Package 'objectProperties'

July 22, 2025

Title A Factory of Self-Describing Properties

Description Supports the definition of sets of properties on objects. Observers can listen to changes on individual properties or the set as a whole. The properties are meant to be fully self-describing. In support of this, there is a framework for defining enumerated types, as well as other bounded types, as S4 classes.

Maintainer Michael Lawrence <michafla@gene.com>

Depends R (>= 2.12), objectSignals (>= 0.10.2)

Imports methods

Version 0.6.8

License GPL (>= 2)

Collate PropertySet-class.R Enum-class.R comp-classes.R

NeedsCompilation no

Author Tengfei Yin [aut],
Michael Lawrence [aut, cre]

Repository CRAN

Date/Publication 2022-05-02 23:02:03 UTC

Contents

as.list-methods 2
comp 2
Enum-class 4
properties 6
PropertySet-class 7
setIntegerWithRange 9
setNumericWithRange 10

Index **12**

as.list-methods *Coercion to* list

Description

Coercion from PropertySet to list.

Arguments

x	A PropertySet object.

Details

This coersion only return a list of properties instances. filtering out singal function and other fields which are not properties.

Value

A list of properties instance.

Author(s)

Tengfei Yin

Examples

```
filt.gen <- setRefClass("Filter", properties(list(cutoff = "NonnegativeInteger",
weight = "PositiveInteger")),
contains = "PropertySet")
obj <- filt.gen$new(cutoff = NonnegativeInteger(0),
weight = PositiveInteger(1))
obj$properties()
as.list(obj)
```

comp *Bounded types for properties*

Description

This set of classes define different numerical object with restriction on it.

Usage

```
PositiveInteger(object)
```

Arguments

object	object to be coerced

Details

These special classes could be registered as signaling fields by calling `signalingFields` or `signalingField`, or using `setProperties`, so they could be used for GUI design, and changing of the fields automatically validate the current value

The construction of these objects has validation with them, please see the example.

PositiveInteger(object) Construct a `PositiveInteger` object

NonpositiveInteger(object) Construct a `NonpositiveInteger` object

NegativeInteger(object) Construct a `NegativeInteger` object

NonnegativeInteger(object) Construct a `NonnegativeInteger` object

Value

An object of the corresponding class

Author(s)

Tengfei Yin, Michael Lawrence

Examples

```
## Constructors
require(objectProperties)
obj <- PositiveInteger(1)
obj <- NonnegativeInteger(0)
obj <- NegativeInteger(-1)
obj <- NonpositiveInteger(0)
## setting as properties
filt.gen <- setRefClass("Filter",
properties(list(cutoff = "NonnegativeInteger",
weight = "PositiveInteger")), contains = "PropertySet")
## new property instance
obj <- filt.gen$new(cutoff = 0, weight = 1)
obj$properties()
as.list(obj)
## get the value
obj$cutoff
## set the value
obj$cutoff <- 30
## the class doesn't change
## if you pass a value which out of boundary, it will throw an error message
obj$cutoff
class(obj$cutoff)
```

Enum-class *Enumerated types*

Description

R functions often have parameters with enumerated values. These are typically passed as a character vector and resolved using match.arg(). The Enum structure is very similar to that of a factor, except the data is character, not integer and with appropriate validation.

Usage

```
setSingleEnum(prefix, levels, contains=character(),
              where=topenv(parent.frame()))

## S4 method for signature 'Enum'
levels(x)
```

Arguments

prefix	Prefix for new subclass of SingleEnum or MultipleEnum, e.g. if prefix is "Geom", the new subclass name would be GeomSingleEnum after calling setSingleEnum.
levels	An vector of characters which define the levels for this class.
contains	What class does this class extended besides SingleEnum.
where	the environment in which to store or remove the definition. Defaults to the top-level environment of the calling function.
x	A Enum object.

Details

The SingleEnum object is different from simple factor. It validates the value to see if it's in the defined levels during construction. and only the value within defined levels is allowed to be set as current chosen value when it is created as property. It is particularly useful for GUI design, such as creating a drop list or ratio buttons for exclusive choice, you can only choose one item within certain choices at one time. setSingleEnum will create a S4 subclass for SingleEnum, and return the class name.

The MultipleEnum has the same design with SingleEnum, except it support multiple choices. So for GUI level, it could be used for creating check boxes. setMultipleEnum will create a S4 subclass for MultipleEnum, and return the class name.

The Enum class is a Class union for SingleEnum and MultipleEnum

Color class is a special character, this properties could be used for creating a widgets which showing a color picker pallete and a text input field, a simple character object will be only treated as simple text in the UI. Color class could be constructed by constructor Color.

ColorEnum class is a VIRTUAL class, which including a set of SingleEnum subclass, when creating widget based on this property, it should be treated as a special color droplist, instead of showing a

droplist of levels of text, it shows a drop list of colors, the levels are treated as color in this class. `setColorEnum` is a convenient class generator function for single enum of `ColorEnum` and it return a class name.

`GlyphEnum` class is a VIRTUAL class, which including a set of `SingleEnum` subclass, when creating widget based on this property, it should be treated as a special glyph droplist, instead of showing a droplist of levels of text, it shows a drop list of different glyphs, the levels are treated as glyphs in this class. Different engine genenerate icons for different glyphs, such as different point size, line type, etc. `setGlyphEnum` is a convenient class generator function for single enum of `GlyphEnum` and it return a class name.

Value

`setSingleEnum` return a class name for `SingleEnum` subclass.`setMultipleEnum` return a class name for `MultipleEnum` subclass. `setColorEnum` return a class name for `ColorEnum` subclass which is also a `SingleEnum`. `setGlyphEnum` return a class name for `GlyphEnum` subclass which is also a `SingleEnum`. All those function return a generator function in R(>= 2.15)

Author(s)

Tengfei Yin, Michael Lawrence

Examples

```
## ------------------------------------------------------------------------
##                   setSingleEnum
## ------------------------------------------------------------------------
ShapeEnum.gen <- setSingleEnum("Shape",
                               levels = c("circle", "line", "rectangle"))

obj <- new("ShapeSingleEnum", "circle")
obj
obj <- "triangle" # doesn't check, because it's not signal field.
obj # it's not SingleEnum object anymore, be careful.
class(obj) # just character

## only set it as properties, allow you to assign the value and
## validate it.
par.gen <- setRefClass("Graph",
                       properties(fields = list(shape = "ShapeSingleEnum"),
                                  prototype = list(shape = new("ShapeSingleEnum",
                                                               "circle"))))
pars <- par.gen$new()
pars$shape
pars$shape <- "line"
pars$shape
class(pars$shape)# still a SingleEnum
## ------------------------------------------------------------------------
##                   setMultipleEnum
## ------------------------------------------------------------------------
ShapeEnum.gen <- setMultipleEnum("Shape",
                               levels = c("circle", "line", "rectangle"))
```

```
par.gen <- setRefClass("Graph",
                       properties(list(shape = "ShapeMultipleEnum")))
## we can initialize in this way too
pars <- par.gen$new(shape = new("ShapeMultipleEnum", c("circle", "line")))
pars$shape
pars$shape <- c("line", "rectangle")
pars$shape
class(pars$shape)# still a MultipleEnum

## Color Single Enum
bgColorSingleEnum.gen <- setColorEnum("bgColor", levels = c("black", "white", "gray"))
obj <- new("bgColorSingleEnum", "white")
## Glyph Single Enum
PointSizeSingleEnum.gen <- setGlyphEnum("PointSize",
  levels = c("1", "2", "5", "10"), contains = "GlyphEnum")
obj <- new("PointSizeSingleEnum", "1")
obj

## ------------------------------------------------------------------------
##                   change levels
## ------------------------------------------------------------------------

geomSingleEnum <- setSingleEnum("geom", c("rect", "triangle"))
obj <- geomSingleEnum("rect")

## change levels
levels(obj)
levels(obj) <- c("rect", "circle")

## changed levels must include current value
try(levels(obj) <- c("triangle", "circle"))

## ------------------------------------------------------------------------
##                   change levels
## ------------------------------------------------------------------------
obj <- factor("a", levels = letters)
SingleEnum(obj)
MultipleEnum(obj)
```

`properties` *Properties signaling fileds*

Description

Convenience function for defining a set of reference class fields that signals when set.

Usage

```
properties(fields=list(), prototype=list())
```

Arguments

fields	list of names of the field and associated fields class
prototype	A list of values declaring a default value for a field.

Details

When constructing signaling fields in this way, each field has the ability to register its own signal and at the same time, there is one top level signal which could be emitted no matter which field changes. Please see the example to learn to register global signal and individual signal.

Value

A list that is easily concatenated into the field list

Author(s)

Michael Lawrence, Tengfei Yin

Examples

```
## we could pass prototype as in S4
GPars.gen <- setRefClass("GraphicProperties",
                fields = properties(fields = list(size = "numeric",
                                                  color = "character"),
                                    prototype = list(size =1,
                                                     color = "red")))

obj <- GPars.gen$new()
## since it's not PropertySet, no global signal
## let's register individual signal
obj$sizeChanged$connect(function(){
  print("size changed")
})
## emit signal
obj$size <- 3
## no signal
obj$color <- "black"
```

PropertySet-class *PropertySet-class*

Description

The PropertySet class is a collection of properties and is useful as a data model, e.g., for storing the parameters of some operation.

setPropertySet is a simple wrapper around setRefClass for creating subclasses of PropertySet. It ensures that all fields of the subclass are defined via properties.

Usage

```
setPropertySet(Class, fields=list(), prototype=list(), contains="PropertySet", ...,
    where=topenv(parent.frame()))
```

Arguments

Class	class name
fields	list of fields
prototype	list of default values, as in setClass.
contains	superclasses, one of which must extend PropertySet
...	additional arguments to setRefClass
where	the environment in which to define the class

Details

PropertySet-class: PropertySet object has following methods, where x is a PropertySet object:

x$properties() Return the classes of the properties as a named character vector. Compare to the fields method on a reference class generator.

as.list(x) Returns a named list of the property values.

When any property in the set changes, the changed(name) signal is emitted, where name is the name of the property that changed.

Value

setPropertySet: the class generator object

Author(s)

Michael Lawrence, Tengfei Yin

Examples

```
filt.gen <- setRefClass("Filter", properties(fields = list(cutoff = "numeric",
                                                           weight = "numeric"),
                                  prototype = list(cutoff = 0, weight = 1)),
                                  contains = "PropertySet")
obj <- filt.gen$new()
obj
obj$properties()
as.list(obj)
obj$changed$connect(function(name) print(name))
obj$cutoffChanged$connect(function() print(paste("change to", obj$cutoff)))
obj$cutoff <- 0
obj$cutoff <- 2
obj$weight <- 3
```

```
## use setPropertySet, the same thing as above
filt.gen <- setPropertySet("Filter", fields = list(cutoff = "numeric",
                                        weight = "numeric"),
                           prototype = list(cutoff = 0, weight = 1))

obj <- filt.gen$new()
obj
obj$properties()
as.list(obj)
obj$changed$connect(function(name) print(name))
obj$cutoffChanged$connect(function() print(paste("change to", obj$cutoff)))
obj$cutoff <- 0
obj$cutoff <- 2
obj$weight <- 3
```

`setIntegerWithRange` *Define a speicific range object*

Description

This class creator is used to define a special property for numeric range, which could be used for UI design and could be setted as signaling field, so it will support validation on the input.

Usage

```
setIntegerWithRange(prefix = "Integer", min, max, where=topenv(parent.frame()))
```

Arguments

`prefix`	Prefix for new class name. Default is "Integer"
`min`	Minimal value for this range object.
`max`	Maximal value for this range object.
`where`	the environment in which to store or remove the definition. Defaults to the top-level environment of the calling function.

Details

The purpose of creating such a class genenrator is to define a special range properties which could be set as singaling field, such as `Properties` object. Then validation will be turned on automatically to make sure the current value is within the defined range. This is particular useful when you try to design a slider widget of such a property, let's say, a alpha blending slider.

Value

A S4 class name in R(< 2.15) and a generator function in R(>= 2.15)

Author(s)

Tengfei Yin

Examples

```
num1to100.gen <- setIntegerWithRange(min = 1L, max = 100L)
par.gen <- setRefClass("Graph",
                       properties(list(size = "IntegerWithMin1Max100")))
pars <- par.gen$new(size = new("IntegerWithMin1Max100", 5.5))
## Covert to integer
pars$size #current value is 5
try(pars$size <- 300) # out of range error
pars$size <- 4.4 # covert to integer
pars$size
```

---

setNumericWithRange *Define a speicific range object*

Description

This class creator is used to define a special property for numeric range, which could be used for UI design and could be setted as signaling field, so it will support validation on the input.

Usage

```
setNumericWithRange(prefix = "Numeric", min, max, where=topenv(parent.frame()))
```

Arguments

prefix	Prefix for new class name.Default is "Numeric"
min	Minimal value for this range object.
max	Maximal value for this range object.
where	the environment in which to store or remove the definition. Defaults to the top-level environment of the calling function.

Details

The purpose of creating such a class genenrator is to define a special range properties which could be set as singaling field, such as `Properties` object. Then validation will be turned on automatically to make sure the current value is within the defined range. This is particular useful when you try to design a slider widget of such a property, let's say, a alpha blending slider.

Value

A S4 class name in R(< 2.15) and a generator function in R(>= 2.15)

Author(s)

Tengfei Yin

Examples

```
num1to100.gen <- setNumericWithRange(min = 1, max = 100)
par.gen <- setRefClass("Graph",
                       properties(list(size = "NumericWithMin1Max100")))
pars <- par.gen$new(size = new("NumericWithMin1Max100", 5))
pars$size #current value is 5
try(pars$size <- 300) # out of range error
pars$size <- 10 #works

## Positive Integer
par.gen <- setRefClass("PI", properties(list(size  = "PositiveInteger"),
                                        list(size = PositiveInteger(2))))
obj <- par.gen$new()
## error
try(obj$size <- -1)
obj$size <- 3
```

Index

a reference class generator, *8*
as.list *(as.list-methods)*, 2
as.list,-method *(as.list-methods)*, 2
as.list,PropertySet-method *(as.list-methods)*, 2
as.list-methods, 2

Color *(Enum-class)*, 4
Color-class *(Enum-class)*, 4
ColorEnum-class *(Enum-class)*, 4
comp, 2

Enum-class, 4

GlyphEnum-class *(Enum-class)*, 4

IntegerWithRange-class *(setIntegerWithRange)*, 9

levels *(Enum-class)*, 4
levels,Enum-method *(Enum-class)*, 4
levels<- *(Enum-class)*, 4
levels<-,Enum-method *(Enum-class)*, 4

MultipleEnum *(Enum-class)*, 4
MultipleEnum-class *(Enum-class)*, 4

NegativeInteger *(comp)*, 2
NegativeInteger-class *(comp)*, 2
NonnegativeInteger *(comp)*, 2
NonnegativeInteger-class *(comp)*, 2
NonpositiveInteger *(comp)*, 2
NonpositiveInteger-class *(comp)*, 2
NumericWithMin0Max1-class *(setNumericWithRange)*, 10
NumericWithRange-class *(setNumericWithRange)*, 10

PositiveInteger *(comp)*, 2
PositiveInteger-class *(comp)*, 2
properties, 6, *7*
PropertySet, *7*
PropertySet-class, 7

setClass, *8*
setColorEnum *(Enum-class)*, 4
setGlyphEnum *(Enum-class)*, 4
setIntegerWithRange, 9
setMultipleEnum *(Enum-class)*, 4
setNumericWithRange, 10
setPropertySet *(PropertySet-class)*, 7
setRefClass, *7*
setSingleEnum *(Enum-class)*, 4
show,PropertySet-method *(as.list-methods)*, 2
SingleEnum *(Enum-class)*, 4
SingleEnum-class *(Enum-class)*, 4