Package 'futile.any'

July 2, 2014

Type Package

Title Futile library to provide some polymorphic operations

Version 1.3.0

Date 2013-02-02

Author Brian Lee Yung Rowe

Maintainer Brian Lee Yung Rowe <r@zatonovo.com>

Depends lambda.r (>= 1.1.0)

Suggests testthat

Description A collection of utility functions that provide polymorphism over certain data types

License LGPL-3

LazyLoad yes

Repository CRAN

Date/Publication 2013-02-03 08:39:52

NeedsCompilation no

R topics documented:

`futile.any-package` *Futile library to provide some polymorphism*

Description

This package contains a collection of utility functions that provide polymorphism over certain data types. These any* functions attempt to consolidate attribute access of lists, vectors, matrices, arrays, and other data structures.

Details

Package:	futile.any
Type:	Package
Version:	1.3.0
Date:	2013-02-02
License:	LGPL-3
LazyLoad:	yes

The anylength and anynames functions consolidate attribute access across many data structures providing a bit of convenience via polymorphism. The anytypes function provides the classes or types of a data.frame-like object. This is useful when parsing data and it is not always clear how values will be parsed.

Author(s)

Brian Lee Yung Rowe <r@zatonovo.com>

See Also

`anylength`, `anynames`, `anytypes`

Examples

```
m <- matrix(c(1,2,3,4,5,6), ncol=2)
anylength(m)

v <- c(1,2,3,4,5)
anylength(v)

m <- matrix(c(1,2,3,4,5,6), ncol=2)
anynames(m) <- c('d','e')
anynames(m)

v <- c(a=1,b=2,c=3,d=4,e=5)
anynames(v)
```

```
l <- list(a=1,b=2,c=3,d=4,e=5)
anynames(l)
```

`anylength` *Get the generic length of an object*

Description

This function consolidates size dimensions for one and two dimensional data structures. The idea is that many operations require knowing either how long a vector is or how many rows are in a matrix. So rather than switching between length and nrow, anylength provides the appropriate polymorphism to return the proper value.

Usage

```
anylength(...)
```

Arguments

...	Abstract function controlled by lambda.r

Details

anylength(data) data - An object

When working with libraries, it is easy to forget the return type of a function, particularly when there are a lot of switches between vectors, matrices, and other data structures. This function along with its `anynames` counterpart provides a single interface for accessing this information across objects.

The core assumption is that in most cases length is semantically synonomous with nrow such that the number of columns in two-dimensional structures is less consequential than the number of rows. This is particularly true of time-based objects, such as zoo or xts where the number of observations is equal to the number of rows in the structure.

Value

The length or nrow of the object

Author(s)

Brian Lee Yung Rowe

See Also

`anynames`

Examples

```
m <- matrix(c(1,2,3,4,5,6), ncol=2)
anylength(m)

v <- c(1,2,3,4,5)
anylength(v)
```

---

anynames *Get the useful names of a data structure*

Description

This function consolidataes data sets within lists and two dimensional data. Like anylength the idea is to unify the accessors for various data structures with a single common interface.

Usage

```
anynames(...)
```

Arguments

...	Abstract function controlled by futile.paradigm

Details

Depending on the type of structure utilized in code, one needs to call either names or colnames to get information related to the data sets within the structure. The use of two separate functions can cause errors and slows development time as data structures passed from intermediate functions may change over time, resulting in a broken interface.

By providing a thin layer over underlying accessors, this function attempts to expedite development and add a bit of polymorphism to the semantics of names. The explicit assumption is that data sets in two dimensional structures are organized by column, as this is compatible with time-series objects such as zoo and xts.

Value

The names or colnames of an object.

Author(s)

Brian Lee Yung Rowe

See Also

anylength

Examples

```
m <- matrix(c(1,2,3,4,5,6), ncol=2)
anynames(m) <- c('a','b')
anynames(m)

v <- c(a=1,b=2,c=3,d=4,e=5)
anynames(v)

l <- list(a=1,b=2,c=3,d=4,e=5)
anynames(l)
```

`anytypes` *Get the useful types of a data structure*

Description

This function consolidataes data sets within lists and two dimensional data. Like `anylength` the idea is to unify the accessors for various data structures with a single common interface.

Usage

```
anytypes(...)
```

Arguments

`...`	Abstract function controlled by lambda.r

Details

anytypes(data, fun = class) data - An object fun - The function to use to get the types. Defaults to class, although type or mode, etc. could be used

Depending on the type of structure utilized in code, one needs to call either names or colnames to get information related to the data sets within the structure. The use of two separate functions can cause errors and slows development time as data structures passed from intermediate functions may change over time, resulting in a broken interface.

By providing a thin layer over underlying accessors, this function attempts to expedite development and add a bit of polymorphism to the semantics of names. The explicit assumption is that data sets in two dimensional structures are organized by column, as this is compatible with time-series objects such as zoo and xts.

Value

The types or classes of a data structure

Author(s)

Brian Lee Yung Rowe

See Also

anynames

Examples

```
d <- data.frame(ints=c(1,2,3), chars=c('a','b','c'), nums=c(.1,.2,.3))
anytypes(d)
```

---

is.bad *Check whether data is bad or empty*

Description

These functions quickly test whether data within an object has bad values or if the object is defined (i.e. not null) but has no data.

Usage

```
is.bad(...)

is.empty(...)
```

Arguments

...	Abstract function controlled by lambda.r

Details

is.empty(x) %::% a : logical

is.bad(x) %::% list : logical

is.bad(x) %::% data.frame : matrix

is.bad(x) %::% matrix : matrix

is.bad(x) %::% a : logical

x - The data to test

Depending on the type of an object, knowing whether an object contains a valid value or not is different. These functions unify the interfaces across different data types quickly indicating whether an object contains bad values and also whether an object has a value set.

For example, a data.frame may be initialized with no data. This results in an object that is non-null but also unusable. Instead of checking whether something is both non-null and has positive length, just check is.bad().

If you know that an object is non-null, then you can call is.empty() which is a shortcut for checking the length of an object.

Value

Logical values that indicate whether the test was successful or not. For matrices and data.frames, a matrix of logical values will be returned.

Author(s)

Brian Lee Yung Rowe

Examples

```
a <- data.frame(a=NULL, b=NULL)
is.bad(a)

b <- list(a=1:3, b=NULL, c=NA, d='foo')
is.bad(b)

c <- list()
is.empty(c)
```

Index