Package: erratum (via r-universe)

September 1, 2024

Title Handle Error and Warning Messages

Version 2.2.0.9000

Description Elegantly handle error and warning messages.

License AGPL-3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.0

Imports R6, rlang

Suggests testthat, covr, shiny

Config/testthat/edition 3

BugReports https://github.com/devOpifex/erratum/issues/

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

RemoteUrl https://github.com/devOpifex/erratum

RemoteRef HEAD

RemoteSha b7648f8a59deb7002f9f9d6f920a9aefe2e00b40

Contents

---

bash *Take a Bash*

Description

Equivalent to `tryCatch()`.

Usage

```
bash(expr, e = NULL, w = NULL)
```

Arguments

expr	Expression to run, passed to `tryCatch()`.
e, w	An object of class `Error` or `Warning` as returned by `e()` or `w()`.

Examples

```
safe_log <- function(x){
 result <- bash(log(x))

 if(is.e(result))
   stop(result$stop())

 return(result)
}

if(interactive())
 safe_log("a")
```

checks *Check*

Description

Check whether an object is an error or a warning.

Usage

```
is.e(obj)

## Default S3 method:
is.e(obj)

## S3 method for class 'err'
is.e(obj)

is.w(obj)

## Default S3 method:
is.w(obj)

## S3 method for class 'err'
is.w(obj)

is.problem(obj)
```

Arguments

obj	Object to check.

Value

A boolean value.

Functions

- `is.e`: Whether the object is an error.
- `is.w`: Whether the object is a warning.
- `is.problem`: Whether the object is an error or a warning.

Examples

```
err <- e("Whoops!")

is.e(err)
is.w(err)
```

chk *Check*

Description

Checks individual objects.

Usage

```
chk(obj)

## Default S3 method:
chk(obj)

## S3 method for class 'err'
chk(obj)
```

Arguments

obj	Object to check.

Details

Runs `warning()` or `stop()` where necessary.

e.observe *Observe with Error Handling*

Description

Observe with erratum error handling.

Usage

```
e.observe(
  x,
  e = NULL,
  w = NULL,
  env = parent.frame(),
  ...,
  label = NULL,
  suspended = FALSE,
  priority = 0,
  domain = shiny::getDefaultReactiveDomain(),
  autoDestroy = TRUE,
  ..stacktraceon = TRUE
)
```

Arguments

x	An expression (quoted or unquoted). Any return value will be ignored.
e	Error handler.
w	Warning handler.
env	The parent environment for the reactive expression. By default, this is the calling environment, the same as when defining an ordinary non-reactive expression. If x is a quosure and quoted is TRUE, then env is ignored.
...	Not used.
label	A label for the observer, useful for debugging.
suspended	If TRUE, start the observer in a suspended state. If FALSE (the default), start in a non-suspended state.
priority	An integer or numeric that controls the priority with which this observer should be executed. A higher value means higher priority: an observer with a higher priority value will execute before all observers with lower priority values. Positive, negative, and zero values are allowed.
domain	See domains.
autoDestroy	If TRUE (the default), the observer will be automatically destroyed when its domain (if any) ends.
..stacktraceon	Advanced use only. For stack manipulation purposes; see stacktrace().

---

e.observe.event *Observe Event with Error Handling*

Description

Observe event with erratum error handling.

Usage

```
e.observe.event(
  eventExpr,
  handlerExpr,
  e = NULL,
  w = NULL,
  event.env = parent.frame(),
  event.quoted = FALSE,
  handler.env = parent.frame(),
  handler.quoted = FALSE,
  ...,
  label = NULL,
  suspended = FALSE,
  priority = 0,
  domain = shiny::getDefaultReactiveDomain(),
```

```
  autoDestroy = TRUE,
  ignoreNULL = TRUE,
  ignoreInit = FALSE,
  once = FALSE
)
```

Arguments

eventExpr	A (quoted or unquoted) expression that represents the event; this can be a simple reactive value like `input$click`, a call to a reactive expression like `dataset()`, or even a complex expression inside curly braces
handlerExpr	The expression to call whenever `eventExpr` is invalidated. This should be a side-effect-producing action (the return value will be ignored). It will be executed within an `isolate()` scope.
e	Error handler.
w	Warning handler.
event.env	The parent environment for the reactive expression. By default, this is the calling environment, the same as when defining an ordinary non-reactive expression. If `eventExpr` is a quosure and `event.quoted` is `TRUE`, then `event.env` is ignored.
event.quoted	If it is `TRUE`, then the `quote()`ed value of `eventExpr` will be used when `eventExpr` is evaluated. If `eventExpr` is a quosure and you would like to use its expression as a value for `eventExpr`, then you must set `event.quoted` to `TRUE`.
handler.env	The parent environment for the reactive expression. By default, this is the calling environment, the same as when defining an ordinary non-reactive expression. If `handlerExpr` is a quosure and `handler.quoted` is `TRUE`, then `handler.env` is ignored.
handler.quoted	If it is `TRUE`, then the `quote()`ed value of `handlerExpr` will be used when `handlerExpr` is evaluated. If `handlerExpr` is a quosure and you would like to use its expression as a value for `handlerExpr`, then you must set `handler.quoted` to `TRUE`.
...	Currently not used.
label	A label for the observer or reactive, useful for debugging.
suspended	If `TRUE`, start the observer in a suspended state. If `FALSE` (the default), start in a non-suspended state.
priority	An integer or numeric that controls the priority with which this observer should be executed. An observer with a given priority level will always execute sooner than all observers with a lower priority level. Positive, negative, and zero values are allowed.
domain	See domains.
autoDestroy	If `TRUE` (the default), the observer will be automatically destroyed when its domain (if any) ends.
ignoreNULL	Whether the action should be triggered (or value calculated, in the case of `eventReactive`) when the input event expression is `NULL`. See Details.

ignoreInit	If TRUE, then, when this observeEvent is first created/initialized, ignore the handlerExpr (the second argument), whether it is otherwise supposed to run or not. The default is FALSE. See Details.
once	Whether this observeEvent should be immediately destroyed after the first time that the code in handlerExpr is run. This pattern is useful when you want to subscribe to a event that should only happen once.

Error *Error*

Description

Error

Error

Super class

erratum::Issue -> Error

Methods

Public methods:

- Error$new()
- Error$stop()
- Error$fatal()
- Error$clone()

Method new():

Usage:

```
Error$new(obj, raiser = getOption("ERR_RAISER_ERROR", stopper))
```

Arguments:

obj A character string or an object of class error, or warning.

raiser Template to raise the issue.

Details: Initialise

Method stop():

Usage:

```
Error$stop()
```

Details: Stop
Analogous to stop()

Method fatal():

Usage:

```
Error$fatal()
```

Details: Fatal
Analogous to `stop()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Error$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ew` *Handlers*

Description

Handle errors and warnings.

Usage

```
e(obj)

w(obj)
```

Arguments

`obj`	A character string or an object of class `error`, or `warning`.

Examples

```
err <- e("Something went wrong")

foo <- function(x){
 if(is.character(x))
   return(err)

 log(x)
}

foo("a")
```

get_call *Extract Call*

Description

Extract call from error and warnings.

Usage

```
get_call(obj)
```

Arguments

obj	Message string, object of class error, or warning.

get_msg *Extract Message*

Description

Extract message from error and warnings.

Usage

```
get_msg(obj)
```

Arguments

obj	Message string, object of class error, or warning.

Issue *Core Class*

Description

Core class to create and handle issues.

Active bindings

`rule` Rules to perform checks, must be functions that accept a single argument and return a boolean.

`message` The message (warning or error).

`call` Expression or function (as string) that led to the issue.

`raiser` Function to run when the `raise` method is called. By default the error uses `stop()` and warning uses `warning()`. The function must accept a single argument: the error message (character vector).

Methods

Public methods:

- `Issue$new()`
- `Issue$print()`
- `Issue$return()`
- `Issue$addRule()`
- `Issue$check()`
- `Issue$raise()`
- `Issue$clone()`

Method `new()`:

Usage:

```
Issue$new(obj, type = c("error", "warning"))
```

Arguments:

`obj` A character string or an object of class `error`, or `warning`.

`type` Type of message.

Details: Initialise

Method `print()`:

Usage:

```
Issue$print()
```

Details: Print
Print message of error or warning.

Method `return()`:

Usage:

```
Issue$return(n = 1)
```

Arguments:

`n` the number of generations to go back, passed to `parent.frame()`.

Details: Return Returns self from parent function.

Method `addRule()`:

Usage:

```
Issue$addRule(fn)
```

Arguments:

`fn` Function defining rule, must accept a single argument and return a boolean.

Details: Add a rule

Method `check()`:

Usage:

```
Issue$check(obj)
```

Arguments:

`obj` Object to check by rules

Details: Add a predicate

Method `raise()`**:**

Usage:

```
Issue$raise(fn = NULL)
```

Arguments:

`fn` A function to use to raise the issue.

Details: Raise error or warning

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Issue$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`latch` *Latch an Error or a Warning*

Description

Latch an error or a warning to an object to indicate an issue associated with it. These can later be checked with `is.e()` and `is.w()`, and can also be `resolve()`.

Usage

```
latch.e(obj, error)

latch.w(obj, warning)

unlatch(obj)
```

Arguments

`obj`	Object to latch the error or warning onto.
`error`, `warning`	Error or warning, the output of `e()` or `w()`.

Functions

- `latche` and `latchw`: latch an error or a warning.
- `unlatch`: unlatch any error or warning.

Examples

```
x <- 1
problematic <- latch.e(x, e("Not right"))

is.e(problematic)

do_sth_with_x <- function(x){
 resolve(x)
 x + 1
}

if(interactive()){
 do_sth_with_x(x)
 do_sth_with_x(problematic)
}

unlatch(problematic)
```

---

raise *Raisers*

Description

Set `raise` method globally, every subsequent `raise` method will make use of this function.

Usage

```
raise.e(fn = NULL)

raise.w(fn = NULL)
```

Arguments

`fn`	Function to run when the `raise` method is called. By default the error uses `stop()` and warning uses `warning()`. The function must accept a single argument: the error message (character vector).

---

resolves *Resolve Errors and Warnings*

Description

Resolve Errors and Warnings

Usage

```
resolve(...)

defer_resolve(...)
```

Arguments

`...`	Objects to check, if any of them is an `Error` then `stop()` is called, if any are `Warnings` then `warning()` is called.

Details

Objects passed are evalutated in order.

Value

Invisiby returns `NULL`

`skip` *Skip*

Description

Skip the rest of the function; calls `return()` in the parent function if any object is an error or (optionally) a warning.

Usage

```
skip(..., w = FALSE)
```

Arguments

`...`	Objects to check, if any of them is an `Error` then it calls `return()` in the parent function, this can optionally be applied if any object is a `Warning` with the `w` argument.
`w`	Whether to also skip is there are `Warning`.

template *Templates*

Description

Define error and warning templates.

Usage

```
template.e(pat = "%s")

template.w(pat = "%s")
```

Arguments

pat	Pattern to use, must include %s, forwarded to sprintf().

Examples

```
msg <- "Something's wrong"

# default
e(msg)

# template
template.e("Whoops: %s - sorry!")
e(msg)

# reset
template.e()
```

Warning *Error*

Description

Error

Error

Super class

erratum::Issue -> Warning

Methods

Public methods:

- `Warning$new()`
- `Warning$warn()`
- `Warning$clone()`

Method `new()`:

Usage:

```
Warning$new(obj, raiser = getOption("ERR_RAISER_WARNING", warner))
```

Arguments:

`obj` A character string or an object of class `error`, or `warning`.

`raiser` Template to raise the issue.

Details: Initialise

Method `warn()`:

Usage:

```
Warning$warn()
```

Details: Warn
Analogous to `warning()`

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Warning$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index