     int none
  ⟨max⟩: none
   OPTIONAL maximum array length that satisfies the validation. MUST be a positive
   integer. The program is ILL-FORMED if max is less than min.
  ⟨length⟩: auto
                                                                                     int auto
   OPTIONAL exact array length that satisfies validation. MUST be a positiive integer.
   The program MAY be ILL-FORMED is concurrently set with either min or max.
                                                                                 function none
  ⟨custom⟩: none
```

#### 2.4 Schema definition functions

OPTIONAL function that, if itself returns none, will produce the error set by customerror.

Argument
⟨custom-error⟩: auto

OPTIONAL error message produced upon failure of custom.

Argument
⟨transform⟩: it=>it

OPTIONAL mapping function called after validation.

Argument

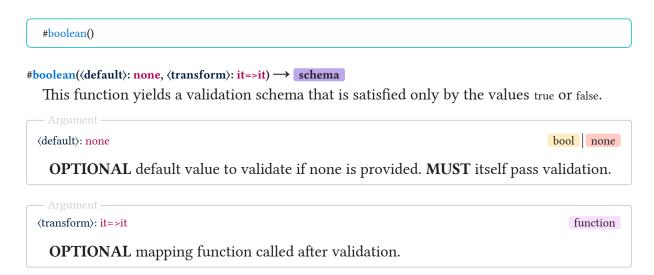
∴⟨args⟩

schema none

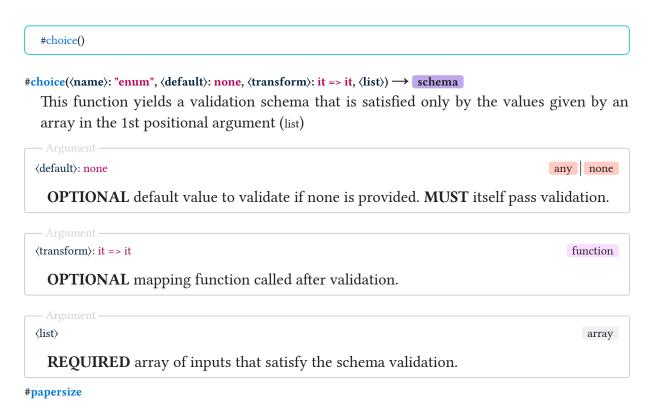
Variadic positional arguments of length 0 or 1. **SHOULD** not contain named arguments.

If no arguments are given, schema defaults to array of #any()

# II.4.3. Boolean

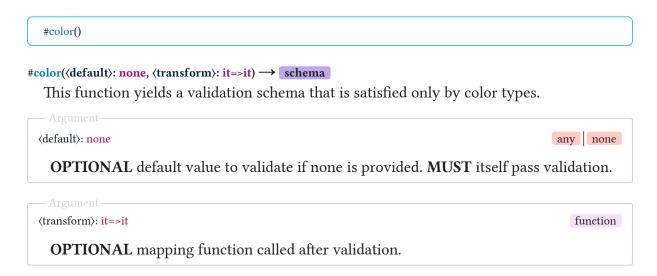


#### II.4.4. Choice



Specialization of #choice() that is satisfied only by valid paper sizes as of Typst 0.11

### II.4.5. Color



# II.4.6. Content

### II.4.7. Dictionary

```
#dictionary()
```

#### #dictionary(.. $\langle args \rangle$ ) $\rightarrow$ schema

Valkyrie schema generator for dictionary types. Named arguments define validation schema for entries. Dictionaries can be nested.

```
#let schema = z.dictionary(
    key1: z.string(),
    key2: z.number()
);
#z.parse((key1: "hello", key2: 0), schema)

(key1: "hello", key2: 0)
```

— Argument — schema schema

Variadic named arguments, the values for which are schema types. **MUST NOT** contain positional arguments. Argument name **MUST** match key name in dictionary type being validated. Argument value **MUST** be a schema type.

### II.4.8. Logical

```
#either()
```

#### $\#either(..\langle options \rangle) \longrightarrow schema$

Valkyrie schema generator for objects that can be any of multiple types.

```
#let schema = z.either(
    z.string(),
    z.number()
);
string: #z.parse("hello", schema) \
    number: #z.parse(123, schema) \
    // something else: #z.parse([content], schema)
    // -> assertion failed: Schema validation failed on argument: Type failed to match any of possible options: string or number. Got content

string: hello
number: 123
```

— Argument — schema

Variadic position arguments for possible types. **MUST** have at least 1 positional argument. Schemas **SHOULD** be given in order of "preference".

#### II.4.9. Number

```
#number()
#number(
  ⟨name⟩: "number",
  ⟨default⟩: none,
  ⟨min⟩: none,
  ⟨max⟩: none,
  ⟨custom⟩: none,
  ⟨custom-error⟩: auto,
  ⟨transform⟩: it=>it,
  ⟨types⟩: "(float, int)"
) → schema
  Valkyrie schema generator for integer- and floating-point numbers
 ⟨name⟩: "number"
                                                                                      internal
   Used internally to generate error messages.
 ⟨default⟩: none
                                                                             int | float | none
   OPTIONAL default value to set if none is provided. MUST respect all other validation
   requirements.
                                                                                   int none
 ⟨min⟩: none
   OPTIONAL minimum value that satisfies the validation. The program is ILL-
   FORMED if min is greater than max.
 ⟨max⟩: none
                                                                                   int none
   OPTIONAL maximum value that satisfies the validation. The program is ILL-
   FORMED if max is less than min.
 ⟨custom⟩: none
                                                                               function none
   OPTIONAL function that, if itself returns none, WILL produce the error set by custom-
   error.
                                                                                   str none
 ⟨custom-error⟩: auto
   OPTIONAL error produced upon failure of custom.
                                                                                      function
 ⟨transform⟩: it=>it
```

#### 2.4 Schema definition functions

#### **OPTIONAL** mapping function called after validation.

Argument (types): "(float, int)" internal

Used internally to correctly specialize.

#### #integer

Specialization of #number() that is only satisfied by whole numbers. Parameters of #number() remain available for further requirements.

#### #floating-point

Specialization of #number() that is only satisfied by floating point numbers. Parameters of #number() remain available for further requirments.

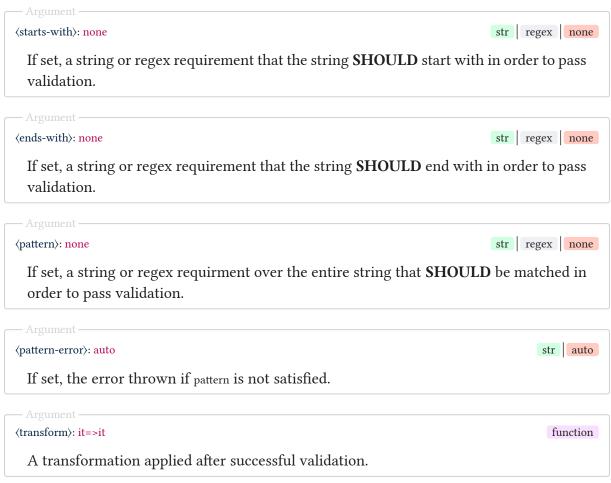
#### #natural

Specialization of #integer() that is only satisfied by positive whole numbers. Parameters of #number() remain available for further requirments.

#### **II.4.10. String**

```
#string()
#string(
  ⟨name⟩: "string",
  ⟨default⟩: none,
  ⟨min⟩: none,
  ⟨max⟩: none,
  ⟨length⟩: auto,
  ⟨includes⟩: "()",
  ⟨starts-with⟩: none,
  ⟨ends-with⟩: none,
  ⟨pattern⟩: none,
  ⟨pattern-error⟩: auto,
  ⟨transform⟩: it=>it
) → schema
  Valkyrie schema generator for strings
                                                                                         internal
  ⟨name⟩: "string"
  ⟨default⟩: none
                                                                                             str
   Default value to set if none is provided. MUST respect all other validation require-
   ments.
                                                                                      int none
  ⟨min⟩: none
   If not none, the minimum string length that satisfies the validation. MUST be a positive
   integer. The program is ILL-FORMED if min is greater than max.
                                                                                      int none
  ⟨max⟩: none
   If not none, the maximum string length that satisfies the validation. MUST be a posi-
   tive integer. The program is ILL-FORMED if max is less than min.
  ⟨length⟩: auto
                                                                                       int auto
   If not auto, the exact string length that satisfies validation. MUST be a positiive integer.
   The program MAY be ILL-FORMED is concurrently set with either min or max.
                                                                                array str none
  ⟨includes⟩: "()"
   If set, a coerced array of required strings that are required to pass validation.
```

#### 2.4 Schema definition functions



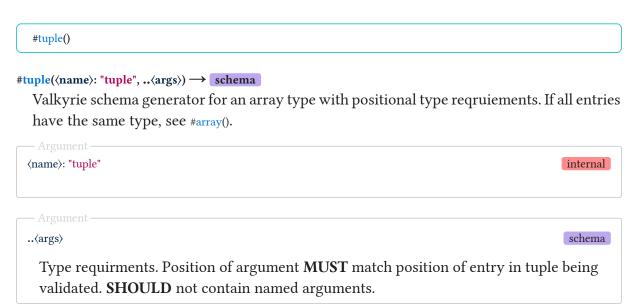
#### #email

A specialization of string that is satisfied only by email addresses. **Note**: The testing is not rigourous to save on complexity.

#ip

A specialization of string that is satisfied only by valid IP addresses. **Note**: The testing **IS** strict.

# **II.4.11.** Tuple

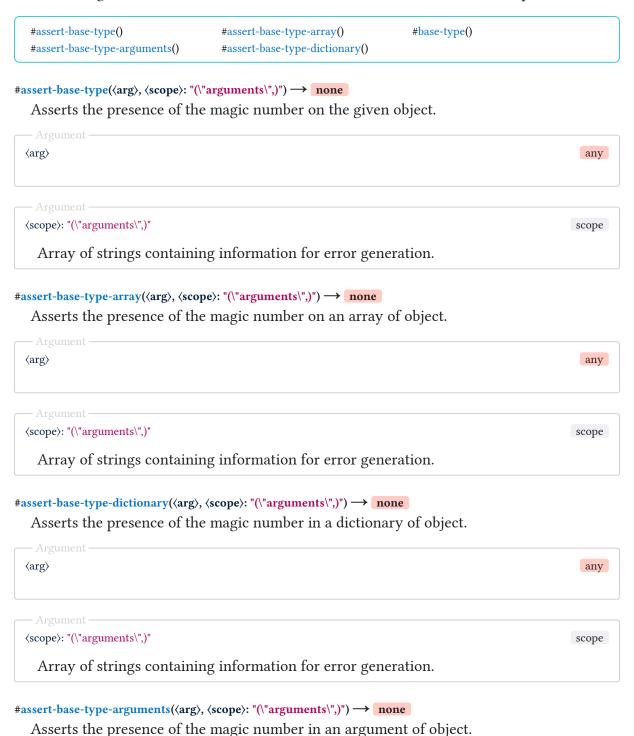


#### Part III.

### **Advanced Documentation**

#### III.1. Internal functions

The following functions are made available to users under the z.advanced namespace.



# 3.1 Internal functions



#### #base-type()

Schema generator. Provides default values for when defining custom types.

#### 3.1 Internal functions

The Typst package ecosystem is large and evergrowing. Eventually, someone, somewhere, will want to validate a type or structure that has never been seen before. If this describes your situation, the following guide may be of use. This section covers different ways complicated types can be defined.

# III.2. Type specialization

#### III.2.1. Novice

It may be the case that your type is simply a narrowing of an already-defined type. In such cases, it may be easy to add a validator for your code. For example, to create a validator for numbers between 5 and 10, you could so as as follows:

```
#let specific-number = z.number.with(min: 5, max: 10)
```

#### III.2.2. Intermediate

If the above method is not sufficient to accurately describe your type, then the custom argument (described above) may be suitable.

```
#let specific-number = z.number.with(
  custom: it => 5 < it and it < 10,
    custom-error: "Value was incorrect",
)</pre>
```

#### III.2.3. Advanced

If the above doesn't work, but would if you had access to information that would otherwise be hidden inside the schema type-like object, then bootstrapping it may be an avenue to explore.

```
#let specific-number(..args) = z.number(..args) + (
    // Configure values manually, perhaps override functions.
    // Check source code of schema generator being bootstrapped.
)
```

#### III.2.4. Wizard

For the most advanced types, creating a schema generator from scratch may be the only way (though this definitely is the last stop, this method should cover all cases). To do so, simply define a function that returns a schema-like dictionary.

```
#let tuple(my-args, ...) = {
    // Shorthand for the definitions shown below. If you do not modify a function,
    // you may as well omit it and have it set to its default by base-type()
    z.advanced.base-type() + (
    // Magic number
    valkyrie-type: true,
    // Member sometimes used by other classes when they report a failed validation
    name: "my-type",
    // Helper function, generally called by validate()
    assert-type: (self, it, scope:(), ctx: ctx(), types: ()) => {
```

#### 3.2 Type specialization

```
if type(it) not in types {
    (self.fail-validation)(
      self,
      scope: scope,
      ctx: ctx,
      message: (
      "Expected "
       + joinWithAnd(types, ", ", " or ")
       + ". Got "
       + type(it)
     ),
    )
    return false
   true
  },
  // Do your validation here. Call fail-validation() if validation failed.
  // Generally, return none also.
  validate: (self, it, scope: (), ctx: (:)) \Longrightarrow it,
  // Customize the mode of failure here
  fail-validation: (self, it, scope: (), ctx: (:), message: "") => {
   let display = "Schema validation failed on " + scope.join(".")
   if message.len() > 0 { display += ": " + message}
   ctx.outcome = display
   if not ctx.soft-error {
    assert(false, message: display)
)
```

# Part IV.

# Index

$\mathbf{A}$
#any 6
#array 7
#assert-base-type 20
#assert-base-type-arguments 20
#assert-base-type-array 20
#assert-base-type-dictionary 20
В
#base-type 21
#boolean 9
C
#choice 10
#color 11
D
#dictionary 13
E
#either 14
NT.
N
#number 15
P
#parse 4
S
#string 17
Т
#tuple 19
Z
#z-ctx 4