Package 'needy'

August 29, 2016

Type Package

Title needy

Description needy is a small utility library designed to make testing function inputs less difficult. R is a dynamically typed language, but larger projects need input checking for scalabity. needy offers a single function, require_a( ), which lets you specify the traits an input object should have, such as class, size, numerical properties or number of parameters, while reducing boilerplate code and aiding debugging.

Version 0.2

Depends R (>= 2.14.0)

Date 2013-06-05

Author Ryan Grannell

Maintainer Ryan Grannell <r.grannell2@gmail.com>

Contact Ryan Grannell <r.grannell2@gmail.com>

BugReports <r.grannell2@gmail.com>

Suggests testthat, combinat

License MIT + file LICENSE

Collate 'report.R' 'require_a.R' 'tools.R' 'trait_tests.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2013-07-31 21:17:45

R topics documented:

require_a *Ensure a value has a desired set of traits.*

Description

Ensure a value has a desired set of traits.

Usage

```
require_a(traits, value, pcall = NULL)

implemented_traits()

add_trait(name, trait_test)
```

Arguments

traits	a character vector, with each element being a space-seperated string of properties to test the value for. See "traits". required.
value	an arbitrary R object to test for certain properties. required.
pcall	an call or string that provides the call to be displayed when an error is thrown by require_a. See details. optional, defaults to displaying the call to require_a().
name	a string giving the name of the test to add. required.
trait_test	a unary function that returns a true or false value. This function should tests for a particular trait.required.

Details

the option `pcall` is included so that it is possible to customise where the errors seem to originate from. for example,

```
myfunc <- function (x) require_a("integer", x,  sys.call( sys.parent(1) ))
```

will display the following if called with a string "a":

```
Error: myfunc("a"): the value "a" didn\'t match any  of the following compound traits: integer
```

In this example, the user-facing function `myfun` is shown to throw the error rather than an obscure inner function, making debugging easier. For cases in which working with the call stack directly (`sys.call()`) is too difficult a string can be passed to `pcall`, and this string is printed in front of the error message

Traits

The `traits` parameter is a character vector of whitespace-seperated traits. For example, the following are syntactically valid

```
"integer"
```

```
"positive integer"
```

```
c("positive integer", "na")
```

```
c("na", "null", "length_one pairlist")
```

while the following are not

```
"positive && integer" # just use whitespace to   'and' traits
```

```
"positive || integer" # use two elements to 'or'   traits
```

The latter two examples, correctly implemented, would be:

```
"positive integer"
```

```
c("positive", "integer")
```

As suggested above, whitespace between traits is interpreted as "trait a AND trait b", while seperate elements are intepreted as `c("trait one", OR "trait two")` the order of traits in a compound trait is not significant; a `"positive integer"` is equivelant to `"integer positive"`.

If a test corresponding to an atomic trait is not found, an error is thrown:

```
require_a("white-whale", 1)
```

```
Error: require_a("white-whale", 1): unrecognised   trait(s): (white-whale)
```

Similarily, if a value doesn't have any other desired compound traits then an error is thrown:

```
require_a(c("length_one list", "null"), 1)
```

```
Error: require_a(c("length_one list", "null"), 1):   the value 1 didn't match any of the following com
traits: length_one and list, or null'
```

As of version 0.2 trait negation is also supported:

```
require_a("!null", NULL)
```

```
Error: require_a("!null", NULL): the value NULL   didn't match any of the following compound traits:
!null'
```

Examples

```
safeMap <- function (f, x) {
# map, with verification by require_a

pcall <- "safeMap(f, x)"
require_a('unary function', f, pcall)
require_a('listy', x)

Map(f, x)
}

safeSum <- function (a, b) {

pcall <- sys.call()
require_a("finite numeric", a, pcall)
require_a("finite numeric", b, pcall)

a + b
}
```

```
definitelyNotNull <- function (x) {

pcall <- sys.call()
require_a("!null", x, pcall)

x
}

safeMatchFun <- function (f) {
# match.fun with arg checking

pcall <- sys.call()
require_a(c("string", "function", "symbol"), f, pcall)
require_a("functionable") #my prefered shorthand for the above

match.fun(f)
}
```

Index