Package: R8 (via r-universe)

November 20, 2026

Title Structs, Enums, and Traits with a 'Rust' Runtime

Version 0.3.0

Date 2026-11-18

Description A developer-facing type system for R whose runtime is the 'rtemis-r8' 'Rust' crate, shared with other host languages. Types are structs, enums, and traits, declared in R or in 'Rust'; values are immutable and validated once per value. This version reports the version of the compiled 'rtemis-r8' crate; declarations, validation, and JSON encoding are under development.

URL https://www.rtemis.org, https://github.com/rtemis-org/r8

BugReports https://github.com/rtemis-org/r8/issues

License BSD_3_clause + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 4.3.0)

Imports methods, stats, utils

Suggests Matrix, roxygen2, testthat (>= 3.0.0)

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc >= 1.88.0, xz

Config/roxygen2/version 8.1.1

Config/pak/sysreqs xz-utils libclang-dev

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

Date/Publication 2026-11-20 08:54:52 UTC

RemoteUrl https://github.com/rtemis-org/r8

RemoteRef HEAD

RemoteSha 2c747007a9ddd344d9c4e5952351eec407e1cf1d

RemoteSubdir r

Contents

field . . . 3
generic . . . 4
hashmap . . . 5
impl . . . 5
implement . . . 6
is_variant . . . 8
json_ir . . . 8
new_enum . . . 9
new_struct . . . 10
new_trait . . . 11
Ops.r8_value . . . 12
option . . . 12
ordered_map . . . 13
print.r8_field_type . . . 13
print.r8_json_ir . . . 14
print.r8_trait . . . 14
print.r8_type . . . 15
print.r8_value . . . 15
r8_conform . . . 16
r8_convert . . . 16
r8_converter . . . 17
r8_core_version . . . 18
r8_declaration . . . 18
r8_document_native . . . 19
r8_equal . . . 20
r8_from_json . . . 20
r8_hash . . . 21
r8_method . . . 22
r8_rd . . . 22
r8_register . . . 23
r8_resolve . . . 24
r8_size . . . 25
r8_validate . . . 25
r8_value_fields . . . 26
r8_value_guards . . . 27
r8_value_text . . . 28
rule . . . 28
standard_traits . . . 29
to_json . . . 30
trait_method . . . 31
type_array . . . 32
type_character . . . 32
type_date . . . 33
type_double . . . 33
type_factor . . . 34
type_integer . . . 35

type_json 35
type_logical 36
type_matrix 36
type_posixct 37
type_r 37
type_raw 38
type_sparse_matrix 38
type_table 39
variant 40
vec 40

Index **42**

`field` *Declare a field*

Description

The facts of one field of a struct, a struct variant, or a table. A bare field type is the short form of `field(type)`.

Usage

```
field(
  type,
  default = NULL,
  description = "",
  sensitive = FALSE,
  unit = NULL,
  external = FALSE,
  unique = FALSE,
  optional = FALSE,
  extensions = NULL
)
```

Arguments

`type`	An `r8_field_type` or an R8 type: The field's type.
`default`	Optional: The default value, used when the field is absent.
`description`	Character: What the field holds.
`sensitive`	Logical: Mask the value in printing, events, errors, and differences.
`unit`	Optional Character: A UCUM unit code, such as `"mg"`.
`external`	Logical: `write_json()` writes the value to an Arrow IPC file.
`unique`	Logical: A table column whose values are distinct.
`optional`	Logical: A table column that may be absent.
`extensions`	Optional named list: Namespaced extensions, such as `list("rtemis:tunable" = TRUE)`, which R8 stores and does not interpret.

Value

An `r8_field`.

Examples

```
field(type_double(exclusive_min = 0), default = 1, description = "Cost.")
```

`generic` *Declare a generic*

Description

A generic is a function that dispatches on the R8 type of its first argument to the method that `implement()` declares for that type. A package assigns it to the generic's name, documents it with roxygen2, and exports it. The generic is a trait with one method named as the generic, `ns::describe`, which Rust and the C interface implement as any trait.

Usage

```
generic(name, args = "x", description = "")

## S3 method for class 'r8_generic'
print(x, ...)
```

Arguments

`name`	Character: The generic's name, an R8 name such as `"describe"`: a letter, then letters, digits, or underscores. It is the name the generic is assigned to.
`args`	Character: The formal argument names. The first is the value dispatched on; `...` is appended when absent.
`description`	Character: What the generic does.
`x`	A generic of class `r8_generic`.
`...`	Not used.

Value

`generic()`: The generic, a function of class `r8_generic`. `print()`: `x`, invisibly.

See Also

implement<- declares a method; `r8_method()` returns one.

Examples

```
op <- options(r8.namespace = "models")
GBM <- new_struct("GBM", fields = list(n_trees = type_integer(min = 1L)))
describe <- generic("describe", "x")
implement(describe, GBM) <- function(x, ...) {
  paste("Gradient boosting,", x$n_trees, "trees")
}
describe(GBM(n_trees = 100L))
options(op)
```

---

hashmap *Map field type*

Description

`HashMap<String, T>`: string keys, sorted at construction.

Usage

```
hashmap(type, keys = NULL)
```

Arguments

`type`	An `r8_field_type` or an R8 type: The value type.
`keys`	Optional `r8_field_type`: A `type_character()` constraining the keys.

Value

An `r8_field_type`.

Examples

```
hashmap(type_integer())
```

---

impl *Implement a trait for a type*

Description

Defines a trait's methods for a type; a method the impl omits takes the trait's default. Rust's orphan rule applies: an impl is declared in the namespace of the trait or of the type. Implementing the same trait for the same type again replaces the impl.

Usage

```
impl(trait, type, methods = list(), tag_value = NULL)
```

Arguments

trait	A trait object.
type	An R8 type object.
methods	Named list: One function per method.
tag_value	Optional Character: The value of the trait's tag member for this type; defaults to the type's name.

Value

NULL, invisibly.

Examples

```
op <- options(r8.namespace = "demo")
Describe <- new_trait("Describe", methods = list(describe = trait_method(function(x) NULL)))
Point <- new_struct("Point", fields = list(x = type_double(), y = type_double()))
impl(Describe, Point, methods = list(describe = function(x) sprintf("(%g, %g)", x$x, x$y)))
describe <- Describe@describe
describe(Point(x = 1, y = 2))
options(op)
```

---

implement *Declare the method of a generic for a type*

Description

`implement(generic, Type) <- function(x, ...)` ... declares the method that `generic` calls for values of `Type`, and returns `generic` unchanged. `generic` is written as its name; the function's arguments begin with the generic's. By the generic:

Usage

```
implement(generic, type) <- value
```

Arguments

generic	A generic, written as its name.
type	An R8 type object.
value	Function: The method.

Details

- A generic from `generic()`, or a trait method such as `Fit@fit`: an impl of its trait.
- `format`, `str`, and `dim`: an impl of the standard trait `Display`, `Debug`, or `Dim`; `r8_cmp`: of `PartialOrd`, which <, <=, >, and >= use.
- `print`: an impl of the standard trait `Print`; `print()` calls the function and returns the value invisibly, and `format()` keeps its own text. The declaration states this in a message.
- Another S3 generic of base R or of an imported package, such as `summary` or `predict`: an S3 method for the qualified class of `Type`.
- The functions through which R8 reads, changes, and compares values (`$`, `[[`, `@`, `as.list`, `$<-`, `[[<-`, `[<-`, `names<-`, `length<-`, `dim<-`, `levels<-`, `==`, `!=`), the comparison and other operators, S4 generics, and functions that are not generics raise `r8_declaration_error`. For `r8_convert` and `r8_converter`, the error names `impl(From(source), Type, ...)`, which declares a conversion.

An impl is declared in the namespace of its generic or of its type.

In a package, `r8_rd()` documents the method in the generic's Rd file.

Value

`generic`, unchanged.

See Also

`generic()`, `r8_method()`.

Examples

```
op <- options(r8.namespace = "models")
Resampler <- new_struct(
  "Resampler",
  fields = list(n_resamples = type_integer(min = 1L))
)
implement(format, Resampler) <- function(x, ...) {
  paste("Resampler,", x$n_resamples, "resamples")
}
implement(summary, Resampler) <- function(object, ...) {
  object$n_resamples
}
r <- Resampler(n_resamples = 10L)
format(r)
summary(r)
options(op)
```

is_variant *Test an enum value's variant*

Description

TRUE when x is a value of an R8 enum whose variant is variant. In a rule, is_variant(kernel, "Rbf") compiles to the same test over the value's JSON form.

Usage

```
is_variant(x, variant)
```

Arguments

x	An R8 enum value.
variant	Character: The variant name.

Value

Logical: One value.

Examples

```
op <- options(r8.namespace = "demo")
Shape <- new_enum("Shape", variants = list(Circle = variant(), Square = variant()))
is_variant(Shape@Circle, "Circle")
options(op)
```

json_ir *IR as JSON text*

Description

JSON text that a declaration carries into its IR as written: a rule's invariant or when as JSONLogic, or a field's default as the value's JSON form. It holds what the R forms do not express, such as an and of three operands, a default that is an enum variant with fields or a struct, and a number that a double does not hold exactly. r8_document_native() writes it where a Rust declaration has no R form.

Usage

```
json_ir(text)
```

Arguments

text	Character: One JSON text.

Value

An r8_json_ir: the text, of that class.

Examples

```
rule("COST_RANGE", json_ir('{"and": [{">": [{"var": "cost"}, 0]},
  {"<=": [{"var": "cost"}, 100]}, {"!==": [{"var": "name"}, ""]}]}'),
  "cost is in (0, 100] and name is not empty.")
field(type_double(), default = json_ir("0.5"))
```

---

new_enum *Declare an enum*

Description

A sum type of unit, struct, and newtype variants. An enum of unit variants is written as the variant name; an enum with another variant as an object whose tag member names the variant. Kernel@Linear is a unit variant's value.

Usage

```
new_enum(name, variants = list(), description = "", tag = "type", version = 1L)

## S3 method for class 'r8_enum_type'
@(object, name)
```

Arguments

name	The variant's name.
variants	Named list: The variants, variant() calls.
description	Character: What the type is.
tag	Character: The member that names the variant in JSON.
version	Integer: The type's version, from 1.
object	An enum type object.

Value

new_enum(): the type object, of class r8_enum_type. Kernel@Linear: the unit variant's value, an R8 value.

Examples

```
op <- options(r8.namespace = "demo")
Kernel <- new_enum("Kernel", variants = list(
  Linear = variant(),
  Rbf = variant(fields = list(gamma = type_double(exclusive_min = 0)))
))
Kernel@Linear
options(op)
```

`new_struct` *Declare a struct*

Description

A product type: named fields, each with a type and per-field facts, and rules over them. The type object constructs values: `Svm(cost = 2)`. Types declared in a package belong to its namespace (the `DESCRIPTION` field `Config/R8/namespace`, or the package name); types declared at the console belong to the option `r8.namespace`, `"global"` by default.

`Type@from(x)` converts `x` with the type's `From` impls.

Usage

```
new_struct(
  name,
  fields = list(),
  description = "",
  version = 1L,
  rules = list(),
  native = FALSE
)

## S3 method for class 'r8_struct_type'
@(object, name)
```

Arguments

`name`	The item's name.
`fields`	Named list: The fields, `field(...)` calls or bare field types.
`description`	Character: What the type is.
`version`	Integer: The type's version, from 1.
`rules`	List: `rule()` calls over the fields.
`native`	Logical: A Rust-native type, whose values the codec in the package's compiled exports builds and holds (`R8.h`). Its constructor signals `r8_codec_missing` until the codec is registered, and `r8_register()` checks that the exports carry the codec. A package whose types are declared in Rust generates these declarations with `r8_document_native()`.

object	A struct type object.

Value

The type object, of class r8_struct_type.

The item.

Examples

```
op <- options(r8.namespace = "demo")
Point <- new_struct("Point", fields = list(x = type_double(), y = type_double()),
  description = "A point.")
options(op)
```

---

new_trait *Declare a trait*

Description

A trait declares methods that types implement with impl(). A method is a generic that dispatches on the type of its first argument, reached as Trait@method; a package binds and exports it under its name, such as fit <- Fit@fit. A method named as an S3 generic of the base packages or of the package's imports, such as summary or format, becomes an S3 method of that generic, and its signature begins with the generic's arguments.

Usage

```
new_trait(name, methods = list(), description = "", tag = NULL)

## S3 method for class 'r8_trait'
@(object, name)
```

Arguments

name	Character: The trait name.
methods	Named list: The methods, trait_method() calls.
description	Character: What the trait is.
tag	Optional Character: The member that names an implementor in JSON.
object	A trait object.

Value

The trait object, of class r8_trait.

Examples

```
op <- options(r8.namespace = "demo")
Fit <- new_trait("Fit", methods = list(fit = trait_method(function(spec, x, y) NULL)))
fit <- Fit@fit
options(op)
```

`Ops.r8_value` *Comparison of R8 values*

Description

`x == y` and `x != y` compare two R8 values with `r8_equal()`, the `PartialEq` standard trait. `<`, `<=`, `>`, and `>=` use the type's `PartialOrd` impl. Other operators raise an error.

Usage

```
## S3 method for class 'r8_value'
Ops(e1, e2)
```

Arguments

`e1`, `e2`	R8 values.

Value

Logical: One value.

`option` *Optional field type*

Description

`Option<T>`: the value may be absent, `NULL`. `resolve` declares what absence means.

Usage

```
option(type, resolve = "none", compute = NULL)
```

Arguments

`type`	An `r8_field_type` or an R8 type: The type when present.
`resolve`	Character {"none", "computed", "runtime"}: `"none"`, absence is final; `"computed"`, `r8_resolve()` fills the value from `compute`; `"runtime"`, `r8_resolve()` fills it from data or the environment.
`compute`	Optional expression: For `resolve = "computed"`, the value as an expression over the other fields, captured unevaluated and written in the rule dialect.

Value

An r8_field_type.

Examples

```
option(type_integer(min = 1L), resolve = "runtime")
option(type_double(), resolve = "computed", compute = dose_mg / 1000)
```

---

ordered_map *Ordered map field type*

Description

Unique string keys in insertion order: a named list in R. Order is part of the value.

Usage

```
ordered_map(type, keys = NULL)
```

Arguments

type	An r8_field_type or an R8 type: The value type.
keys	Optional r8_field_type: A type_character() constraining the keys.

Value

An r8_field_type.

Examples

```
ordered_map(type_double())
```

---

print.r8_field_type *Print a field type*

Description

Print a field type

Usage

```
## S3 method for class 'r8_field_type'
print(x, ...)
```

Arguments

x	An r8_field_type.
...	Not used.

Value

x, invisibly.

---

print.r8_json_ir *Print IR as JSON text*

Description

Print IR as JSON text

Usage

```
## S3 method for class 'r8_json_ir'
print(x, ...)
```

Arguments

x	An r8_json_ir.
...	Not used.

Value

x, invisibly.

---

print.r8_trait *Print a trait object*

Description

Print a trait object

Usage

```
## S3 method for class 'r8_trait'
print(x, ...)
```

Arguments

x	A trait object.
...	Not used.

Value

x, invisibly.

`print.r8_type` *Print a type object*

Description

Print a type object

Usage

```
## S3 method for class 'r8_type'
print(x, ...)
```

Arguments

x	An R8 type object.
...	Not used.

Value

x, invisibly.

`print.r8_value` *Print an R8 value*

Description

Print an R8 value

Usage

```
## S3 method for class 'r8_value'
print(x, ...)
```

Arguments

x	An R8 value.
...	Not used.

Value

x, invisibly.

r8_conform *Conform an R object to a type*

Description

Turns an ordinary R object into a valid input for a struct type and constructs the value: it removes names and other attributes from scalar and vector fields, converts a character vector to a factor with declared levels, `POSIXlt` to `POSIXct`, a tibble or data.table to a data.frame, resets row names, expands an `Rle`, constructs a nested struct from a named list, and drops members that are not fields. It reports each change in one message. Construction itself stays strict: a value `r8_conform()` cannot make valid raises the same errors as the type's constructor.

Usage

```
r8_conform(x, type)
```

Arguments

`x`	Named list: The fields of a value.
`type`	An R8 struct type object.

Value

The R8 value.

Examples

```
op <- options(r8.namespace = "demo")
Weights <- new_struct("Weights", fields = list(w = vec(type_double(min = 0))))
q <- stats::quantile(c(1, 2, 3))
x <- r8_conform(list(w = q), Weights)
options(op)
```

r8_convert *Convert a value to an R8 type*

Description

Calls the `From(source)` impl of `type` for the class of `x`, or for its R8 type when `x` is an R8 value, and checks that the result is a value of `type`.

Usage

```
r8_convert(x, type)
```

Arguments

x	The value to convert.
type	An R8 type object.

Value

A value of type.

Examples

```
op <- options(r8.namespace = "demo")
Pct <- new_struct("Pct", fields = list(p = type_double(min = 0, max = 100)))
impl(From("numeric"), Pct, methods = list(from = function(x) Pct(p = x * 100)))
r8_convert(0.25, Pct)$p
options(op)
```

---

r8_converter *A conversion function for an R8 type*

Description

Returns function(x) r8_convert(x, type), which a type's package exports under a name of its choice, such as as_granges().

Usage

```
r8_converter(type)
```

Arguments

type	An R8 type object.

Value

Function.

Examples

```
op <- options(r8.namespace = "demo")
Pct <- new_struct("Pct2", fields = list(p = type_double(min = 0, max = 100)))
impl(From("numeric"), Pct, methods = list(from = function(x) Pct(p = x * 100)))
as_pct <- r8_converter(Pct)
as_pct(0.5)$p
options(op)
```

r8_core_version *Version of the rtemis-r8 crate*

Description

Returns the version of the `rtemis-r8` Rust crate compiled into R8, which provenance records alongside the versions of R8 and R.

Usage

```
r8_core_version()
```

Value

Character: The version, for example `"0.1.0"`.

Examples

```
r8_core_version()
```

r8_declaration *Declaration of a type*

Description

The IR document of a type: its namespace and its declaration, as JSON text.

Usage

```
r8_declaration(type)
```

Arguments

`type`	An R8 type object.

Value

Character: The IR document.

Examples

```
op <- options(r8.namespace = "demo")
Point <- new_struct("Point", fields = list(x = type_double(), y = type_double()))
r8_declaration(Point)
options(op)
```

r8_document_native *Generate the R declarations of types declared in Rust*

Description

Writes R/r8-types.R, the R declaration of each type that a package declares in Rust and lists in its compiled exports (R8.h): one new_struct() or new_enum() call per type, Rust-native types with native = TRUE, with roxygen comments from the Rust doc comments. The R code of the package then has the type objects, their constructors, and their declarations at install, before the shared object loads, and r8_register() checks at load that each R declaration has the declaration hash of its Rust declaration.

Usage

```
r8_document_native(path = ".", check = FALSE)
```

Arguments

path	Character: The package's root directory.
check	Logical: Compare the rendered file with R/r8-types.R and signal an error when they differ, without writing.

Details

The generator reads the IR document from the compiled shared object, src/<package><dynlib.ext>, which it loads with dyn.load() and unloads when it finishes; the package itself is not loaded, so the generator runs while its R declarations are absent or differ from the Rust ones. Before it writes, it declares the rendered file in the session and compares the declaration hash of every type with the exports' hash, so it runs in an R session of its own, without the package loaded: Rscript -e "R8::r8_document_native()", after pkgbuild::compile_dll() and before roxygen.

Value

The path of R/r8-types.R, invisibly.

Examples

```
## Not run:
pkgbuild::compile_dll()
system2("Rscript", c("-e", shQuote("R8::r8_document_native()")))
roxygen2::roxygenise()

## End(Not run)
```

r8_equal *Equality of two R8 values*

Description

TRUE when both values have the same type and equal fields: their normalized forms are the same canonical JSON, and their runtime-only fields are identical(). A provenance record does not enter the comparison.

Usage

```
r8_equal(x, y)
```

Arguments

x, y	R8 values.

Value

Logical: One value.

Examples

```
op <- options(r8.namespace = "demo")
Point <- new_struct("Point", fields = list(x = type_double(), y = type_double()))
r8_equal(Point(x = 1, y = 2), Point(x = 1L, y = 2))
options(op)
```

r8_from_json *Read an R8 value from JSON*

Description

Reads a document of type: checks "$type" and the versions in "$types", validates the value, and returns it.

Usage

```
r8_from_json(type, json)
```

Arguments

type	An R8 type object.
json	Character: The JSON document.

Value

An R8 value.

Examples

```
op <- options(r8.namespace = "demo")
Point <- new_struct("Point", fields = list(x = type_double(), y = type_double()))
r8_from_json(Point, to_json(Point(x = 1, y = 2)))
options(op)
```

`r8_hash` *Content hash of an R8 value*

Description

The SHA-256 of the RFC 8785 canonical JSON of the type's name, its declaration hash, and the value, written `sha256:<hex>`. Values that differ only in member order or in defaults written out hash the same. A matrix, an array, a sparse matrix, a table, and a `vec()` of logical, integer, double, date, or date-time values enter through the digest of their canonical binary form. A runtime-only field's value stays out of the hash; its presence enters it. The hash is computed from the R objects in place.

Usage

```
r8_hash(x)
```

Arguments

`x`	An R8 value.

Value

Character: The hash.

Examples

```
op <- options(r8.namespace = "demo")
Point <- new_struct("Point", fields = list(x = type_double(), y = type_double()))
r8_hash(Point(x = 1, y = 2))
options(op)
```

r8_method *The method of a generic for a type*

Description

Returns the function that `implement()` or `impl()` declared for `type`, for tests and inspection.

Usage

```
r8_method(generic, type)
```

Arguments

`generic`	A generic, as `implement()` takes it.
`type`	An R8 type object.

Value

Function. A generic without a method for `type` raises `r8_dispatch_error`.

Examples

```
op <- options(r8.namespace = "models")
GBM <- new_struct("GBM", fields = list(n_trees = type_integer(min = 1L)))
describe <- generic("describe", "x")
implement(describe, GBM) <- function(x, ...) "Gradient boosting"
r8_method(describe, GBM)
options(op)
```

r8_rd *Rd files for R8 declarations*

Description

A roxygen2 roclet that writes the Rd files of the `rd` roclet and adds R8 methods and enum variants to them. A package lists it in place of `rd`: `Roxygen: list(roclets = c("collate", "namespace", "R8::r8_rd"))` in its `DESCRIPTION`.

Usage

```
r8_rd()
```

Details

Each `implement(generic, Type) <- function(...)` call adds an entry to the usage section of an Rd file, `## R8 method for struct 'Type'` (or `enum`) followed by the call with the method's arguments. The entry goes to the Rd file of the generic when the package declares and documents the generic, an R8 generic or an S3 generic, and otherwise to the Rd file of the type. A roxygen block above the call adds its `@param` entries to that file; a call without a block is found by scanning `R/`, and is left out when neither the generic nor the type has an Rd file in the package. The usage of an enum lists its variants, such as `Method@KFold(k)` and `Method@Bootstrap`.

Value

A roxygen2 roclet, of class `roclet_r8_rd`.

Examples

```
## Not run:
roxygen2::roxygenise(roclets = c("collate", "namespace", "R8::r8_rd"))

## End(Not run)
```

`r8_register` *Register a package's R8 declarations*

Description

A package's top-level `new_struct()`, `new_enum()`, `new_trait()`, and `impl()` calls run when the package is installed, and record themselves in the package's namespace. `r8_register()`, called from the package's `.onLoad()`, registers them in the session: it checks each declaration again and makes its types, traits, and impls available. Registering a package again replaces its declarations.

Usage

```
r8_register(pkgname = NULL)
```

Arguments

`pkgname`	Optional Character: The package; by default, the package whose code calls `r8_register()`.

Details

A package with compiled exports (`R8.h`) registers them after its R declarations. Every type of the exports has an R declaration with the same declaration hash, the type of every codec is declared with `native = TRUE`, and every type declared with `native = TRUE` has a codec; otherwise registration signals `r8_declaration_error` and lists each difference. `r8_document_native()` generates the R declarations of types declared in Rust.

Value

The number of declarations registered, invisibly.

Examples

```
## Not run:
.onLoad <- function(libname, pkgname) {
  R8::r8_register()
}

## End(Not run)
```

`r8_resolve` *Resolve a value's pending fields*

Description

Fills the absent fields that a type declares with `option(T, resolve = )`: a `"runtime"` field from `context`, a `"computed"` field from its expression over the other fields, in dependency order. The result is a new value, validated; `x` is unchanged. A runtime field missing from `context` stays absent.

Usage

```
r8_resolve(x, context = list())
```

Arguments

`x`	An R8 struct value.
`context`	Named list: Values of runtime fields, by name.

Value

The resolved R8 value.

Examples

```
op <- options(r8.namespace = "demo")
Dose <- new_struct("Dose", fields = list(
  mg = field(option(type_double(), resolve = "runtime"), unit = "mg"),
  g = field(option(type_double(), resolve = "computed", compute = mg / 1000), unit = "g")
))
r8_resolve(Dose(), context = list(mg = 250))$g
options(op)
```

r8_size *Size of an R8 value*

Description

The memory an R8 value holds, in bytes. For a value of a Rust-native type, the bytes its codec reports for the state, plus the size of each field read so far that is not a view of the state; a view's memory is the state's. For an R-native value, `utils::object.size()`.

Usage

```
r8_size(x)
```

Arguments

x	An R8 value.

Value

Double: The size in bytes.

Examples

```
op <- options(r8.namespace = "demo")
Resampler <- new_struct("Resampler", fields = list(n = type_integer()))
r8_size(Resampler(n = 10L))
options(op)
```

r8_validate *Validate an R8 value*

Description

Checks a value against its type again, rules included: a value built by its constructor is valid, and an object reassembled from parts may not be. With `complete = TRUE`, a value whose runtime or computed fields are still absent is an error that names them.

Usage

```
r8_validate(x, complete = FALSE)
```

Arguments

x	An R8 value.
complete	Logical: Require every runtime and computed field to be present.

Value

x, invisibly.

Examples

```
op <- options(r8.namespace = "demo")
Fit <- new_struct("FitSettings", fields = list(
  mtry = option(type_integer(min = 1L), resolve = "runtime")
))
r8_validate(Fit())
try(r8_validate(Fit(), complete = TRUE))
options(op)
```

---

r8_value_fields *Fields of an R8 value*

Description

x$name and x[["name"]] read a field by its exact name; an unknown name is an error. x$name <- v and x[["name"]] <- v return a new value, validated; x is unchanged.

@ reads the associated items of a type, not the fields of a value.

Usage

```
## S3 method for class 'r8_value'
x$name

## S3 method for class 'r8_value'
x[[i, ...]]

## S3 replacement method for class 'r8_value'
x$name <- value

## S3 replacement method for class 'r8_value'
x[[i]] <- value

## S3 method for class 'r8_value'
@(object, name)
```

Arguments

x	An R8 value.
name	The name.
i	Character: The field name.
...	Not used.
value	The new field value.
object	An R8 value.

Value

The field's value, or the new R8 value.

Does not return.

`r8_value_guards` *Guards on R8 values*

Description

`names<-`, `length<-`, `[<-`, `dim<-`, and `levels<-` would change a value without validation, so they abort; `[[` with a number aborts, as fields are read by name.

Usage

```
## S3 replacement method for class 'r8_value'
names(x, ...) <- value

## S3 replacement method for class 'r8_value'
length(x, ...) <- value

## S3 replacement method for class 'r8_value'
x[...] <- value

## S3 replacement method for class 'r8_value'
dim(x, ...) <- value

## S3 replacement method for class 'r8_value'
levels(x, ...) <- value
```

Arguments

`x`	An R8 value.
`...`	Not used.
`value`	Not used.

Value

Does not return.

r8_value_text *Text of R8 values*

Description

The default `Display` and `Debug`: the type, and each field on its own line.

Usage

```
## S3 method for class 'r8_value'
format(x, ...)

## S3 method for class 'r8_value'
str(object, ...)
```

Arguments

`x`, `object`	An R8 value.
`...`	Not used.

Value

`format()`: Character, one element per line. `str()`: `NULL`, invisibly.

rule *Declare a rule*

Description

A rule states what must hold for a value. The invariant is an R expression over the type's fields, captured unevaluated: `cost <= 100`, `kernel$gamma <= 10`, `is.null(a) || !is.null(b)`. R's `==` and `!=` are strict equality. A rule holds when its `when` condition is `FALSE`, or when a field it reads is absent; `is.null()` tests presence and is not a read. Inside `all()` and `any()`, `.x` is the element. `json_ir()` gives an invariant or a condition as JSONLogic text, for JSONLogic without an R form.

Usage

```
rule(id, invariant, message, when = NULL)
```

Arguments

`id`	Character: The rule's ID, an R8 name such as `"RBF_GAMMA"`.
`invariant`	Expression: What must hold.
`message`	Character: What a violation means, for the user.
`when`	Optional expression: The condition under which the rule applies.

Value

An r8_rule.

Examples

```
rule("COST_MAX", cost <= 100, "cost is at most 100.")
rule("RBF_COST", cost <= 100, "cost is at most 100 with the Rbf kernel.",
  when = is_variant(kernel, "Rbf"))
```

---

standard_traits *Standard traits*

Description

Traits that R8 declares in its namespace R8, which a type implements with impl() like any trait:

Usage

```
Display

Debug

Dim

PartialOrd

Print

r8_cmp(x, y)

From(source)
```

Arguments

x, y	Values of one R8 type, for r8_cmp().
source	Character: The class or qualified R8 type converted from.

Details

- Display: format(x, ...), which print() shows for a type without a Print impl.
- Print: print(x, ...), which print() calls before it returns x invisibly; format() keeps the Display text.
- Debug: str(object, ...).
- Dim: dim(x); nrow() and ncol() follow.

- `PartialOrd`: `cmp(x, y)`, which returns a negative number, zero, or a positive number as `x` sorts before, with, or after `y`; `<`, `<=`, `>`, and `>=` use it, and `r8_cmp()` calls it.
- `From(source)`: `from(x)`, which converts a value of the class or R8 type `source` to the implementing type; `r8_convert()`, `Type@from()`, and `r8_converter()` use it.

R8 provides `format()`, `print()`, and `str()` for every value, `==` and `r8_hash()` through `r8_equal()`, `to_json()` and `r8_from_json()`, and a type whose required fields have defaults is constructed with no arguments.

Value

A trait object.

Examples

```
op <- options(r8.namespace = "demo")
Money <- new_struct("Money", fields = list(cents = type_integer()))
impl(Display, Money, methods = list(format = function(x, ...) sprintf("$%.2f", x$cents / 100)))
impl(PartialOrd, Money, methods = list(cmp = function(x, y) x$cents - y$cents))
impl(From("numeric"), Money, methods = list(
  from = function(x) Money(cents = as.integer(round(x * 100)))
))
m <- r8_convert(1.5, Money)
print(m)
m < Money(cents = 200L)
options(op)
```

---

to_json *Write an R8 value as JSON*

Description

The value's document: `"$type"`, `"$types"` (the version of every type it contains), then its fields in declaration order.

Usage

```
to_json(x)
```

Arguments

`x`	An R8 value.

Value

Character: The JSON document.

Examples

```
op <- options(r8.namespace = "demo")
Point <- new_struct("Point", fields = list(x = type_double(), y = type_double()))
to_json(Point(x = 1, y = 2))
options(op)
```

`trait_method` *Declare a trait method*

Description

Declare a trait method

Usage

```
trait_method(fn, description = "", default = NULL, stochastic = FALSE)
```

Arguments

`fn`	Function: Its formals are the method's signature; the first formal is the value the method dispatches on.
`description`	Character: What the method does.
`default`	Optional function: A default implementation, used for a type whose impl does not define the method.
`stochastic`	Logical: The method draws random numbers.

Value

An `r8_trait_method`.

Examples

```
trait_method(function(spec, x, y) NULL, description = "Trains a model.")
```

type_array *Array field type*

Description

A dense array of a scalar type, an atomic vector with `dim` in R.

Usage

```
type_array(type, dim = NULL)
```

Arguments

`type`	An `r8_field_type` of a scalar type, or `option()` of one.
`dim`	Optional Integer: The dimensions.

Value

An `r8_field_type`.

Examples

```
type_array(type_integer(), dim = c(2L, 3L, 4L))
```

type_character *Character field type*

Description

A string without U+0000, never `NA`.

Usage

```
type_character(pattern = NULL, values = NULL)
```

Arguments

`pattern`	Optional Character: A regular expression in Rust `regex` syntax that the string matches.
`values`	Optional Character: The admitted values.

Value

An `r8_field_type`.

Examples

```
type_character(values = c("gaussian", "binomial"))
```

`type_date` *Date field type*

Description

A calendar date from 0000-01-01 to 9999-12-31.

Usage

```
type_date()
```

Value

An `r8_field_type`.

Examples

```
type_date()
```

`type_double` *Double field type*

Description

A double. The domain is finite unless `finite = FALSE`, which admits `Inf`, `-Inf`, and `NaN`.

Usage

```
type_double(
  min = NULL,
  max = NULL,
  exclusive_min = NULL,
  exclusive_max = NULL,
  finite = TRUE,
  values = NULL
)
```

Arguments

`min`	Optional Numeric: Smallest admitted value.
`max`	Optional Numeric: Largest admitted value.
`exclusive_min`	Optional Numeric: The value must be above it.
`exclusive_max`	Optional Numeric: The value must be below it.
`finite`	Logical: Restrict the domain to finite values.
`values`	Optional Numeric: The admitted values.

Value

An `r8_field_type`.

Examples

```
type_double(exclusive_min = 0)
```

`type_factor` *Factor field type*

Description

A vector of categorical values.

Usage

```
type_factor(levels = NULL, ordered = FALSE)
```

Arguments

`levels`	Optional Character: The declared levels. Without them, the levels come from the data.
`ordered`	Logical: An ordered factor.

Value

An `r8_field_type`.

Examples

```
type_factor(levels = c("+", "-", "*"))
```

type_integer *Integer field type*

Description

An integer from -2147483647 to 2147483647. A double given for an integer field converts when it is a whole number in range.

Usage

```
type_integer(
  min = NULL,
  max = NULL,
  exclusive_min = NULL,
  exclusive_max = NULL,
  values = NULL
)
```

Arguments

min	Optional Integer: Smallest admitted value.
max	Optional Integer: Largest admitted value.
exclusive_min	Optional Integer: The value must be above it.
exclusive_max	Optional Integer: The value must be below it.
values	Optional Integer: The admitted values.

Value

An r8_field_type.

Examples

```
type_integer(min = 1L)
```

type_json *JSON field type*

Description

Any JSON value: objects as named lists, arrays as unnamed lists, numbers as doubles.

Usage

```
type_json()
```

Value

An r8_field_type.

Examples

```
type_json()
```

---

type_logical *Logical field type*

Description

A logical value, TRUE or FALSE, never NA.

Usage

```
type_logical()
```

Value

An r8_field_type.

Examples

```
type_logical()
```

---

type_matrix *Matrix field type*

Description

A dense matrix of a scalar type, an atomic vector with dim in R. Given an R8 struct type, a matrix whose columns are the struct's fields, all of one scalar type.

Usage

```
type_matrix(type, nrow = NULL, ncol = NULL)
```

Arguments

type	An r8_field_type of a scalar type (or option() of one, for missing cells), or an R8 struct type.
nrow	Optional Integer: The number of rows.
ncol	Optional Integer: The number of columns.

Value

An r8_field_type.

Examples

```
type_matrix(option(type_double()), nrow = 2L)
```

---

type_posixct *Date-time field type*

Description

A UTC instant in microseconds, from 1827-04-16T00:06:12.629505Z to 2112-09-17T23:53:47.370495Z, the range a double holds exactly.

Usage

```
type_posixct()
```

Value

An r8_field_type.

Examples

```
type_posixct()
```

---

type_r *Runtime-only R field type*

Description

An R value of a class, held in R only: absent from JSON and from the hash. A field of this type is declared inside option().

Usage

```
type_r(class)
```

Arguments

class	Character: The R class.

Value

An r8_field_type.

Examples

```
option(type_r("lm"), resolve = "runtime")
```

`type_raw` *Raw field type*

Description

Bytes, a raw vector in R and canonical padded base64 in JSON.

Usage

```
type_raw()
```

Value

An r8_field_type.

Examples

```
type_raw()
```

`type_sparse_matrix` *Sparse matrix field type*

Description

A matrix in compressed sparse column form, a dgCMatrix (package Matrix) in R.

Usage

```
type_sparse_matrix(type = type_double(), nrow = NULL, ncol = NULL)
```

Arguments

type	An r8_field_type: type_double() or type_logical().
nrow	Optional Integer: The number of rows.
ncol	Optional Integer: The number of columns.

Value

An r8_field_type.

Examples

```
type_sparse_matrix()
```

---

type_table *Table field type*

Description

A data.frame of equal-length columns, each a scalar type, a factor, or option() of one, with automatic row names. An open table also accepts columns its declaration does not name. A table's rules run once per row and read the row's columns by name. Given an R8 struct type, a table whose columns are the struct's fields.

Usage

```
type_table(columns = list(), open = FALSE, rules = list())
```

Arguments

columns	Named list of fields or field types, or an R8 struct type.
open	Logical: Accept columns the declaration does not name.
rules	List: rule() calls over a row's columns.

Value

An r8_field_type.

Examples

```
type_table(
  list(start = type_integer(), width = type_integer(min = 0L)),
 rules = list(rule("END_FITS", start + width - 1 <= 2147483647, "The end fits in an integer."))
)
```

variant *Declare an enum variant*

Description

A unit variant has no fields; a struct variant has named fields; a newtype variant holds one value of a struct, `variant(of = GRanges)`.

Usage

```
variant(fields = list(), description = "", rules = list(), of = NULL)
```

Arguments

`fields`	Named list: The fields of a struct variant.
`description`	Character: What the variant is.
`rules`	List: `rule()` calls over the variant's fields.
`of`	Optional R8 struct type: The struct a newtype variant holds.

Value

An `r8_variant`.

Examples

```
variant(fields = list(
  gamma = field(type_double(exclusive_min = 0), description = "Kernel width.")
))
```

vec *Vector field type*

Description

`Vec<T>`: an atomic vector for a scalar `T`, a list otherwise.

Usage

```
vec(type, min_length = NULL, max_length = NULL, unique = FALSE)
```

Arguments

`type`	An `r8_field_type` or an R8 type: The element type.
`min_length`	Optional Integer: Fewest elements.
`max_length`	Optional Integer: Most elements.
`unique`	Logical: Elements are distinct.

Value

An r8_field_type.

Examples

```
vec(type_double(min = 0), min_length = 1L)
```

Index

[<-.r8_value *(r8_value_guards)*, 27
[[.r8_value *(r8_value_fields)*, 26
[[<-.r8_value *(r8_value_fields)*, 26
$.r8_value *(r8_value_fields)*, 26
$<-.r8_value *(r8_value_fields)*, 26

Debug *(standard_traits)*, 29
Dim *(standard_traits)*, 29
dim<-.r8_value *(r8_value_guards)*, 27
Display *(standard_traits)*, 29

field, 3
format.r8_value *(r8_value_text)*, 28
From *(standard_traits)*, 29

generic, 4
generic(), *7*

hashmap, 5

impl, 5
implement, 6
implement<-, *4*
implement<- *(implement)*, 6
is_variant, 8

json_ir, 8

length<-.r8_value *(r8_value_guards)*, 27
levels<-.r8_value *(r8_value_guards)*, 27

names<-.r8_value *(r8_value_guards)*, 27
new_enum, 9
new_struct, 10
new_trait, 11

Ops.r8_value, 12
option, 12
ordered_map, 13

PartialOrd *(standard_traits)*, 29
Print *(standard_traits)*, 29
print.r8_field_type, 13
print.r8_generic *(generic)*, 4
print.r8_json_ir, 14
print.r8_trait, 14
print.r8_type, 15
print.r8_value, 15

r8_cmp *(standard_traits)*, 29
r8_conform, 16
r8_convert, 16
r8_converter, 17
r8_core_version, 18
r8_declaration, 18
r8_document_native, 19
r8_equal, 20
r8_from_json, 20
r8_hash, 21
r8_method, 22
r8_method(), *4*, *7*
r8_rd, 22
r8_register, 23
r8_resolve, 24
r8_size, 25
r8_validate, 25
r8_value_fields, 26
r8_value_guards, 27
r8_value_text, 28
rule, 28

standard_traits, 29
str.r8_value *(r8_value_text)*, 28

to_json, 30
trait_method, 31
type_array, 32
type_character, 32
type_date, 33
type_double, 33
type_factor, 34

type_integer, 35
type_json, 35
type_logical, 36
type_matrix, 36
type_posixct, 37
type_r, 37
type_raw, 38
type_sparse_matrix, 38
type_table, 39

variant, 40
vec, 40