Package ‘qtbase’

May 9, 2018

Version 1.0.14
Title Interface Between R and 'Qt'
Author Michael Lawrence, Deepayan Sarkar
Depends R (>= 2.10.0), methods, utils
URL http://qtinterfaces.r-forge.r-project.org
Maintainer Michael Lawrence <michafla@gene.com>
Description Dynamic bindings to the 'Qt' library for calling 'Qt'
methods and extending 'Qt' classes from R. Other packages build upon 'qtbase'
to provide special-purpose high-level interfaces to specific parts of 'Qt'.
License GPL (>= 2)
SystemRequirements Qt4 libraries and headers (http://qt.nokia.com),
cmake (http://www.cmake.org)
Acknowledgements QtRuby and Qyoto projects for head-start on the
Smoke-based interface
OS_type unix
NeedsCompilation yes
Repository CRAN
Date/Publication 2018-05-09 09:03:20 UTC

R topics documented:

as.QImage .......................................................... 2
constructors ....................................................... 3
DataFrameModel ................................................... 5
dim-methods ....................................................... 6
qconnect ........................................................... 7
qfindChild ......................................................... 8
qinvoke ............................................................ 8
qlibrary ........................................................... 9
qmethods .......................................................... 10
qmocMethods ..................................................... 11
Description

Coercion methods for converting R objects to QImage.

Usage

```r
as.QImage(x, ...)
```

Arguments

- `x` Object to coerce. By default, a `raster` object, or `matrix` as returned by `col2rgb` (or something coercible to one).
- `...` Arguments to pass to methods

Value

A QImage object

Author(s)

Michael Lawrence
## Constructors

<table>
<thead>
<tr>
<th>Convenience constructors</th>
</tr>
</thead>
</table>

### Description

These functions construct some simple Qt helper objects. They offer convenience beyond that of the native constructor, largely because they might coerce from an R equivalent to a Qt object. Also, unlike C++, argument skipping is supported.

### Usage

- `qrect(x0, y0, x1, y1)`
- `qpoint(x, y)`
- `qsize(width, height)`
- `qpolygon(x = NULL, y = NULL)`
- `qfont(family = baseFont$family(), pointsize = baseFont$pointSize(), weight = baseFont$weight(), italic = baseFont$style() == Qt$QFont$StyleItalic, baseFont = Qt$QApplication$font())`  
- `qpen(brush = qbrush(), width = 0L, style = Qt$Qt$SolidLine, cap = Qt$Qt$SquareCap, join = Qt$Qt$BevelJoin)`
- `qbrush(color = qcolor(), style = Qt$Qt$SolidPattern)`
- `qcolor(red = 0, green = 0, blue = 0, alpha = 255)`
- `qtransform(m11 = 1.0, m12 = 0.0, m13 = 0.0, m21 = 0.0, m22 = 1.0, m23 = 0.0, m31 = 0.0, m32 = 0.0, m33 = 1.0)`

### Arguments

- **x0**  
  Top-left X coordinate. If a vector of length two, taken as `x0` and `x1`. If a 2x2 matrix, the columns represent the X and Y positions. If missing, a "null" rectangle is returned.

- **y0**  
  Top-left Y coordinate. If a vector of length two, taken as `y0` and `y1`.

- **x1**  
  Bottom-right X coordinate.

- **y1**  
  Bottom-right Y coordinate.

- **x**  
  X coordinate. If a vector of length two, taken as `x` and `y`. For `qpolygon`, passed to `xy.coords`.

- **y**  
  Y coordinate. For `qpolygon`, passed to `xy.coords`.

- **width**  
  Width for the size. If a vector of length two, taken as `width` and `height`. If missing, a "null" size is returned. For `qpen`, width of the line.

- **height**  
  Height for the size.

- **family**  
  Font family name.

- **pointsize**  
  Font point size.

- **weight**  
  Font weight (bold).

- **italic**  
  Whether font is italicized.
baseFont: The base font to use for the defaults. By default, this is the global application font.

brush: Something coercible to a QBrush object via qbrush(brush).

style: Line dashing for qpen, pattern type for qbrush.

cap: Line cap style.

join: Line join style.

color: Something coercible to a QColor via qcolor(color).

red: Red component, 0-255. If a string, taken as color name and converted. If a matrix, taken as output of col2rgb.

green: Green component, 0-255.

blue: Blue component, 0-255.

alpha: Alpha component, 0-255.

m11: Horizontal scaling factor

m12: Vertical shearing factor

m13: Horizontal projection factor

m21: Horizontal shearing factor

m22: Vertical scaling factor

m23: Vertical projection factor

m31: Horizontal translation factor

m32: Vertical translation factor

m33: Division factor

Details

For structures with a F variant, like QSizeF, the variant is chosen based on the type of the input. If the values are double, a QSizeF instance is returned, otherwise an instance of QSize.

Value

An instance, RQtObject

Author(s)

Michael Lawrence, Deepayan Sarkar

Examples

```
## notice the coercion chaining:
redColor <- qcolor("red")
redBrush <- qbrush("red")
redPen <- qpen("red")
blackPen <- qpen()

## Qt's constructor uses width/height, not x1/y1
```
DataFramesModel

## Description

The `qdataFrameModel` function creates a DataFrameModel, an implementation of `QAbstractItemModel` using a `data.frame`. This makes it easy and fast to display and edit a `data.frame` in a `QTableView` or any other derivative of `QAbstractItemView`. The `qdataFrame` and `qdataFrameModel` functions allow one to get and set the `data.frame` underlying the model after construction.

## Usage

```r
qdataFrameModel(df, parent=NULL, useRoles=FALSE, editable=character(), ...) qdataFrame(model) <- value qdataFrame(model)
```

## Arguments

- **df**: The `data.frame` that provides the data of the model
- **...**: Extra arguments passed to `qdataFrameModel`, which actually loads the `data.frame` into the model.
- **model**: DataFrameModel instance
- **parent**: The parent QObject
- **useRoles**: Whether to interpret column names as indicating alternative roles; see details.
- **editable**: Character vector of column names in the `data.frame` that should be editable
- **value**: A `data.frame` that provides the data of the model

## Details

`qdataFrameModel`: While a simple `data.frame` can be displayed as a textual table, fancier tables require multiple data columns mapped to a single model column, each playing a separate 'role'. To specify additional roles, pass `useRoles = TRUE`. A role may be any string; those used by Qt are listed in the `Qt::ItemDataRole` enumeration. The display and edit roles are reserved (see below). See the documentation of the QStyledItemDelegate class for its expected data types for each role.

A simple way to encode this is in the column name, syntax: `.headerName1[.headerName2][.etc].role`. Examples:
dim-methods

- `.carColor.background` (background color for `carColor` column)
- `.foreground` (foreground color for all columns)
- `.firstName.lastName.font` (special font for first and last name columns)

The set of model columns is derived from the unique header names. Display-role columns are those not prefixed by a period. If the column name in the data matches a string in the `editable` argument, the data is used for both the edit and display roles.

Value

`qDataFrameModel`: An instance of C++ `DataFrameModel`

Note

Calling the `headerData` method on `DataFrameModel` from R will not yield the expected result, because Smoke does not know of `DataFrameModel` and thus misses the override of the non-pure virtual. We can special case this if need-be.

Author(s)

Michael Lawrence

---

**dim-methods**

Get the dimensions of rectangles and rectangular objects...

Description

Get the dimensions of rectangles and rectangular objects

Usage

```r
## S3 method for class 'QRectF'
dim(x)

## S3 method for class 'QGraphicsScene'
dim(x)

## S3 method for class 'QGraphicsItem'
dim(x)

## S3 method for class 'QGraphicsView'
dim(x)
```

Arguments

- `x` A rectangular object, like `QRect` or `QGraphicsItem`
qconnect

Connect a signal handler

Description

Like many other GUI toolkits, Qt supports connecting handler functions to signals, which are emitted in response to user actions. The qconnect function connects an R function to a signal on a Qt object.

Usage

qconnect(x, signal, handler, user.data)

Arguments

- x: The Qt object providing the signal
- signal: The name of the signal
- handler: The R function to handle the signal
- user.data: Data to pass to the R function as the last parameter. If omitted, no such argument is passed to the handler.

Details

In Qt, only other QObject instances can listen to a signal, so this function creates a dummy object that invokes the R function upon receipt of the signal.

To disconnect a handler, call the disconnect method on the signal emitter x, passing it the returned dummy QObject; see the example.

Value

A dummy QObject instance that is listening to the signal. Will become invalid when the signal emitter is invalidated.

Author(s)

Michael Lawrence

Examples

window <- Qt$QWidget()
button <- Qt$QPushButton(window)
listener <- qconnect(button, "pressed", function(x) print("hello world"))
button$disconnect(listener)
qfindChild  
*Find child by name*

**Description**

Finds a child QObject by name. Mirrors a function in Qt that is inaccessible via the ordinary interface because it is a template.

**Usage**

```cpp
qfindChild(x, name)
```

**Arguments**

- `x`  
The parent QObject
- `name`  
The name of the child

**Details**

This is particularly useful when working with QtDesigner, where objects in the UI file are named.

**Value**

The child QObject

**Author(s)**

Michael Lawrence

**Examples**

```cpp
parent <- Qt$QObject()
child <- Qt$QObject(parent)
child$objectName <- "foo"
qfindChild(parent, "foo")
```

---

qinvoke  
*Invoke a method*

**Description**

These functions invoke a method on an object or class. Usually, one does not call these functions directly but uses the $ short-cut, instead.
Usage

qinvoke(x, method, ...)  
qinvokeStatic(x, method, ...)

Arguments

x The object or class with the method  
method The name of the method  
... Arguments to pass to the method

Details

Perhaps the only reason to use one of these functions directly is in the context of functional iteration, e.g., when calling the same method on every object in a list with `lapply`.

Value

The return value of the method

Author(s)

Michael Lawrence

Examples

widgets <- replicate(length(letters), Qt$QWidget())  
mapply(qinvoke, widgets, "setWindowTitle", letters)

---

qlibrary

*Populate a library object*

Description

This function is only for use by packages that provide a library to R. It should be called in the `.onLoad` function of the package.

Usage

qlibrary(lib, namespace = deparse(substitute(lib)), register = TRUE)
Arguments

lib
An environment object that exists in the namespace of the package. The name of the object in the calling frame must match the name of the module in Smoke, unless the name attribute is set.

namespace
If not NULL, the implicit top-level namespace of the library. Many libraries have a top-level namespace, which usually should be ignored (implied) for convenience. Note that this is not the case for Qt, even though it does place many enumerations inside the Qt namespace.

register
Whether to register the library in the global list.

Value

The library object, but catching the return value is useless, since, as an environment, it was modified by reference.

Author(s)

Michael Lawrence

See Also

The vignette, once it exists, for creating packages based on qtbase

Examples

```r
## regenerate the Qt library object
Qt <- new.env()
qlibrary(Qt, NULL)
```

---

qmethods  Introspect methods and properties

Description

There are two types of exposed class members: methods and, for QObject derivatives, properties. These functions return data.frame objects describing methods and properties.

Usage

```r
qmethods(x)
qproperties(x)
```

Arguments

x
Object of class RQtClass for qmethods, a QObject instance for qproperties
qmocMethods

Value

For qmethods: a data.frame with columns for method name, return type name, signature string, and whether the method is static and/or protected.

For qproperties: a data.frame with columns for type name, and whether the property is readable and/or writeble. The property names are stored in the row names.

Author(s)

Michael Lawrence

Examples

qmocMethods(x)
qsignals(x)
qslots(x)

qmocMethods

List MOC methods

Description

Access information describing the Qt Meta Object Compiler (MOC) methods defined for a class. These are typically signals and slots, although arbitrary methods/constructors are also supported.

Usage

qmocMethods(x)
qsignals(x)
qslots(x)

Arguments

x

A QObject object or derived class (as an RQtClass).

Value

For qmocMethods, a data.frame with columns for the method name, type ("signal", "slot", etc), signature, return type, and nargs (argument count).

qsignals returns a similar data.frame containing only signal methods and the columns name and signature.

qslots returns a similar data.frame containing only slot methods and the columns name, signature and return.

Author(s)

Michael Lawrence
qrTextFormattingDelegate

See Also

qsetSignal, qsetSlot for defining new MOC methods on user classes

Examples

qmocMethods(Qt$QWidget)
widget <- Qt$QWidget()
qsignals(widget)

qrTextFormattingDelegate

R-style Text Formatting Delegate

Description

Constructs an object of class RTextFormattingDelegate, which implements QItemDelegate for displaying text using the R formatting engine. This allows table and tree widgets (and any other widget that delegates to ItemDelegate) to correctly display NA values and to display numbers using the scipen and digits options.

Usage

qrTextFormattingDelegate(parent = NULL)

Arguments

parent The parent object, or NULL for none.

Value

An RTextFormattingDelegate object

Author(s)

Michael Lawrence

Examples

data(mtcars)
model <- qdataFrameModel(mtcars)
view <- Qt$QTableView()
view$setModel(model)
delegate <- qrTextFormattingDelegate()
view$setItemDelegate(delegate)
view
qsetClass | Define a Qt/C++ class in R

**Description**

C++ libraries like Qt often expect/require clients to extend classes in the API. qsetClass will define a class in terms of its name, parent (single inheritance) and constructor. Methods are added with qsetMethod, qsetSlot, and qsetSignal. The qsetProperty function defines new properties. Calling qsetRefClass on an RQtClass creates a corresponding "RS" reference class, thus unifying R/Qt classes with the methods package.

**Usage**

```r
qsetClass(name, parent, constructor = function(...) parent(...),
          where = topenv(parent.frame()))
qsetMethod(name, class, FUN,
           access = c("public", "protected", "private"))
qsetSlot(signature, class, FUN,
       access = c("public", "protected", "private"))
qsetSignal(signature, class,
       access = c("public", "protected", "private"))
qsetProperty(name, class, type = NULL, read = function() this[[.name]],
        write = function(val) this[[.name]] <- val, notify = NULL,
        constant = FALSE, final = FALSE, stored = TRUE, user = FALSE)
qsetRefClass(Class, where = topenv(parent.frame()), ...)
```

**Arguments**

- **name**: The name of the class or method.
- **parent**: Object of `RQtClass` representing the parent. Only single inheritance is supported.
- **constructor**: A function for constructing an instance. By default, arguments are passed to parent constructor. See details.
- **where**: The environment in which to define the class. Behaves like `setClass`. Usually not specified.
- **class, Class**: Object of `RQtClass` on which to define the method.
- **FUN**: The function that implements the method.
- **access**: The access modifier; same meaning as in C++. public methods may be invoked from any context, protected only by methods of this class or a subclass, and private only by methods of this class.
- **signature**: The name and types of the arguments. If there are no arguments, and the no return value, this is just the name. Otherwise, provide the signature in C++ syntax, as in: `int myMethod(int, const char*)` for a method named `myMethod` that accepts two parameters, one an integer and one a string, and then returns an integer. We are essentially using C++ as a DSL for specifying
signatures; the types must be C++ types, because this method is made available to external systems (like DBus and Javascript via QtWebKit).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>type</code></td>
<td>The type name for the property</td>
</tr>
<tr>
<td><code>read</code></td>
<td>The function for reading (getting) the value of this property. By default, this assumes the property is stored as a field named of the form <code>.name</code>.</td>
</tr>
<tr>
<td><code>write</code></td>
<td>The function for writing (setting) the value of this property. By default, this assumes the property is stored as a field named of the form <code>.name</code>.</td>
</tr>
<tr>
<td><code>notify</code></td>
<td>The name of a previously defined signal that is emitted when the property is modified, or <code>NULL</code> for none.</td>
</tr>
<tr>
<td><code>constant</code></td>
<td>A hint to Qt that the property does not change; not actually enforced and rarely used.</td>
</tr>
<tr>
<td><code>final</code></td>
<td>A hint to Qt that the property is not to be overridden; not actually enforced and rarely used.</td>
</tr>
<tr>
<td><code>stored</code></td>
<td>A hint to Qt that the property is stored in the object, i.e., not dynamically computed. Could be helpful when serializing objects.</td>
</tr>
<tr>
<td><code>user</code></td>
<td>Whether the property is the primary user-facing property for this class. Like the current value for a scrollbar. Used when autowiring widgets to data models, as in <code>QDataWidgetMapper</code>.</td>
</tr>
<tr>
<td>...</td>
<td>Additional arguments to pass to <code>setRefClass</code>.</td>
</tr>
</tbody>
</table>

**Details**

The side-effect of `qsetClass` is that a `RQtClass` object for the new `R` class is assigned into the `where` argument.

Within the scope of a method or constructor, the symbols are first resolved against the members of the class (including inherited members). The search then proceeds to the enclosure of the `R` function, and on up the conventional search path.

For chaining up, there is a special function named `super` that is defined differently for methods and constructors. Within a constructor, `super` will invoke the constructor of the super class, as in Java. For a method, the first argument passed to `super` should be the name of the method in the parent class to invoke (also similar to Java).

**Value**

For `qsetClass`, the `RQtClass` object (supports chaining with `qsetMethod`).

For `qsetMethod`, `qsetSlot`, `qsetSignal`, and `qsetProperty`, the name of the method/property (supports chaining).

For `qsetRefClass`, the reference class generator object corresponding to the `R/C++` class.

**Author(s)**

Michael Lawrence
Examples

e <- Qt$QLineEdit()
qsetClass("positiveValidator", Qt$QValidator)
qsetMethod("validate", positiveValidator, function(input, pos) {
    val <- suppressWarnings(as.integer(input))
    if (!is.na(val)) {
        if (val > 0)
            Qt$QValidator$Acceptable
        else Qt$QValidator$Invalid
    } else {
        if (input == "")
            Qt$QValidator$Acceptable
        else Qt$QValidator$Invalid
    }
})

v <- positiveValidator(e)
e$setValidator(v)
e$show()

qsetStyleSheet
Access style sheets

Description
Convenience functions for defining style sheets in Qt.

Usage
qsetStyleSheet(..., what = ".", widget = NULL, append = TRUE)
qStyleSheet(widget = NULL)

Arguments
... Optionally named arguments specifying style sheet parameters.
what The entity to which the parameters apply.
widget Widget containing the style sheet. If NULL, it applies application-wide.
append Whether the rules should be appended to the existing rules.

Value
For qStyleSheet, the style sheet in textual form.

Author(s)
Michael Lawrence
qtimer  

**Description**

A convenience function for creating a QTimer object and registering an R handler that is fired after a specified delay. The timer continues to fire until it is stopped. This does not work yet.

**Usage**

qtimer(delay, handler)

**Arguments**

- `delay` The delay, in milliseconds.
- `handler` The R function to fire after the delay.

**Value**

A QTimer object. To stop it, call the `stop` method.

**Author(s)**

Michael Lawrence

**Examples**

```r
timer <- qtimer(2000, function() {
  print("time out")
})
timer$singleShot <- TRUE
timer$start()
```

---

**RQtClass  

**Class Definitions**

**Description**

Each class defined by a library has a corresponding RQtClass object. Each object is a function and serves as the constructor for the class. Special behavior is defined, so that the function also behaves as a container of static methods and enumeration values defined by the class.
Details

Besides serving as a constructor, this object contains two types of members: static methods and enumeration values. Their names may be listed by calling `names` on the object. To access a member, use `$`, which supports auto-completion, thanks to the `names` method.

Namespaces within a library are also represented as `RQtClass` objects, even though they are not actually classes.

Author(s)

Michael Lawrence

See Also

`RQtLibrary` (container of class objects), `RQtObject` (an instance of a class)

Examples

```ruby
## calling a constructor
widget <- Qt$QWidget()
## calling a static method
Qt$QWidget$tr("hello world")
## access an enum value within the 'Qt' namespace
Qt$Qt$AbsoluteSize
```

Description

Every library bound by Smoke is represented by a top-level library object, which is an environment containing an `RQtClass` object for each class defined by the library.

Details

Since `RQtLibrary` is essentially a plain old environment object, the conventional environment API behaves as expected. List the names of the available classes with `ls` and retrieve a definition with `$` or `get`, for example.

Author(s)

Michael Lawrence

See Also

`RQtClass`, which describes a class
**RQtObject**

**Instance of a Class**

**Description**

 Represents an instance of a class, as a special type of environment. In this case, the members are methods and, for QObject-derivatives, properties.

**Details**

 The ordinary environment API is supported, with restrictions on assignments. Retrieve objects with $ or get. Replacing a method is not supported. For QObject instances, the members include the properties, which may be set through assignment, e.g., with the $<- method.

 C++ operators are also supported, with the familiar syntax. Most C++ operators have obvious R equivalents. Two exceptions are the >> and << operators, which are defined as the infix functions %>% and %<<% in R.

 Some classes in the Qt API are light-weight, and their instances are usually treated as values. For some of the most prevalent of these types, we define coercion methods to convenient R equivalents. Below, we list the R type corresponding to each Qt type. Coercion methods are defined for the conventional generic for the type, e.g., as.integer for integers. For those types with a variant ending in F, like QSizeF, the F variant yields double values in R, while QSize would be integer.

```
QRect(F) => matrix
QPoint(F) => vector, integer (double)
QSize(F) => vector, integer (double)
QPolygon(F) => matrix
QTransform => matrix
QColor => matrix
```

**Author(s)**

Michael Lawrence

**See Also**

RQtClass (for constructing instances)
Examples

```r
widget <- Qt$QWidget()
widget$setWindowTitle("Hello World")
widget$windowTitle # access a property
widget$windowTitle <- "Hello Again"

size <- qsize(2L, 3L)
as.integer(size * 2) # operators work too
as.integer(size) * 2
```
Index

.onLoad, 9
<<%(RQtObject), 18
>>%(RQtObject), 18
as.QImage, 2
col2rgb, 2, 4
constructors, 3

DataFrameModel, 5
dim-methods, 6
dim.QGraphicsItem (dim-methods), 6
dim.QGraphicsScene (dim-methods), 6
dim.QGraphicsView (dim-methods), 6
dim.QRect (dim-methods), 6
dim.QRectF (dim-methods), 6
ls, 17

qbrush (constructors), 3
qcol2rgb (RQtObject), 18
qcolor (constructors), 3
qconnect, 7
qdataFrame (DataFrameModel), 5
qdataFrame<- (DataFrameModel), 5
qdataFrameModel (DataFrameModel), 5
qfindChild, 8
qfont (constructors), 3
qinvoke, 8
qinvokeStatic (qinvoke), 8
qlibrary, 9
qmethods, 10
qmocMethods, 11
open (constructors), 3
qpoint (constructors), 3
qpolygon (constructors), 3
qproperties (qmethods), 10
qrect (constructors), 3
qTextFormattingDelegate, 12
qsetClass, 13
qsetMethod (qsetClass), 13
qsetProperty (qsetClass), 13
qsetRefClass (qsetClass), 13
qsetSignal, 12
qsetSignal (qsetClass), 13
qsetSlot, 12
qsetSlot (qsetClass), 13
qsetStyleSheet, 15
qsignals (qmocMethods), 11
qsize (constructors), 3
qslots (qmocMethods), 11
qstyleSheet (qsetStyleSheet), 15
Qt (RQtLibrary), 17
qtimer, 16
qtransform (constructors), 3
RQtClass, 13, 16, 17, 18
RQtLibrary, 17, 17
R(Qt)Object, 4, 17, 18
RQtObject-class (R(Qt)Object), 18

setClass, 13
xy.coords, 3