Package: RKernel (via r-universe)

September 4, 2024

Title Yet another R kernel for Jupyter

Version 1.0

Date 2024-09-04

Maintainer Martin Elff <martin@elff.eu>

Description Provides a kernel for Jupyter.

Encoding UTF-8

License GPL-2

Imports tools, utils, pbdZMQ (>= 0.2-1), jsonlite (>= 0.9.6), uuid, digest, R6, svglite, repr, htmltools, htmlwidgets, base64enc, httpgd, callr, processx

Depends R (>= 3.0.1)

Suggests roxygen2

Enhances memisc, shiny

SystemRequirements jupyter

RoxygenNote 7.3.1

Collate 'json.R' 'display.R' 'GraphicsClass.R' 'View.R' 'aa-versions.R' 'callbacks.R' 'cell-options.R' 'check-options.R' 'comms.R' 'compare-plots.R' 'completions.R' 'dap-helpers.R' 'dap-server.R' 'dataTable.R' 'dictionary.R' 'env-browser.R' 'events.R' 'help.R' 'hooks.R' 'inspect-reply.R' 'install.R' 'is-complete.R' 'json-connect.R' 'kernel.R' 'magic-codes.R' 'main.R' 'output.R' 'proto.R' 'scrolling-table.R' 'session.R' 'sharedHelp.R' 'startup.R' 'traitlets.R' 'traitlets-R6instance.R' 'traitlets-boolean.R' 'traitlets-bytes.R' 'traitlets-unicode.R' 'traitlets-color.R' 'traitlets-vector.R' 'traitlets-datetime.R' 'traitlets-numbers.R' 'traitlets-str_enum.R' 'utils.R' 'widget.R' 'widget-layout.R' 'widget-core.R' 'widget-dom.R' 'widget-description.R' 'widget-value.R' 'widget-boolean.R' 'widget-box.R' 'widget-button.R' 'widget-color.R' 'widget-date.R' 'widget-datetime.R' 'widget-env-browser.R' 'widget-float.R' 'widget-integer.R' 'widget-output.R'

'widget-interaction.R' 'widget-media.R' 'widget-play.R'
'widget-plot.R' 'widget-progress.R' 'widget-slider.R'
'widget-selection.R' 'widget-selectioncontainer.R'
'widget-sidecar.R' 'widget-string.R' 'widget-tagsinput.R'
'widget-templates.R' 'widget-time.R' 'widget-upload.R'
'widget-virtual-table.R' 'widget_link.R'

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

RemoteUrl https://github.com/melff/RKernel

RemoteRef HEAD

RemoteSha bf95ece5730041b087fa72670ebbddbbf57a9e5a

Contents

`add_displayed_classes` *Add a Class to the 'Displayed' Ones*

Description

Add a class to those who are output using `display()` when they are autoprinted, i.e. returned as the value of the last expression in a Jupyter notebook cell.

Usage

```
add_displayed_classes(x)
```

Arguments

x	A character string, the name of a class.

`add_paged_classes` *Add a Class to the 'Paged' Ones*

Description

Add a class to those who are output using `Page()` when they are autoprinted, i.e. returned as the value of the last expression in a Jupyter notebook cell.

Usage

```
add_paged_classes(x)
```

Arguments

x	A character string, the name of a class.

alert *Create an alert in the frontend*

Description

Creates Javascript code and sends it to the frontend that opens an alert box in the browser.

Usage

```
alert(text)
```

Arguments

text	A character string with the text that appears in the

Boolean *Boolean Traitlets*

Description

A class and a constructor function to create boolean trait(let)s.

Usage

```
Boolean(...)
```

Arguments

...	Arguments that are passed to the initialize method of 'BooleanClass'

Super class

RKernel::Trait -> Boolean

Public fields

value A logical vector, usually of length 1

optional Logical, whether an initializing logical value must be provided

coerce Logical, whether 'as.logical()' is implicitely used when a value is assigned to the trait

length Integer, the length of the logical vector that poses as the value of the traitlet.

Methods

Public methods:

- BooleanClass$validator()
- BooleanClass$new()
- BooleanClass$clone()

Method `validator()`**:** A validator method

Usage:

```
BooleanClass$validator(value)
```

Arguments:

`value` A value to be checked for validity

Method `new()`**:** The initializing method

Usage:

```
BooleanClass$new(
  initial,
  coerce = TRUE,
  optional = length(initial) == 0,
  length = 1
)
```

Arguments:

`initial` A value with which the traitlet is initialized

`coerce` Logical, used to initialize the 'coerce' field

`optional` Logical, used to initialize the 'optional' field

`length` Integer, used to initialize the 'length' field

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`BoundedFloatRangeWidget`

Widgets for Floating Point Number Ranges

Description

An R6 class and a constructor function for the creation of widgets that can be used to manipulate floating pairs of point numbers that are bounded within an interval, where a pair defines a number range.

Usage

```
BoundedFloatRangeWidget(value, min, max, ...)
```

Arguments

value	A pair of floating point values.
min	The lower bound of the enclosing interval.
max	The upper bound of the enclosing interval.
...	Other arguments, passed to the superclass initializer.

Details

The function BoundedFloatRangeWidget creates objects of the R6 Class "BoundedFloatRangeWidgetClass", which in turn have the S3 class attribute "BoundedFloatRangeWidget"

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> BoundedFloatRangeWidget

Public fields

value A Float traitlet.

min A Float traitlet, the minimum allowed value.

max A Float traitlet, the maximum allowed value.

Methods

Public methods:

- BoundedFloatRangeWidgetClass$validate_value()
- BoundedFloatRangeWidgetClass$validate_min()
- BoundedFloatRangeWidgetClass$validate_max()
- BoundedFloatRangeWidgetClass$new()
- BoundedFloatRangeWidgetClass$clone()

Method validate_value(): Validate the "value" after assignment.

Usage:

```
BoundedFloatRangeWidgetClass$validate_value(value)
```

Arguments:

value A value, should be numeric.

Method validate_min(): Validate the "min" field after assignment.

Usage:

```
BoundedFloatRangeWidgetClass$validate_min(min)
```

Arguments:

`min` A minimum value, should be numeric.

Method `validate_max()`**:** Validate the "max" field after assignment.

Usage:

```
BoundedFloatRangeWidgetClass$validate_max(max)
```

Arguments:

`max` A maximum value, should be numeric.

Method `new()`**:**

Usage:

```
BoundedFloatRangeWidgetClass$new(...)
```

Arguments:

`...` Arguments passed to the superclass initializer.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`BoundedFloatText` *Widgets for Text Elements with Floating Point Numbers Bounded within an Interval*

Description

An R6 class and a constructor function for the creation of widgets that can be used to manipulate floating point numbers.

Usage

```
BoundedFloatText(value = 0, min = 0, max = 100, step = 0.1, ...)
```

Arguments

`value`	Initial value of the floating point number.
`min`	Lower limit of the enclosing interval.
`max`	Upper limit of the enclosing interval.
`step`	Increment by which the number is increased or decreased by the text field controls.
`...`	Other arguments.

Details

The function BoundedFloatText creates objects of the R6 Class "BoundedFloatTextClass", which in turn have the S3 class attribute "BoundedFloatText".

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::BoundedFloatWidget -> BoundedFloatText

Public fields

_model_name Name of the Javascript model in the frontend.

_view_name Name of the Javascript view in the frontend.

disabled A Boolean traitlet, whether the text widget is disabled.

continuous_update A Boolean traitlet, whether the text widget is continuously updated upon change in the frontend.

step A Float traitlet, a step size by which the value is incremented or decremented if the arrows are clicked.

Methods

Public methods:

- BoundedFloatTextClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

BoundedFloatWidget *Widgets for Bounded Floating Point Numbers*

Description

An R6 class and a constructor function for the creation of widgets that can be used to manipulate floating point numbers that are bounded within an interval

Usage

```
BoundedFloatWidget(value, min, max, ...)
```

Arguments

value	The floating point value
min	The lower bound of the interval
max	The upper bound of the interval
...	Other arguments, passed to the superclass initializer

Details

The function BoundedFloatWidget creates objects of the R6 Class "BoundedFloatWidgetClass", which in turn have the S3 class attribute "BoundedFloatWidget"

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> BoundedFloatWidget

Public fields

value A Float traitlet.

min A Float traitlet, the minimum allowed value.

max A Float traitlet, the maximum allowed value.

Methods

Public methods:

- BoundedFloatWidgetClass$validate_value()
- BoundedFloatWidgetClass$validate_min()
- BoundedFloatWidgetClass$validate_max()
- BoundedFloatWidgetClass$new()
- BoundedFloatWidgetClass$clone()

Method validate_value(): Validate the "value" after assignment.

Usage:

```
BoundedFloatWidgetClass$validate_value(value)
```

Arguments:

value A value, should be numeric.

Method validate_min(): Validate the "min" field after assignment.

Usage:

```
BoundedFloatWidgetClass$validate_min(min)
```

Arguments:

min A minimum value, should be numeric.

Method validate_max(): Validate the "max" field after assignment.

Usage:

```
BoundedFloatWidgetClass$validate_max(max)
```

Arguments:

`max` A maximum value, should be numeric.

Method `new()`**:** Initialize an object.

Usage:

```
BoundedFloatWidgetClass$new(value, min, max, ...)
```

Arguments:

`value` The floating point value.

`min` The lower bound of the interval.

`max` The upper bound of the interval.

`...` Other arguments, passed to the superclass initializer.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`BoundedIntText` *Widgets for Text Elements with Integer Numbers Bounded within an Interval*

Description

An R6 class and a constructor function for the creation of widgets that can be used to manipulate integer numbers

Usage

```
BoundedIntText(value = 0, min = 0, max = 100, step = 1, ...)
```

Arguments

`value`	Initial value of the integer number
`min`	Lower limit of the enclosing interval
`max`	Upper limit of the enclosing interval
`step`	Increment by which the number is increased or decreased by the text field controls
`...`	Arguments passed to the superclass constructor

Details

The function BoundedIntText creates objects of the R6 Class "BoundedIntTextClass", which in turn have the S3 class attribute "BoundedIntText"

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::BoundedIntWidget -> BoundedIntText

Public fields

_model_name Name of the Javascript model in the frontend.

_view_name Name of the Javascript view in the frontend.

disabled A Boolean traitlet, whether the text widget is disabled.

continuous_update A Boolean traitlet, whether the text widget is continuously updated upon change in the frontend.

step A Integer traitlet, a step size by which the value is incremented or decremented if the arrows are clicked.

Methods

Public methods:

- BoundedIntTextClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

BoundedIntWidget *Widgets for Bounded Integer Numbers*

Description

An R6 class and a constructor function for the creation of widgets that can be used to manipulate integer numbers that are bounded within an interval.

Usage

```
BoundedIntWidget(value, min, max, ...)

BoundedIntRangeWidget(value, min, max, ...)
```

Arguments

value	A pair of integer values
min	The lower bound of the enclosing interval
max	The upper bound of the enclosing interval
...	Other arguments, passed to the superclass initializer

Details

The function BoundedIntWidget creates objects of the R6 Class "BoundedIntWidgetClass", which in turn have the S3 class attribute "BoundedIntWidget".

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> BoundedIntWidget

Public fields

value A Integer traitlet.

min A Integer traitlet, the minimum allowed value.

max A Integer traitlet, the maximum allowed value.

Methods

Public methods:

- BoundedIntWidgetClass$validate_value()
- BoundedIntWidgetClass$validate_min()
- BoundedIntWidgetClass$validate_max()
- BoundedIntWidgetClass$new()
- BoundedIntWidgetClass$clone()

Method validate_value(): Validate the "value" after assignment.

Usage:

```
BoundedIntWidgetClass$validate_value(value)
```

Arguments:

value A value, should be an integer number.

Method validate_min(): Validate the "min" field after assignment.

Usage:

```
BoundedIntWidgetClass$validate_min(min)
```

Arguments:

min A minimum value, should be an integer number.

Method validate_max(): Validate the "max" field after assignment.

Usage:

```
BoundedIntWidgetClass$validate_max(max)
```

Arguments:

`max` A maximum value, should be an integer number.

Method `new()`**:**

Usage:

```
BoundedIntWidgetClass$new(...)
```

Arguments:

`...` Arguments passed to the superclass initializer

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `BoundedIntRangeWidget`

Public fields

`value` A Integer traitlet.

`min` A Integer traitlet, the minimum allowed value.

`max` A Integer traitlet, the maximum allowed value.

Methods

Public methods:

- `BoundedIntRangeWidgetClass$validate_value()`
- `BoundedIntRangeWidgetClass$validate_min()`
- `BoundedIntRangeWidgetClass$validate_max()`
- `BoundedIntRangeWidgetClass$new()`
- `BoundedIntRangeWidgetClass$clone()`

Method `validate_value()`**:** Validate the "value" after assignment.

Usage:

```
BoundedIntRangeWidgetClass$validate_value(value)
```

Arguments:

`value` A value, should be an integer number.

Method `validate_min()`**:** Validate the "min" field after assignment.

Usage:

```
BoundedIntRangeWidgetClass$validate_min(min)
```

Arguments:

`min` A minimum value, should be an integer number.

Method `validate_max()`**:** Validate the "max" field after assignment.

Usage:

```
BoundedIntRangeWidgetClass$validate_max(max)
```

Arguments:

`max` A maximum value, should be an integer number.

Method `new()`**:**

Usage:

```
BoundedIntRangeWidgetClass$new(...)
```

Arguments:

`...` Arguments passed to the superclass initializer.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`BoundedLogFloatWidget` *Widgets for Bounded Floating Point Numbers on a logarithmic scale*

Description

An R6 class and a constructor function for the creation of widgets that can be used to manipulate floating point numbers that are bounded within an interval on an logarithmic scale

Usage

```
BoundedLogFloatWidget(value, min, max, base, ...)
```

Arguments

`value`	The floating point value.
`min`	The lower bound of the interval.
`max`	The upper bound of the interval.
`base`	The base of the logarithm.
`...`	Other arguments, passed to the superclass initializer.

Details

The function BoundedLogFloatWidget creates objects of the R6 Class "BoundedLogFloatWidget-Class", which in turn have the S3 class attribute "BoundedLogFloatWidget"

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> BoundedLogFloatWidget

Public fields

value A Float traitlet.

min A Float traitlet, the minimum allowed value.

max A Float traitlet, the maximum allowed value.

base A Float traitlet, the logarithmic base.

Methods

Public methods:

- BoundedLogFloatWidgetClass$validate_value()
- BoundedLogFloatWidgetClass$validate_min()
- BoundedLogFloatWidgetClass$validate_max()
- BoundedLogFloatWidgetClass$new()
- BoundedLogFloatWidgetClass$clone()

Method validate_value(): Validate the "value" after assignment.

Usage:

```
BoundedLogFloatWidgetClass$validate_value(value)
```

Arguments:

value A value, should be numeric.

Method validate_min(): Validate the "min" field after assignment.

Usage:

```
BoundedLogFloatWidgetClass$validate_min(min)
```

Arguments:

min A minimum value, should be numeric.

Method validate_max(): Validate the "max" field after assignment.

Usage:

```
BoundedLogFloatWidgetClass$validate_max(max)
```

Arguments:

max A maximum value, should be numeric.

Method new(): Initialize an object.

Usage:

```
BoundedLogFloatWidgetClass$new(value, min, max, ...)
```

Arguments:

`value` The floating point value.

`min` The lower bound of the interval.

`max` The upper bound of the interval.

`...` Other arguments, passed to the superclass initializer.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Boxes` *Box Containers*

Description

Classes and constructor functions to create box container widgets

Usage

```
Box(..., layout = NULL)

HBox(..., layout = NULL, wrap = FALSE)

VBox(..., layout = NULL, wrap = FALSE)

GridBox(..., layout = NULL)
```

Arguments

`...`	Arguments passed to the superclass constructor
`wrap`	Logical value, whether lines of widgets should be wrapped?

Details

`VBox` creates vertical boxes, `HBox` creates horizontal boxes, `GridBox` creates a grid box. In a vertical box widgets are arranged one below the other, in a horizontal box widgets are arranged side-by-side, in a grid box widget are arraned in a grid.

Functions

- `Box()`: A baseline box constructor
- `HBox()`: A horizontal box constructor
- `VBox()`: A vertical box constructor
- `GridBox()`: A grid box constructor

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `Box`

Public fields

`_model_module` Name of the Javascript module with the model

`_model_module_version` Version of the module where the model is defined

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript model view in the frontend

`_view_module` Name of the module where the view is defined

`_view_module_version` Version of the module where the view is defined

`children` A generic vector with the widgets in the container

`box_style` The string that describes the button style

Methods

Public methods:

- `BoxClass$new()`
- `BoxClass$notify_children_displayed()`
- `BoxClass$clone()`

Method `new()`**:** An initializer function

Usage:

```
BoxClass$new(children = list(), ...)
```

Arguments:

`children` A list of widgets

`...` Other arguments, passed to the superclass method

Method `notify_children_displayed()`**:** Notifies children that they are displayed

Usage:

```
BoxClass$notify_children_displayed()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::Box -> HBox

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript model view in the frontend

Methods

Public methods:

- HBoxClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::Box -> VBox

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript model view in the frontend

Methods

Public methods:

- VBoxClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::Box -> GridBox

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript model view in the frontend

Methods

Public methods:

- GridBoxClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Button` *Buttons*

Description

Classes and constructor functions for boxes and box styles

Usage

```
Button(...)

ButtonStyle(...)
```

Arguments

`...`	Arguments passed to the inializer

Functions

- `Button()`: A button constructor
- `ButtonStyle()`: A constructor for a button style

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `Button`

Public fields

`_model_module` Name of the Javascript module with the model

`_model_module_version` Version of the module where the model is defined

`_model_name` Name of the Javascript model in the frontend

`_view_module` Name of the module where the view is defined

`_view_module_version` Version of the module where the view is defined

`_view_name` Name of the Javascript model view in the frontend

`description` A button description

`disabled` Boolean, whether the button is disabled

`icon` Name of an optional icon

`button_style` The string that describes the button style

`tooltip` An optional tooltip string

`style` The button style, an object of class "ButtonStyleClass"

Methods

Public methods:

- `ButtonClass$on_click()`
- `ButtonClass$click()`
- `ButtonClass$clone()`

Method `on_click()`**:** Add or remove a click handler

Usage:

```
ButtonClass$on_click(handler, remove = FALSE)
```

Arguments:

`handler` A function that is called when the button is clicked

`remove` Logical value, whether the handler is to be removed

Method `click()`**:** Function that calls the click event handlers

Usage:

```
ButtonClass$click()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DescriptionStyle` -> `ButtonStyle`

Public fields

`_model_name` Name of the Javascript model in the frontend

`button_color` The colour of the button

`font_family` The font family of the button label

`font_size` The font size of the button label

`font_style` The font style of the button label

`font_variant` The font variant of the button label

`font_weight` The font weight of the button label

`text_color` The text colour of the button label

`text_decoration` The text decoration of the button label

Methods

Public methods:

- ButtonStyleClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Bytes` *Raw Bytes Traitlets*

Description

A class and a constructor function to create (raw) bytes trait(let)s.

Usage

```
Bytes(...)
```

Arguments

`...`	Arguments that are passed to the initialize method of 'BytesClass'

Super class

`RKernel::Trait` -> `Bytes`

Public fields

`value` A raw bytes vector

`optional` Logical, whether an initializing logical value must be provided

`coerce` Logical, whether 'as.raw()' is implicitely used when a value is assigned to the trait

Methods

Public methods:

- BytesClass$validator()
- BytesClass$new()
- BytesClass$clone()

Method `validator()`**:** A validator method

Usage:

```
BytesClass$validator(value)
```

Arguments:

`value` A value to be checked for validity

Method `new()`**:** The initializing method

Usage:

```
BytesClass$new(initial = raw(0), coerce = TRUE, optional = TRUE)
```

Arguments:

`initial` A value with which the traitlet is initialized

`coerce` Logical, used to initialize the 'coerce' field

`optional` Logical, used to initialize the 'optional' field

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`CallbackDispatcher` *A Dispatcher for Callbacks*

Description

Objects in this class are collections of callbacks functions usually related to certain events. The function `CallbackDispachter` can be used as an constructor

Usage

```
CallbackDispatcher(...)
```

Arguments

`...`	Arguments passed to the inializer

Functions

- `CallbackDispatcher()`: The constructor function, returns an Object of Class "CallbackDispatcherClass"

Methods

Public methods:

- CallbackDispatcherClass$register()
- CallbackDispatcherClass$clear()
- CallbackDispatcherClass$suspend_handlers()
- CallbackDispatcherClass$activate_handlers()
- CallbackDispatcherClass$run()
- CallbackDispatcherClass$clone()

Method `register()`**:** Register a function as a callback

Usage:

```
CallbackDispatcherClass$register(handler, remove)
```

Arguments:

`handler` A function

`remove` A logical value; whether the function is added or removed from the list of callbacks

Method `clear()`**:** Remove all callback functions

Usage:

```
CallbackDispatcherClass$clear()
```

Method `suspend_handlers()`**:** Suspend registered callback functions

Usage:

```
CallbackDispatcherClass$suspend_handlers()
```

Method `activate_handlers()`**:** (Re-)activate registered callback functions

Usage:

```
CallbackDispatcherClass$activate_handlers()
```

Method `run()`**:** Run all registered callback functions

Usage:

```
CallbackDispatcherClass$run(...)
```

Arguments:

`...` Aruments passed on to the handler functions

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

cell.options *Set options locally for the current jupyter notebook cell*

Description

Set options locally for the current jupyter notebook cell

Usage

```
cell.options(...)
```

Arguments

...	Options, see options.

cell.par *Set graphics parameters locally for the current jupyter notebook cell*

Description

Set graphics parameters locally for the current jupyter notebook cell

Usage

```
cell.par(...)
```

Arguments

...	Graphics parameters, see par.

Checkboxes *Checkbox Widgets*

Description

A class and a constructor function to create checkbox widgets

Usage

```
CheckboxStyle(...)

Checkbox(...)
```

Arguments

...	Arguments passed to the inializer

Functions

- `CheckboxStyle()`: The constructor for checkbox widgets
- `Checkbox()`: The constructor for checkbox styles

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DescriptionStyle` -> `CheckboxStyle`

Public fields

`_model_name` Name of the Javascript model in the frontend

`background` The background color

`required_version` Minimum required ipywidgets version in which the current widget class is supported.

Methods

Public methods:

- `CheckboxStyleClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `Checkbox`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript model view in the frontend

`description` A button description

`disabled` Boolean, whether the button is disabled

`indent` Boolean, whether to indent the checkbox

`value` Boolean, whether the box is checked

`style` The checkbox style, an object of class "CheckboxStyleClass"

Methods

Public methods:

- `CheckboxClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`ColorPicker` *Color picker widgets*

Description

A class and constructor function to create color picker widgets

Usage

```
ColorPicker(...)
```

Arguments

`...`	Arguments passed to the inializer

Functions

- `ColorPicker()`: The constructor for color-picker widgets

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `ColorPicker`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript model view in the frontend

`concise` Boolean, whether the a short version should be shown

`disabled` Boolean, whether the button is disabled

`value` Unicode string, the color value

Methods

Public methods:

- ColorPickerClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`ColorTrait` *A Color String Trait*

Description

A Color String Trait

A Color String Trait

Usage

```
Color(...)
```

Arguments

`...`	Arguments passed to the trait instance initializer

Super classes

`RKernel::Trait` -> `RKernel::Unicode` -> `Color`

Public fields

`optional` Logical value, whether a length-zero value is allowed.

Methods

Public methods:

- ColorTraitClass$validator()
- ColorTraitClass$clone()

Method `validator()`**:** Check the value assigned to the traitlet.

Usage:

```
ColorTraitClass$validator(value)
```

Arguments:

`value` The value assigned to the traitlet.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Comm` *Comms - connections between the kernel and the frontend*

Description

This R6 Class provides for bidirectional communication between the R Kernel and the Jupyter frontend, e.g. a Jupyter notebook

Usage

```
Comm(...)
```

Arguments

`...`	Arguments passed to the inializer

Details

Objects of this class are used to communicate to the frontend via custom messages.

Functions

- `Comm()`: A constructor function for objects of class "CommClass"

Public fields

`id` A character string, the comm id

`target_name` A character string, the target

`handlers` A list of handler functions

`data` A list of data

Methods

Public methods:

- CommClass$new()
- CommClass$open()
- CommClass$send()
- CommClass$close()
- CommClass$clone()

Method `new()`**:** Initialize a 'Comm' object

Usage:

```
CommClass$new(target_name, id = uuid(), handlers = list())
```

Arguments:

`target_name` A string, the name of the target

`id` A string, the comm id

`handlers` A list of handler functions

`kernel` The relevant kernel

Method `open()`**:** Open a comm

Usage:

```
CommClass$open(data, metadata = emptyNamedList, buffers = NULL)
```

Arguments:

`data` A named list

`metadata` A named list

`buffers` A list of raw vectors or NULL

Method `send()`**:** Send data through a comm

Usage:

```
CommClass$send(data, metadata = emptyNamedList, buffers = NULL)
```

Arguments:

`data` A named list

`metadata` A named list

`buffers` A list of raw vectors or NULL

Method `close()`**:** Close a comm

Usage:

```
CommClass$close(
  data = emptyNamedList,
  metadata = emptyNamedList,
  buffers = NULL
)
```

Arguments:

`data` A named list

`metadata` A named list

`buffers` A list of raw vectors or NULL

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`CommManager` *A Manager for Comms*

Description

Objects of this class are used internally to manage comms, they are not meant to be used by end-users.

Usage

```
CommManager(...)
```

Arguments

...	Arguments passed to the inializer

Details

See the documentation of Jupyter custom messages.

Functions

- `CommManager()`: A constructor for objects in the 'CommManagerClass'

Public fields

`comms` A list of Comms.

Methods

Public methods:

- `CommManagerClass$add_handlers()`
- `CommManagerClass$remove_handlers()`
- `CommManagerClass$has_handlers()`
- `CommManagerClass$get_handlers()`
- `CommManagerClass$get_comms()`
- `CommManagerClass$new_comm()`

- CommManagerClass$handle_open()
- CommManagerClass$handle_close()
- CommManagerClass$handle_msg()
- CommManagerClass$send()
- CommManagerClass$send_open()
- CommManagerClass$send_close()
- CommManagerClass$list_targets()
- CommManagerClass$clone()

Method `add_handlers()`**:** Add a handler for a comm target

Usage:

```
CommManagerClass$add_handlers(target_name, handlers)
```

Arguments:

`target_name` A string, the name of the target.
`handlers` A named list of handlers

Method `remove_handlers()`**:** Remove the handlers of a comm target

Usage:

```
CommManagerClass$remove_handlers(target_name)
```

Arguments:

`target_name` A string, the name of the target

Method `has_handlers()`**:** Check if handlers for a target exist

Usage:

```
CommManagerClass$has_handlers(target_name)
```

Arguments:

`target_name` A string, the name of the target

Method `get_handlers()`**:** Get the handlers for a target

Usage:

```
CommManagerClass$get_handlers(target_name)
```

Arguments:

`target_name` A string, the name of the target

Method `get_comms()`**:** Get all comms or all comms related to a target

Usage:

```
CommManagerClass$get_comms(target_name = NULL)
```

Arguments:

`target_name` A string, the name of the target or NULL. If NULL,

Method `new_comm()`**:** Create a new comm related to a target

Usage:

```
CommManagerClass$new_comm(target_name)
```

Arguments:

`target_name` A string, the name of the target

Method `handle_open()`**:** Handle a 'comm open' request from the frontend

Usage:

```
CommManagerClass$handle_open(target_name, id, data)
```

Arguments:

`target_name` A string, the name of the target

`id` A string, the comm id

`data` Data sent by the frontend

Method `handle_close()`**:** Handle a 'comm close' request from the frontend

Usage:

```
CommManagerClass$handle_close(id, data)
```

Arguments:

`id` A string, the comm id

`data` Data sent by the frontend

Method `handle_msg()`**:** Handle a comm message from the frontend

Usage:

```
CommManagerClass$handle_msg(id, data)
```

Arguments:

`id` A string, the comm id

`data` Data sent by the frontend

Method `send()`**:** Send data to the frontend

Usage:

```
CommManagerClass$send(id, data, metadata = emptyNamedList, buffers = NULL)
```

Arguments:

`id` A string, the comm id

`data` A named list

`metadata` A named list

`buffers` A list of raw vectors or NULL

Method `send_open()`**:** Send an 'open' request to the frontend

Usage:

```
CommManagerClass$send_open(
  id,
  target_name,
  data,
  metadata = emptyNamedList,
  buffers = NULL
)
```

Arguments:

`id` A string, the comm id

`target_name` A string, the name of the target

`data` A named list

`metadata` A named list

`buffers` A list of raw vectors or NULL

Method `send_close()`**:** Send an 'close' request to the frontend

Usage:

```
CommManagerClass$send_close(
  id,
  data = emptyNamedList,
  metadata = emptyNamedList,
  buffers = NULL
)
```

Arguments:

`id` A string, the comm id

`data` A named list

`metadata` A named list

`buffers` A list of raw vectors or NULL

Method `list_targets()`**:** Return a list of targets

Usage:

```
CommManagerClass$list_targets()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`CoreWidgetClass` *Yet Another Widget Class*

Description

This is a base class for widgets that use the 'controls' module.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `CoreWidget`

Public fields

`_model_module` Name of the Javascript module with the model

`_model_module_version` Version of the module where the model is defined

`_model_name` Name of the Javascript model in the frontend

`_view_module` Version of the module where the view is defined

`_view_module_version` Version of the module where the view is defined

Methods

Public methods:

- `CoreWidgetClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`CSS` *Send CSS code to the frontend*

Description

Send CSS code in a character string or a text file to the frontend.

Usage

```
CSS(text, file)
```

Arguments

`text`	A character string with CSS styling information
`file`	Path of a file with CSS styling information

`dataTable` *HTML Tables with Interactive Controls*

Description

Objects of class "dataTable" provide HTML tables with interactive controls powered by the DataTable Javascript library.

Usage

```
dataTable(x, ...)

## Default S3 method:
dataTable(x, ...)

## S3 method for class 'data.frame'
dataTable(x, ...)

## S3 method for class 'dataTable'
display_data(
  x,
  ...,
  metadata = emptyNamedList,
  id = attr(x, "id"),
  update = FALSE
)
```

Arguments

`x`	A "dataTable" object
`...`	Other arguments passed to the initialization method of 'dataTableClass' R6 objects
`metadata`	A list of metadata strings
`id`	An ID string
`update`	A logical value; whether an existing display item will be updated

Methods (by class)

- `dataTable(default)`: Default method
- `dataTable(data.frame)`: data.frame method

Methods (by generic)

- `display_data(dataTable)`: dataTable method for display_data

Functions

- `dataTable()`: A dataTable constructor

Public fields

`w` A container widget or NULL

`page` Number of the current page

`m` Width of the object divided by 'size'

`r` Remainder of the widht of the object divided by 'size'

`size` Number of columns in each group for horizontal paging

`iframe` An <iframe> container or NULL

`b_left` Button to scroll left

`b_right` Button to scroll right

`b_first` Button to scroll to the first group of columns

`b_last` Button to scroll to the last group of columns

`dt` HTML code for the visible table

`obj` The tabular object being dispayed

`label` A string label that shows the columns being displayed

`style` A string with CSS styling

`navigator` A container widgets that contains the navigator buttons

`scrollY` The vertical scroll amount

`height` The height of the iframe

Methods

Public methods:

- `dataTableClass$new()`
- `dataTableClass$show_columns()`
- `dataTableClass$page_left()`
- `dataTableClass$page_right()`
- `dataTableClass$page_first()`
- `dataTableClass$page_last()`
- `dataTableClass$clone()`

Method `new()`**:** Initialize the DataTable

Usage:

```
dataTableClass$new(obj, size = 50, nlines = min(nrow(obj), 15), ...)
```

Arguments:

`obj` The object to be displayed

`size` An integer, the number of columns pre-formatted on each page.

`nlines` An integer, the number of rows of each page

... Other arguments, ignored

Method `show_columns()`**:** Show which columns are displayed

Usage:

```
dataTableClass$show_columns()
```

Method `page_left()`**:** Go one page to the left

Usage:

```
dataTableClass$page_left()
```

Method `page_right()`**:** Go one page to the right

Usage:

```
dataTableClass$page_right()
```

Method `page_first()`**:** Go to the first page (to the left)

Usage:

```
dataTableClass$page_first()
```

Method `page_last()`**:** Go to the last page (to the right)

Usage:

```
dataTableClass$page_last()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`DateClass` *Date Traitlets*

Description

A class and constructor of date traitlets.

Usage

```
Date(...)

## S3 method for class 'DateClass'
as.Date(x, ...)
```

Arguments

`...`	Other arguments.
`x`	A date traitlet.

Super class

`RKernel::Trait` -> `DateClass`

Public fields

`value` A date.

`coerce` Logical value, whether assignments to the value field should be coerced to the appropriate type.

Methods

Public methods:

- `DateClass$validator()`
- `DateClass$new()`
- `DateClass$clone()`

Method `validator()`**:** Check the value assigned to the traitlet.

Usage:

```
DateClass$validator(value)
```

Arguments:

`value` The value assigned to the traitlet.

Method `new()`**:** Initialize the traitlet.

Usage:

```
DateClass$new(
  initial = as.Date(integer(0)),
  year = integer(0),
  month = integer(0),
  day = integer(0),
  coerce = TRUE
)
```

Arguments:

`initial` An optional Date object or date string

`year` An optional integer

`month` An optional integer

`day` An optional integer

`coerce` An optional logical value

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

DatePicker *Date Picker Widgets*

Description

An R6 class and constructor function for date picker widgets

Usage

```
DatePicker(...)
```

Arguments

...	Arguments passed to the inializer

Functions

- `DatePicker()`: A constructor for dat picker widgets

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `DatePicker`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`value` The date

`disabled` Boolean, whether the user can make changes

`min` Minimum selectable date

`max` Maximum selectable date

`step` Date step used for the picker in days

Methods

Public methods:

- `DatePickerClass$validate_value()`
- `DatePickerClass$validate_min()`
- `DatePickerClass$validate_max()`
- `DatePickerClass$new()`
- `DatePickerClass$clone()`

Method `validate_value()`**:** Check wether "value" is within range.

Usage:

```
DatePickerClass$validate_value(value)
```

Arguments:

`value` A value

Method `validate_min()`**:** Validate the "min" field after assignment.

Usage:

```
DatePickerClass$validate_min(min)
```

Arguments:

`min` A minimum value, should be an integer number.

Method `validate_max()`**:** Validate the "max" field after assignment.

Usage:

```
DatePickerClass$validate_max(max)
```

Arguments:

`max` A maximum value, should be an integer number.

Method `new()`**:**

Usage:

```
DatePickerClass$new(...)
```

Arguments:

`...` Arguments passed to the superclass initializer

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`DatetimeClass` *Datetime Traitlets*

Description

A class and constructor of datetime traitlets.

Usage

```
Datetime(...)

Time(...)
```

Arguments

`...`	Arguments that are passed to the initialize method of 'TimeClass'

Super class

RKernel::Trait -> DatetimeClass

Public fields

value A date.

coerce Logical value, whether assignments to the value field should be coerced to the appropriate type.

Methods

Public methods:

- DatetimeClass$validator()
- DatetimeClass$new()
- DatetimeClass$clone()

Method validator(): Check the value assigned to the traitlet.

Usage:

```
DatetimeClass$validator(value)
```

Arguments:

value The value assigned to the traitlet.

Method new(): Initialize the traitlet.

Usage:

```
DatetimeClass$new(initial = as.POSIXct(integer(0)), coerce = TRUE)
```

Arguments:

initial An optional POSIXct object or an object coercive into such an object

coerce An optional logical value

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

`DatetimePicker` *Datetime Picker Widgets*

Description

An R6 class and constructor function for datetime picker widgets

Usage

```
DatetimePicker(...)

NaiveDatetimePicker(...)
```

Arguments

`...`	Arguments passed to the inializer

Functions

- `DatetimePicker()`: A constructor for dat picker widgets
- `NaiveDatetimePicker()`: A constructor for dat picker widgets

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `DatetimePicker`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`value` The date and time. If non-zero length, must have valid timezone info.

`disabled` Boolean, whether the user can make changes

`min` Minimum selectable date and time. If non-zero length, must have valid timezone info.

`max` Maximum selectable date and time. If non-zero length, must have valid timezone info.

Methods

Public methods:

- `DatetimePickerClass$validate_tz()`
- `DatetimePickerClass$validate_value()`
- `DatetimePickerClass$validate_min()`
- `DatetimePickerClass$validate_max()`
- `DatetimePickerClass$new()`
- `DatetimePickerClass$clone()`

Method `validate_tz()`**:** Check whether time zone is valid.

Usage:

```
DatetimePickerClass$validate_tz(value)
```

Arguments:

`value` A value

Method `validate_value()`**:** Check wether "value" is within range.

Usage:

```
DatetimePickerClass$validate_value(value)
```

Arguments:

`value` A date and time to be checked for validity

Method `validate_min()`**:** Validate the "min" field after assignment.

Usage:

```
DatetimePickerClass$validate_min(min)
```

Arguments:

`min` A minimum date and time to be checked for validity

Method `validate_max()`**:** Validate the "max" field after assignment.

Usage:

```
DatetimePickerClass$validate_max(max)
```

Arguments:

`max` A maximum date and time to be checked for validity

Method `new()`**:**

Usage:

```
DatetimePickerClass$new(...)
```

Arguments:

`...` Arguments passed to the superclass initializer

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::DatetimePicker -> NaiveDatetimePicker

Public fields

`_model_name` Name of the Javascript model in the frontend

`value` The date and time. If non-zero length, must have valid timezone info.

`min` Minimum selectable date and time. If non-zero length, must have valid timezone info.

`max` Maximum selectable date and time. If non-zero length, must have valid timezone info.

Methods

Public methods:

- `NaiveDatetimePickerClass$validate_tz()`
- `NaiveDatetimePickerClass$validate_value()`
- `NaiveDatetimePickerClass$new()`
- `NaiveDatetimePickerClass$clone()`

Method `validate_tz()`**:** Check whether time zone is valid.

Usage:

```
NaiveDatetimePickerClass$validate_tz(value)
```

Arguments:

`value` A value

Method `validate_value()`**:** Check wether "value" is within range.

Usage:

```
NaiveDatetimePickerClass$validate_value(value)
```

Arguments:

`value` A date and time to be checked for validity

Method `new()`**:**

Usage:

```
NaiveDatetimePickerClass$new(...)
```

Arguments:

`...` Arguments passed to the superclass initializer

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`DescriptionStyle` *Styling of Decscription Widgets*

Description

Objects of this class contain the CSS styling of description widgets

Usage

```
DescriptionStyle(...)
```

Arguments

`...`	Arguments passed to the inializer

Functions

- `DescriptionStyle()`: A Constructor Function for "DescriptionStyle" objects

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `DescriptionStyle`

Public fields

`_model_module` Name of the Javascript module with the model

`_model_module_version` Version of the module where the model is defined

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript model view in the frontend

`_view_module` Version of the module where the view is defined

`_view_module_version` Version of the module where the view is defined

`description_width` Width of the description

Methods

Public methods:

- `DescriptionStyleClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

DescriptionWidget *Decscription Widgets*

Description

Objects of this class have an optional description and a description tooltip field

Usage

```
DescriptionWidget(...)
```

Arguments

...	Arguments passed to the inializer

Functions

- `DescriptionWidget()`: A Constructor Function for "DescriptionWidget" objects

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `DescriptionWidget`

Public fields

`_model_module` Name of the Javascript module with the model
`_model_module_version` Version of the module where the model is defined
`_view_module` Version of the module where the view is defined
`_view_module_version` Version of the module where the view is defined
`_model_name` Name of the Javascript model in the frontend
`description` An optional description string
`description_tooltip` An optional description tooltip
`tooltip` An optional description tooltip
`description_html` Boolean, whether HTML is allowed in the description
`style` A "DescriptionStyle" object

Methods

Public methods:

- `DescriptionWidgetClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Dict *Dictionary Traitlets*

Description

A class and a constructor of dictionary trait(let)s. These are lists with unique element names.

Usage

```
Dict(...)
```

Arguments

...	Arguments that are passed to the initialize method of 'DictClass'

Super classes

`RKernel::Trait` -> `RKernel::List` -> `Dict`

Methods

Public methods:

- `DictClass$validator()`
- `DictClass$clone()`

Method `validator()`**:** A function that checks the validity of an assigned value, i.e. whether the assigned value is a list with unique names

Usage:

```
DictClass$validator(value)
```

Arguments:

`value` A value to be assigned as the traitlet value

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`dictionary` *A data type analogous to Python dictionaries*

Description

Objects of class "dictionary" behave similar to dictionaries. They can contain any other kind of objects, but like with Python dictionaries, only scalar indices are allowed. Unlike with Python dictionaries, numeric indices can be used well as character indices.

Usage

```
dictionary(...)

## S3 method for class 'dictionary'
x[i]

## S3 replacement method for class 'dictionary'
x[i] <- value

## S3 method for class 'dictionary'
print(x, force = FALSE, ...)
```

Arguments

`...`	Arbitrary objects. Should be tagged, yet currently name tags are not yet checked for.
`x`	A dictionary object
`i`	A scalar integer or character string
`value`	An arbitrary object
`force`	A logical scalar, if TRUE, each element of the dictionary is printed, if FALSE, just a brief summary is printed.

Methods (by generic)

- `[`: Get an element from a dictionary
- `` `[`(dictionary) <- value ``: Set an element in a dictionary
- `print(dictionary)`: Print a dictionary

Functions

- `dictionary()`: A dictionary constructor

display *Display an R Object*

Description

Sends a 'display_data' message to the frontend. Allows users to create rich display of R objects.

Usage

```
display(...)
```

Arguments

...	Arguments passed to 'display_data' methods

display_data *Prepare an R Object for Being Displayed*

Description

A generic function that prepares R objects for display using `display()`

Usage

```
display_data(x, ...)

## Default S3 method:
display_data(
  x,
  ...,
  data,
  metadata = emptyNamedList,
  id = UUIDgenerate(),
  update = FALSE
)

## S3 method for class 'htmlwidget'
display_data(
  x,
  ...,
  metadata = emptyNamedList,
  id = UUIDgenerate(),
  update = FALSE
)
```

```
## S3 method for class 'recordedplot'
display_data(
  x,
  width = getOption("jupyter.plot.width", 7),
  height = getOption("jupyter.plot.height", 7),
  resolution = getOption("jupyter.plot.res", 144),
  id = UUIDgenerate(),
  update = FALSE,
  ...
)

## S3 method for class 'display_data'
display_data(x, ...)

## S3 method for class 'update_display_data'
display_data(x, ...)

## S3 method for class 'display_data'
update(object, ...)

## S3 method for class 'data.frame'
display_data(
  x,
  ...,
  metadata = emptyNamedList,
  id = UUIDgenerate(),
  update = FALSE
)

## S3 method for class 'matrix'
display_data(
  x,
  ...,
  metadata = emptyNamedList,
  id = UUIDgenerate(),
  update = FALSE
)

## S3 method for class 'html_elem'
display_data(
  x,
  ...,
  metadata = emptyNamedList,
  id = UUIDgenerate(),
  update = FALSE
)

## S3 method for class 'shiny.tag'
```

```
display_data(
  x,
  ...,
  metadata = emptyNamedList,
  id = UUIDgenerate(),
  update = FALSE
)

## S3 method for class 'iframe'
display_data(
  x,
  ...,
  metadata = emptyNamedList,
  id = attr(x, "id"),
  update = FALSE
)

## S3 method for class 'help_files_with_topic'
display_data(
  x,
  ...,
  id = UUIDgenerate(),
  update = FALSE,
  embedded = FALSE,
  include_button = TRUE
)

## S3 method for class 'hsearch'
display_data(x, ..., id = UUIDgenerate(), update = FALSE)

## S3 method for class 'Widget'
display_data(x, ..., metadata = emptyNamedList, id = uuid(), update = FALSE)
```

Arguments

x	An object
...	Optional arguments tagged by mime types with mime data
data	A list with named elements, containing mime data
metadata	A list with named elements, containing metadata
id	An identifier string
update	A logical value, whether a new display item should be created or an existing one should be updated
width	Width of the diplayed plot
height	Height of the displayed plot
resolution	Resolution in ppi, see png
object	An object of class "display_data"

pointsize	Point size, see png
scale	The amount by which the plots are scaled in the frontend
units	Units of width and height, see png

Value

An object of class "display_data"

Methods (by class)

- display_data(default): Default method
- display_data(htmlwidget): S3 method for html widgets
- display_data(recordedplot): S3 method for plots saved with 'recordPlot()'
- display_data(display_data): S3 method for "display_data" objects
- display_data(update_display_data): S3 method for "update_display_data" objects
- display_data(data.frame): S3 method for class 'data.frame'
- display_data(matrix): S3 method for matrices
- display_data(html_elem): S3 method for "html_elem" objects (see html)
- display_data(shiny.tag): S3 methods for "shiny" objects
- display_data(iframe): S3 methods for "iframe" objects
- display_data(help_files_with_topic): S3 method for help pages
- display_data(hsearch): S3 method for results of 'help.search()'
- display_data(Widget): Method for jupyter widgets

Methods (by generic)

- update(display_data): "update" method for "display_data" objects

---

display_id *Get the id of an object display*

Description

This function returns the id of an object created by display_data() or update_display_data().

Usage

```
display_id(x)

## S3 method for class 'display_data'
display_id(x)

## S3 method for class 'update_display_data'
display_id(x)
```

Arguments

x	An object of class "display_data" or "update_display_data"

Value

a character string with the id.

Methods (by class)

- `display_id(display_data)`: S3 method for "display_data" objects
- `display_id(update_display_data)`: S3 method for "update_display_data" objects

DOMWidgetClass *A Base Class for DOM Widgets*

Description

This is a base class for all widgets that are supposed to be part of the document object model

Usage

```
DOMWidget(...)
```

Arguments

...	Arguments passed to the inializer

Functions

- `DOMWidget()`: The DOM widget constructor function

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `DOMWidget`

Public fields

`_model_module` Name of the Javascript module with the model

`_model_module_version` Version of the module where the model is defined

`_model_name` Name of the Javascript model in the frontend

`_dom_classes` A set of character strings that indicate the DOM classes the widget is assigned to

`layout` The layout, a "LayoutClass" Widget

Methods

Public methods:

- DOMWidgetClass$add_class()
- DOMWidgetClass$remove_class()
- DOMWidgetClass$has_class()
- DOMWidgetClass$clone()

Method `add_class()`**:** Add a class attribute to the DOM element

Usage:

```
DOMWidgetClass$add_class(className)
```

Arguments:

`className` Name of the class attribute

Method `remove_class()`**:** Remove a class attribute to the DOM element

Usage:

```
DOMWidgetClass$remove_class(className)
```

Arguments:

`className` Name of the class attribute

Method `has_class()`**:** Check whether the DOM element has a class attribute

Usage:

```
DOMWidgetClass$has_class(className)
```

Arguments:

`className` Name of the class attribute

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`envBrowser` *Environment Browser Widgets*

Description

This function returns a widget that allows to browse within an environment

Usage

```
envBrowser(
  pos = -1,
  name = NULL,
  envir,
  parent = NULL,
  all.names = FALSE,
  pattern = NULL,
  mode = "any"
)
```

Arguments

pos	integer indicating the search position, or -1 for the current environment.
name	optional name indicating a position in the search path, see ls.
envir	environment to use, see ls.
parent	an optional parent environment.
all.names	logical; if true names starting with '.' are not omitted, see ls.
pattern	an optional pattern of names to restrict browsing to, see ls.
mode	an optional string indicating the mode of objects to which browsing is restricted, see ls.

EventManager *A Manager for Comms*

Description

Objects of this class are used internally to manage events, they are not meant to be used by end-users.

Usage

```
EventManager(...)
```

Arguments

...	Arguments passed to the inializer

Functions

- `EventManager()`: The constructor function, returns an Object of Class "EventManagerClass"

Methods

Public methods:

- EventManagerClass$new()
- EventManagerClass$send()
- EventManagerClass$on()
- EventManagerClass$activate()
- EventManagerClass$suspend()
- EventManagerClass$clear()
- EventManagerClass$resume()
- EventManagerClass$has()
- EventManagerClass$clone()

Method `new()`**:** Initialize an event manager

Usage:

```
EventManagerClass$new(type)
```

Arguments:

`type` A string, the type of event (e.g. "print")

Method `send()`**:** Send an event

Usage:

```
EventManagerClass$send(event, ...)
```

Arguments:

`event` A string, the name of an event

`...` Other arguments, sent to the event handler(s)

Method `on()`**:** Install a handler for an event

Usage:

```
EventManagerClass$on(event, handler, remove = FALSE)
```

Arguments:

`event` A string, the name of an event

`handler` A function

`remove` A logical value, whether the handler is to be removed

Method `activate()`**:** Activate handlers

Usage:

```
EventManagerClass$activate(event = NULL, all = TRUE)
```

Arguments:

`event` A string, the name of an event, ignored if 'all' is TRUE.

`all` A logical value, if TRUE, all handlers that belong to the event type of the event manager are activated.

Method `suspend()`**:** Suspend handlers

Usage:

```
EventManagerClass$suspend(event = NULL, all = TRUE)
```

Arguments:

`event` A string, the name of an event, ignored if 'all' is TRUE.

`all` A logical value, if TRUE, all handlers that belong to the event type of the event manager are suspended.

Method `clear()`**:** Clear (i.e. remove) handlers for an event

Usage:

```
EventManagerClass$clear(event)
```

Arguments:

`event` A string, the name of an event

Method `resume()`**:** Resume (i.e. reactivate) an event handler

Usage:

```
EventManagerClass$resume()
```

Method `has()`**:** Check whether the event manager has handlers for the given events

Usage:

```
EventManagerClass$has(events)
```

Arguments:

`events` A character vector with names of events

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`FileUpload` *File Upload Widgets*

Description

Class and constructor for file upload widgets

Usage

```
FileUpload(...)
```

Arguments

`...`	Any arguments used to initialize the fields of the object

Functions

- `FileUpload()`: The FileUpload constructor function

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `FileUpload`

Public fields

`_model_name` Name of the Javascript model in the frontend.

`_view_name` Name of the Javascript view in the frontend.

`accept` A character string that defineds the accepted file type. If empty, all files are accepted.

`multiple` A Boolean traitlet, whether multiple files are accepted.

`disabled` A Boolean traitlet, whether the widget is disabled.

`icon` A character string, the font-awesome without the 'fa-' prefix.

`button_style` The string that describes the button style

`style` The button style, an object of class "ButtonStyleClass".

`error` A string with an error message, if applicable.

`value` The uploaded data.

`names` of the uploaded files

Active bindings

`names` of the uploaded files

Methods

Public methods:

- `FileUploadClass$new()`
- `FileUploadClass$handle_buffers()`
- `FileUploadClass$clone()`

Method `new()`**:** A generic initializer function

Usage:

```
FileUploadClass$new(description = "Upload", ...)
```

Arguments:

`description` The button description

`...` Any arguments used to initialize the fields of the object

Method `handle_buffers()`**:** Handle buffers in message

Usage:

```
FileUploadClass$handle_buffers(msg)
```

Arguments:

`msg` A comm message

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Fixed` *Fixed Arguements in Interactive Widgets*

Description

The function 'Fixed' returns ist argument marked with a class attribute "Fixed", so that it is not made into a widget when passed to `Interactive`

Usage

```
Fixed(x)
```

Arguments

`x`	An object.

`Float` *Floating Point Number Traitlets*

Description

A class and a constructor function to create floating point vector trait(let)s.

Usage

```
Float(...)

## S3 method for class 'Float'
as.integer(x, ...)

## S3 method for class 'Float'
as.numeric(x, ...)
```

Arguments

`...`	Other arguments.
`x`	A floating point traitlet.

Super class

`RKernel::Trait` -> `Float`

Public fields

`value` A numeric vector.

`optional` Logical value, whether a length-zero value is allowed.

`coerce` Logical value, whether assignments to the value field should be coerced to the appropriate type.

`length` Integer number, the length the value should have.

Methods

Public methods:

- `FloatClass$validator()`
- `FloatClass$new()`
- `FloatClass$clone()`

Method `validator()`**:** Check the value assigned to the traitlet.

Usage:

```
FloatClass$validator(value)
```

Arguments:

`value` The value assigned to the traitlet.

Method `new()`**:** Initialize the traitlet.

Usage:

```
FloatClass$new(
  initial = numeric(0),
  coerce = TRUE,
  optional = length(initial) == 0,
  length = 1L
)
```

Arguments:

`initial` A numeric vector, the initial value for the traitlet.

`coerce` coerce Logical value, whether assignments to the value field should be coerced to the appropriate type.

`optional` Logical value, whether a length-zero value is allowed.

`length` Integer number, the length the value should have.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

FloatText *Widgets for Text Elements with Floating Point Numbers*

Description

An R6 class and a constructor function for the creation of widgets that can be used to manipulate floating point numbers.

Usage

```
FloatText(value = 0, step = 0.1, ...)
```

Arguments

value	Initial value of the floating point number.
step	Increment by which the number is increased or decreased by the text field controls.
...	Other arguments.

Details

The function FloatText creates objects of the R6 Class "FloatTextClass", which in turn have the S3 class attribute "FloatText".

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::FloatWidget -> FloatText

Public fields

_model_name Name of the Javascript model in the frontend.

_view_name Name of the Javascript view in the frontend.

disabled A Boolean traitlet, whether the text widget is disabled.

continuous_update A Boolean traitlet, whether the text widget is continuously updated upon change in the frontend.

step A Float traitlet, a step size by which the value is incremented or decremented if the arrows are clicked.

Methods

Public methods:

- FloatTextClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`FloatWidget` *Widgets for Floating Point Numbers*

Description

An R6 class and a constructor function for the creation of widgets that can be used to manipulate floating point numbers

Usage

```
FloatWidget(value, ...)
```

Arguments

`value`	The floating point value
`...`	Other arguments, passed to the superclass initializer

Details

The function `FloatWidget` creates objects of the R6 Class "FloatWidgetClass", which in turn have the S3 class attribute "FloatWidget"

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `FloatWidget`

Public fields

`value` A Float traitlet

Methods

Public methods:

- `FloatWidgetClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

HasTraits *The Base Class of Objects with Traits*

Description

Objects in class HasTraits have traits as components that are correctly initialized using delayed construction with the `TraitInstance` function.

Public fields

`traits` A list of traits

`suspended` Logical value; whether notifying observers is suspended.

`observers` A list of observers, i.e. callback functions called by the `notify` method.

Methods

Public methods:

- `HasTraits$new()`
- `HasTraits$notify()`
- `HasTraits$observe()`
- `HasTraits$validate()`
- `HasTraits$clone()`

Method `new()`**:** Initialize an object

Usage:

```
HasTraits$new(...)
```

Arguments:

`...` Initializsing values

Method `notify()`**:** Notify observers about a trait being set to a value.

Usage:

```
HasTraits$notify(tn, value)
```

Arguments:

`tn` A string, the name of the trait.

`value` The value to which the trait is set.

Method `observe()`**:** Install or remove an observer function.

Usage:

```
HasTraits$observe(tn, observer, remove = FALSE)
```

Arguments:

`tn` A string, the name of a trait.

`observer` A callback function, which should take three arguments, (1) the trait name, (2) the object that has the trait, (3)

`remove` A logical value, indicates whether the observer is to be removed or added

Method `validate()`**:** Install or remove the validator function of a trait.

Usage:

```
HasTraits$validate(tn, validator, remove = FALSE)
```

Arguments:

`tn` A string, the name of a trait.

`validator` A callback function

`remove` A logical value, indicates whether the validator is to be removed or added

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`help.start` *Start interactive help system*

Description

A variant of `help.start` that works when called from inside a Jupyter notebook.

Usage

```
help.start(
  update = FALSE,
  gui = "irrelevant",
  browser = getOption("browser"),
  remote = NULL
)
```

Arguments

`update`	A logical value. This formal argument exists for compatibility reasons only.
`gui`	A character string. This formal argument exists for compatibility reasons only.
`browser`	A character string. This formal argument exists for compatibility reasons only.
`remote`	A character string. This formal argument exists for compatibility reasons only.

IFrame *Include HTML content using in an iframe*

Description

Display the contents of a webpage or other HTML content by including output using an [iframe](https://html.spec.whatwg.org
embed-object.html).

Usage

```
IFrame(url, width = "100%", height = "70ex", class = NULL, srcdoc = FALSE)
```

Arguments

url	A character string, the URL of the content to be included
width	A character string that specifies the width of the iframe
height	A character string that specifies the width of the iframe
class	An optional character string with DOM classes to be assigned to the iframe.
srcdoc	Logical, whether to use a 'src' (FALSE, the default) or 'srcdoc' attribute.

install *install the R Kernel*

Description

install the R Kernel

Usage

```
install()

installspec(user = TRUE, prefix = NULL, single_blas = FALSE)
```

Arguments

user	Logical, whether to install the kernel in the user's home directory
prefix	NULL or a character string with a path prefix

Functions

- `installspec()`: Install the R Kernel spec

`Integer` *Integer Traitlets*

Description

A class and a constructor function to create integer vector trait(let)s.

Usage

```
Integer(...)

## S3 method for class 'Integer'
as.integer(x, ...)

## S3 method for class 'Integer'
as.numeric(x, ...)

## S3 method for class 'Integer'
to_json(x, ...)
```

Arguments

`...`	Other arguments.
`x`	An integer traitlet.

Super class

`RKernel::Trait` -> `Integer`

Public fields

`value` An integer vector.

`optional` Logical value, whether a length-zero value is allowed.

`coerce` Logical value, whether assignments to the value field should be coerced to the appropriate type.

`length` Integer number, the length the value should have.

Methods

Public methods:

- `IntegerClass$validator()`
- `IntegerClass$new()`
- `IntegerClass$clone()`

Method `validator()`**:** Check the value assigned to the traitlet.

Usage:

```
IntegerClass$validator(value)
```

Arguments:

`value` The value assigned to the traitlet.

Method `new()`**:** Initialize the traitlet.

Usage:

```
IntegerClass$new(
  initial = integer(0),
  coerce = TRUE,
  optional = length(initial) == 0,
  length = 1L
)
```

Arguments:

`initial` An integer vector, the initial value for the traitlet.

`coerce` coerce Logical value, whether assignments to the value field should be coerced to the appropriate type.

`optional` Logical value, whether a length-zero value is allowed.

`length` Integer number, the length the value should have.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`interaction` *Interactions Using Widgets*

Description

A variety of functions to create interactive function calls

Usage

```
interactive_output(
  FUN,
  controls,
  out,
  button = NULL,
  continuous_update = TRUE,
  autorun = TRUE,
  clear = FALSE,
  mime_type = "text/plain"
)
```

```
mkWidgets(...)

Interactive(
  FUN,
  ...,
  continuous_update = TRUE,
  append_output = FALSE,
  use_display = TRUE
)

interact(
  FUN,
  ...,
  continuous_update = TRUE,
  append_output = FALSE,
  use_display = TRUE
)
```

Arguments

FUN	A function to called with arguments manipulated using interactive widgets.
controls	A list of controlling widgets, usually created with the function mkwidgets
out	An output widget, i.e. a widget in class "OutputWidget"
button	An (optional) button widget; when clicked, the function FUN is called.
continuous_update	A logical value, if TRUE the function FUN is called whenever one of the controlling widgets changes a value
autorun	Logical, whether the function FUN will be automatically called when any of the controlling widget values is changed or only when button is clicked.
clear	Logical, whether out is cleared before each call of FUN.
mime_type	A character string that specifies the mime type as which the return value of FUN is displayed.
...	Named arguments, transformed into widgets using the generic function mkWidget.
append_output	Logical, whether existing output should be appended to or overwritten.
use_display	Logical, whether the display mechanism is used internally for output streams.

IntText *Widgets for Text Elements with Integer Numbers*

Description

An R6 class and a constructor function for the creation of widgets that can be used to manipulate integer numbers

Usage

```
IntText(value = 0, step = 1, ...)
```

Arguments

value	Initial value of the integer number
step	Increment by which the number is increased or decreased by the text field controls
...	Arguments passed to the superclass constructor

Details

The function IntText creates objects of the R6 Class "IntTextClass", which in turn have the S3 class attribute "IntText"

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::IntWidget -> IntText

Public fields

_model_name Name of the Javascript model in the frontend.

_view_name Name of the Javascript view in the frontend.

disabled A Boolean traitlet, whether the text widget is disabled.

continuous_update A Boolean traitlet, whether the text widget is continuously updated upon change in the frontend.

step An Integer traitlet, a step size by which the value is incremented or decremented if the arrows are clicked.

Methods

Public methods:

- IntTextClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

IntWidget *Widgets for Integer Numbers*

Description

An R6 class and a constructor function for the creation of widgets that can be used to manipulate integer numbers.

Usage

```
IntWidget(value, ...)
```

Arguments

value	The integer value.
...	Other arguments, passed to the superclass initializer.

Details

The function IntWidget creates objects of the R6 Class "IntWidgetClass", which in turn have the S3 class attribute "IntWidget".

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> IntWidget

Public fields

value A Integer traitlet.

Methods

Public methods:

- IntWidgetClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Javascript *Send Javascript to the frontend*

Description

Send Javascript code in a character string or a text file to the frontend.

Usage

```
Javascript(text, file, as_tag = FALSE)
```

Arguments

text	A character string with Javascript code
file	Path of a file with Javascript code
as_tag	Logical, whether to return a '<script>' tag

Value

An S3 object of class "display_data" with mime data of type "application/javascript"

Kernel *The Kernel Class*

Description

An object of this class handles the low-level communication with the Jupyter frontend or kernel manager. There should only be one object of this class in existence.

Public fields

`r_session` See `RKernelSession`.

`DAPServer` The current DAP server

Methods

Public methods:

- `Kernel$new()`
- `Kernel$start_r_session()`
- `Kernel$start()`
- `Kernel$run()`
- `Kernel$poll_and_respond()`
- `Kernel$clear_output()`
- `Kernel$stream()`

- Kernel$stdout()
- Kernel$stderr()
- Kernel$r_session_msg()
- Kernel$execute_result()
- Kernel$display_send()
- Kernel$send_error()
- Kernel$send_comm()
- Kernel$get_parent()
- Kernel$get_conn_info()
- Kernel$is_child()
- Kernel$input_request()
- Kernel$read_stdin()
- Kernel$send_debug_event()
- Kernel$clone()

Method `new()`**:** Initialize the kernel

Usage:

```
Kernel$new(conn_info)
```

Arguments:

`conn_info` A list with the connection info from the front-end

Method `start_r_session()`**:**

Usage:

```
Kernel$start_r_session()
```

Method `start()`**:**

Usage:

```
Kernel$start()
```

Method `run()`**:** Run the kernel.

Usage:

```
Kernel$run()
```

Method `poll_and_respond()`**:** A single iteration of the kernel loop

Usage:

```
Kernel$poll_and_respond()
```

Method `clear_output()`**:** Clear the current output cell in the frontend.

Usage:

```
Kernel$clear_output(wait)
```

Arguments:

`wait` Logical value, whether to wait until output is cleared.

Method `stream()`**:** Stream text to the frontend.

Usage:

```
Kernel$stream(text, stream)
```

Arguments:

`text` Text to be sent to the frontend

`stream` A string to select the stream – either "stout" or "stderr"

Method `stdout()`**:** Stream text to the frontend via 'stdout' stream.

Usage:

```
Kernel$stdout(text)
```

Arguments:

`text` Text to be sent to the frontend

Method `stderr()`**:** Stream text to the frontend via 'stderr' stream.

Usage:

```
Kernel$stderr(text)
```

Arguments:

`text` Text to be sent to the frontend

Method `r_session_msg()`**:** Stream message created from R session process to the frontend via 'stderr' stream.

Usage:

```
Kernel$r_session_msg(msg)
```

Arguments:

`msg` The message created by the 'RKernelSession' object.

Method `execute_result()`**:** Send execution results to the frontend

Usage:

```
Kernel$execute_result(data, metadata = emptyNamedList)
```

Arguments:

`data` Execution result in rich format

`metadata` A list with metadata

Method `display_send()`**:** Send rich format data to the frontend

Usage:

```
Kernel$display_send(msg)
```

Arguments:

`msg` A list with the appropriate structure. [TODO]

Method `send_error()`**:** Send an error message and traceback to the frontend.

Usage:

```
Kernel$send_error(name, value, traceback)
```

Arguments:

`name` A string, the error name.
`value` A string, the value of the error message.
`traceback` A character vector with the traceback.

Method `send_comm()`**:** Send a message via a comm.

Usage:

```
Kernel$send_comm(msg)
```

Arguments:

`msg` A list containing a comm message.

Method `get_parent()`**:** The parent of the message currently sent.

Usage:

```
Kernel$get_parent(channel = "shell")
```

Arguments:

`channel` A string, the relevant input channel.

Method `get_conn_info()`**:** Return the current connection info.

Usage:

```
Kernel$get_conn_info()
```

Method `is_child()`**:** Check if the current process is a fork from the original kernel process

Usage:

```
Kernel$is_child()
```

Method `input_request()`**:** Send an input request to the frontend

Usage:

```
Kernel$input_request(prompt = "", password = FALSE)
```

Arguments:

`prompt` A prompt string
`password` Logical value; whether the input should be hidden like in a password dialog

Method `read_stdin()`**:** Read a line from the frontend

Usage:

```
Kernel$read_stdin()
```

Method `send_debug_event()`**:** Send a debug event to the frontend

Usage:

```
Kernel$send_debug_event(content)
```

Arguments:

`content` A list, content provided by the debug adapter

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

LaTeXMath *Send LaTeX math to the frontend*

Description

Send LaTeX code for math in a character string to the frontend to be formatted by MathJax

Usage

```
LaTeXMath(text)
```

Arguments

text	A character string with LaTeX code for math

Layout *Widget Layout Manipulation*

Description

An R6 class and a constructor function for the creation of a layout widget, which itself is used to manipulate the layout of a DOMWidget.

Usage

```
Layout(...)
```

Arguments

...	Arguments passed to the inializer

Details

The function Layout creates objects of the R6 Class "LayoutClass", which in turn have the S3 class attribute "Layout"

Functions

- Layout(): The Layout constructor function

Super classes

RKernel::HasTraits -> RKernel::Widget -> Layout

Public fields

`_view_name` Name of the Javascript view in the frontend.

`_view_module` Name of the Javascript view module in the frontend.

`_view_module_version` Version of the Javascript view module in the frontend.

`_model_name` Name of the Javascript model in the frontend.

`align_content` An optional string, if non-empty, one of "flex-start", "flex-end", "center", "space-between", "space-around", "space-evenly", "stretch"

`align_items` An optional string, if non-empty, one of "flex-start", "flex-end", "center", "baseline", "stretch"

`align_self` An optional string, if non-empty, one of "flex-start", "flex-end", "center", "baseline", "stretch"

`bottom` Position from bottom, an optional string that should, if non-empty, contain a valid CSS dimension

`border` An optional string with a valid CSS border specification

`border_top` An optional string with a valid CSS border specification

`border_right` An optional string with a valid CSS border specification

`border_bottom` An optional string with a valid CSS border specification

`border_left` An optional string with a valid CSS border specification

`display` An optional string with a valid CSS display property

`flex` An optional string with a valid CSS flex property

`flex_flow` An optional string with a valid CSS flex_flow property

`height` An optional string with a valid CSS height

`justify_content` An optional string, if non-empty, one of "flex-start", "flex-end", "center", "space-between", "space-around".

`justify_items` An optional string, if non-empty, one of "flex-start", "flex-end", or "center"

`left` Position from left, an optional string that should, if non-empty, contain a valid CSS dimension

`margin` An optional string, if non-empty, should be a valid CSS margin specification

`max_height` An optional string, if non-emtpy, should be a valid CSS dimension

`max_width` An optional string, if non-emtpy, should be a valid CSS dimension

`min_height` An optional string, if non-emtpy, should be a valid CSS dimension

`min_width` An optional string, if non-emtpy, should be a valid CSS dimension

`overflow` An optonal string, if non-empty, should be a valid CSS overflow specification

`order` An optional string, if non-empty should contain a number

`padding` An optional string, if non-emtpy should be a valid CSS dimension

`right` Position from right, an optional string, if non-empty, should be a valid CSS dimension

`top` Position from top, an optional string, if non-empty, should be a valid CSS dimension

`visibility` An optional string, if non-empty, should be either "visible" or "hidden"

`width` An optional string, if non-empty, should be a valid CSS dimension

`object_fit` An optional string, if non-empty, should be one of "contain", "cover", "fill", "scale-down", "none"

`object_position` An optional string, if non-empty, should be a valid CSS object-position specification

`grid_auto_columns` An optional string, if non-empty should be valid CSS code for the grid-auto-columns property

`grid_auto_flow` An optional string, if non-empty should be valid CSS code for the grid-auto-flow property

`grid_auto_rows` An optional string, if non-empty should be valid CSS code for the grid-auto-rows property

`grid_gap` An optional string, if non-empty should be valid CSS code for the grid-gap property

`grid_template_rows` An optional string, if non-empty should be valid CSS code for the grid-template-rows property

`grid_template_columns` An optional string, if non-empty should be valid CSS code for the grid-template-columns property

`grid_template_areas` An optional string, if non-empty should be valid CSS code for the grid-template-areas property

`grid_row` An optional string, if non-empty should be valid CSS code for the grid-row property

`grid_column` An optional string, if non-empty should be valid CSS code for the grid-column property

`grid_area` An optional string, if non-empty should be valid CSS code for the grid-area property

Methods

Public methods:

- `LayoutClass$observe_border()`
- `LayoutClass$new()`
- `LayoutClass$clone()`

Method `observe_border()`**:** Synchronize border traits

Usage:

```
LayoutClass$observe_border(nm, self, value)
```

Arguments:

`nm` Name of the trait (a dummy argument)

`self` The object

`value` A CSS string

Method `new()`**:** Initialize an object

Usage:

```
LayoutClass$new(...)
```

Arguments:

`...` Arguments passed to the superclass initializer

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`LayoutTemplates` *Widget Layout Templates*

Description

R6 classes and constructor functions for widget layout templates

Usage

```
AppLayout(...)

GridspecLayout(...)

## S3 method for class 'GridspecLayout'
x[i, j, ..., drop = TRUE]

## S3 replacement method for class 'GridspecLayout'
x[i, j] <- value

TwoByTwoLayout(...)
```

Arguments

`...`	Arguments used to initialize the fields
`x`	A GridspecLayout object
`i`	Integer value(s) referring to the row(s)
`j`	Integer value(s) referring to the column(s)
`drop`	Logical, whether the result is a widget or a list with one element if both i an j select a single element
`value`	One or more widgets put at the idicated positions in the grid

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::Box` -> `RKernel::GridBox` -> `TemplateBase`

Public fields

`grid_gap` The grid-gap CSS attribute

`justify_content` The justify-content CSS attribute

`align_items` The align-items CSS attribute

`width` The width CSS attribute

`height` The height CSS attribute

Methods

Public methods:

- `TemplateBaseClass$new()`
- `TemplateBaseClass$clone()`

Method `new()`**:** Initializer

Usage:

```
TemplateBaseClass$new(...)
```

Arguments:

`...` Arguments used to initialize the fields

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::Box` -> `RKernel::GridBox` -> `RKernel::TemplateBase` -> `AppLayout`

Public fields

`header` Widget that appears in the header section

`footer` Widget that appears in the footer section

`left_sidebar` Widget that appears as left sidebar

`right_sidebar` Widget that appears as right sidebar

`center` Widget that appears in the center section

`pane_widths` Unicode string with CSS widths for the app panes

`pane_heights` Unicode string with CSS heights for the app panes

`merge` Boolean, whether space of missing widgets chould be merged

Methods

Public methods:

- AppLayoutClass$new()
- AppLayoutClass$clone()

Method `new()`**:** Initializer method

Usage:

```
AppLayoutClass$new(...)
```

Arguments:

`...` Arguments, passed on to the superclass initializer

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::Box -> RKernel::GridBox -> RKernel::TemplateBase -> GridspecLayout

Methods

Public methods:

- GridspecLayoutClass$new()
- GridspecLayoutClass$set_item()
- GridspecLayoutClass$get_item()
- GridspecLayoutClass$clone()

Method `new()`**:** Initializer function

Usage:

```
GridspecLayoutClass$new(nrow, ncol, ...)
```

Arguments:

`nrow` A positive integer, the number of rows

`ncol` A positive integer, the number of columns

`...` Other arguments, passed to the superclass initializer

Method `set_item()`**:** Set widget in grid cells

Usage:

```
GridspecLayoutClass$set_item(i, j, value)
```

Arguments:

`i` The rows into which the widget is to be placed

`j` The columns into which the widget is to be placed
`value` A widget

Method `get_item()`**:** Get widget from grid cells

Usage:

```
GridspecLayoutClass$get_item(i, j)
```

Arguments:

`i` The rows where the widget is located
`j` The columns where the widget is located

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::Box` -> `RKernel::GridBox` -> `RKernel::TemplateBase` -> `TwoByTwoLayout`

Public fields

`top_left` Widget that appears on the top left
`top_right` Widget that appears on the top right
`bottom_left` Widget that appears on the bottom left
`bottom_right` Widget that appears on the bottom right
`merge` Boolean, whether space of missing widgets chould be merged

Methods

Public methods:

- `TwoByTwoLayoutClass$new()`
- `TwoByTwoLayoutClass$clone()`

Method `new()`**:** Initializer method

Usage:

```
TwoByTwoLayoutClass$new(...)
```

Arguments:

`...` Arguments, passed on to the superclass initializer

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`List` *List Traitlets*

Description

A class and a constructor function to create list trait(let)s.

Usage

```
List(...)
```

Arguments

`...`	Arguments that are passed to the initialize method of 'ListClass'

Super class

`RKernel::Trait` -> `List`

Public fields

`value` A list

Methods

Public methods:

- `ListClass$validator()`
- `ListClass$clone()`

Method `validator()`**:** A function that checks the validity of an assigned value, i.e. whether the assigned value is a list

Usage:

```
ListClass$validator(value)
```

Arguments:

`value` A value to be assigned as the traitlet value

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

ls_str *A HTML version of 'ls.str()'*

Description

This function is deprecated. Use `envBrowser` instead.

Usage

```
ls_str(pos = -1, name, envir, all.names = FALSE, pattern, mode = "any")
```

Arguments

pos	integer indicating `search` path position, or -1 for the current environment.
name	an optional name indicating search path position, see `ls`.
envir	the environment to look into
all.names	logical value, if FALSE objects with names that start with a dot are ignored
pattern	a character string, the pattern of the names of the objects to show
mode	a character string, the mode of the objects to be shown

See Also

`ls.str`

main *Main entry point of the package*

Description

This function reads the connection info file, creases a "Kernel" object and runs it, i.e. starts the kernel.

Usage

```
main()
```

MediaWidget *Media widgets*

Description

Classes and constructors to wrap media into widgets

Usage

```
ImageWidget(...)

VideoWidget(...)

AudioWidget(...)
```

Arguments

...	Any arguments used to initialize the fields of the object

Functions

- `ImageWidget()`: The ImageWidget constructor function
- `VideoWidget()`: The VideoWidget constructor function
- `AudioWidget()`: The AudioWidget constructor function

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `MediaWidget`

Public fields

`format` A string, giving the graphics fromat.

`value` A Bytes traitlet.

Methods

Public methods:

- `MediaWidgetClass$from_url()`
- `MediaWidgetClass$from_file()`
- `MediaWidgetClass$on_change()`
- `MediaWidgetClass$new()`
- `MediaWidgetClass$clone()`

Method `from_url()`**:** Create media widget from url

Usage:

```
MediaWidgetClass$from_url(url, width = NULL, height = NULL)
```

Arguments:

`url` A character string

`width` A character string with CSS width specification

`height` A character string with CSS height specification

Method `from_file()`**:** Create media widget from file

Usage:

```
MediaWidgetClass$from_file(filename)
```

Arguments:

`filename` A character string

Method `on_change()`**:** Add or remove a handler to be called if value is changed.

Usage:

```
MediaWidgetClass$on_change(handler, remove = FALSE)
```

Arguments:

`handler` A function that is called when the button is clicked.

`remove` Logical value, whether the handler is to be removed.

Method `new()`**:** Initialize an object

Usage:

```
MediaWidgetClass$new(from_file = NULL, from_url = NULL, ...)
```

Arguments:

`from_file` An optional character string, name of the file from which to initialize the widget.

`from_url` An optional character string, URL from which to initialize the widget.

`...` Other arguments, passed to the superclass initializer.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::MediaWidget` -> `ImageWidget`

Public fields

`_view_name` Name of the Javascript view in the frontend.

`_model_name` Name of the Javascript model in the frontend.

`format` A string, giving the graphics fromat.

`width` A string, describing the width in CSS language, e.g. "480px".

`height` A string, describing the height in CSS language, e.g. "480px".

Methods

Public methods:

- ImageWidgetClass$from_file()
- ImageWidgetClass$clone()

Method `from_file()`**:** Create image widget from file

Usage:

```
ImageWidgetClass$from_file(filename, width = NULL, height = NULL)
```

Arguments:

`filename` A character string

`width` A character string with CSS width specification

`height` A character string with CSS height specification

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::MediaWidget -> VideoWidget

Public fields

`_view_name` Name of the Javascript view in the frontend.

`_model_name` Name of the Javascript model in the frontend.

`format` A string, giving the video fromat.

`width` A string, describing the width in CSS language, e.g. "480px".

`height` A string, describing the height in CSS language, e.g. "480px".

`autoplay` Boolean, when TRUE the video starts when it is displayed.

`loop` Boolean, when TRUE the video restarts after finishing.

`controls` Boolean, when TRUE then video controls are shown.

Methods

Public methods:

- VideoWidgetClass$from_file()
- VideoWidgetClass$clone()

Method `from_file()`**:** Create image widget from file

Usage:

```
VideoWidgetClass$from_file(filename, width = NULL, height = NULL)
```

Arguments:

`filename` A character string

`width` A character string with CSS width specification

`height` A character string with CSS height specification

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::MediaWidget` -> `AudioWidget`

Public fields

`_view_name` Name of the Javascript view in the frontend.

`_model_name` Name of the Javascript model in the frontend.

`format` A string, giving the audio fromat.

`autoplay` Boolean, when TRUE the video starts when it is displayed.

`loop` Boolean, when TRUE the video restarts after finishing.

`controls` Boolean, when TRUE then video controls are shown.

Methods

Public methods:

- `AudioWidgetClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

mkWidget *Controls for Interactive Widgets*

Description

A set of functions that can be used to create interactive widgets and to interact with widgets.

Usage

```
mkWidget(x, ...)

## S3 method for class 'integer'
mkWidget(x, description = NULL, ...)

## S3 method for class 'numeric'
mkWidget(x, description = NULL, ...)

## S3 method for class 'logical'
mkWidget(x, description = NULL, ...)

## S3 method for class 'character'
mkWidget(x, description = NULL, ...)

## S3 method for class 'Fixed'
mkWidget(x, ...)

## S3 method for class 'ValueWidget'
mkWidget(x, ...)
```

Arguments

x	an object
...	Other arguments, passed to more specific methods or ignored
description	NULL or a character string that contains a description.

Details

The function mkWidget is a generic function that creates a widget that allows to manipulate the arguments of a function that is called in an interactive widget. This generic function is called by the function mkWidgets. The function Fixed marks a value as fixed, so that mkWidget returns it as is.

OutputWidget *Widgets to receive output*

Description

Classes and constructors to wrap output created by code

Usage

```
OutputWidget(append_output = FALSE, ...)

## S3 method for class 'OutputWidget'
with(data, expr, envir = list(), enclos = parent.frame(), clear = TRUE, ...)
```

Arguments

append_output	Logical value, whether new output is appended to existing output in the widget or the output is overwritten
...	Other arguments, ignored.
data	An "OutputWidget" object
expr	An expression to evaluate, or a sequence of expression, encapsulated by curly braces.
enclos	An enclosing environment.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> OutputWidget

Public fields

_view_name Name of the Javascript model view in the frontend

_model_name Name of the Javascript model in the frontend

_view_module Name of the module where the view is defined

_model_module Name of the Javascript module with the model

_view_module_version Version of the module where the view is defined

_model_module_version Version of the module where the model is defined

msg_id Unicode string with the id of the last message sent to the frontend.

outputs A list with output strings

Methods

Public methods:

- OutputWidgetClass$new()
- OutputWidgetClass$display()
- OutputWidgetClass$clear()
- OutputWidgetClass$stdout()
- OutputWidgetClass$stderr()
- OutputWidgetClass$clone()

Method `new()`**:** Initializing function

Usage:

```
OutputWidgetClass$new(append_output = TRUE, ...)
```

Arguments:

`append_output` Logical, whether existing output should be appended to or overwritten.

`...` Any other arguments, passed to the superclass initializer.

`envir` An environment, where expressions are evaluated.

`use_display` Logical, whether the display mechanism is used internally for output streams.

Method `display()`**:** A variant of `display` for output within a display widget.

Usage:

```
OutputWidgetClass$display(...)
```

Arguments:

`...` Further arguments, passed on to the 'evaluate' method of the `Context` class,

Method `clear()`**:** Clear the output

Usage:

```
OutputWidgetClass$clear(wait = FALSE)
```

Arguments:

`wait` Logical, whether to wait for the frontend to clear the output.

Method `stdout()`**:**

Usage:

```
OutputWidgetClass$stdout(text)
```

Method `stderr()`**:**

Usage:

```
OutputWidgetClass$stderr(text)
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Page *Display an object using the Jupyter notebook pager*

Description

This function allows to display an *R* object in the pager of a Jupyter notebook. Note that acts like `display()` when Jupyter Lab is used.

Usage

```
Page(x, ...)

## Default S3 method:
Page(x, start = 1, ...)
```

Arguments

`x`	An object to be displayed in the Notebook pager
`...`	Other arguments, ignored or passed to specific methods.
`start`	Integer, where to start the output.

Methods (by class)

- `Page(default)`: S3 default method – calls `display_data` and marks it as pager payload

Play *A Player Widget*

Description

An R6 class and a constructor function for the creation of a player widget, which automatically increases its value-

Usage

```
Play(
  value = 0L,
  min = 0L,
  max = 100L,
  interval = 100L,
  step = 1L,
  show_repeat = TRUE,
  ...
)
```

Arguments

value	Integer, an initial value.
min	Integer, the minimum value.
max	Integer, the maximum value.
interval	Integer, the maximum value of the intrval .
step	The maximum value for the play control.
show_repeat	Logical, whether to show a repeat toggle button.
...	Further arguments, passed to the superclass constructor.

Functions

- Play(): The player widget constructor function.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::BoundedIntWidget -> Play

Public fields

_model_name Name of the Javascript model in the frontend

_view_name Name of the Javascript model view in the frontend

interval An Integer traitlet, the time interval between between two steps.

step An Integer traitlet, the step size.

_playing A Boolean traitlet, indicates wether the player widget is running.

playing A Boolean traitlet, indicates wether the player widget is running.

_repeat A Boolean traitlet, indicates wether the the repeat toggle is on.

repeat A Boolean traitlet, indicates wether the the repeat toggle is on.

show_repeat A Boolean traitlet, determines whether to show a repeat toggle button.

Methods

Public methods:

- PlayClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

PlotWidget *Widgets to receive plots graphics*

Description

Class and constructors show graphics created by code

Usage

```
SVGWidget(...)

## S3 method for class 'SVGWidget'
with(data, expr, envir = list(), enclos = parent.frame(), ...)

PlotWidget(...)
```

Arguments

...	Arguments, passed to the ImageWidget constructors.
data	An "SVGWidget" object
expr	An expression to evaluate, or a sequence of expression, encapsulated by curly braces.
enclos	An enclosing environment.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::StringWidget -> RKernel::HTML -> SVGWidget

Public fields

context A Context instance or NULL

Methods

Public methods:

- SVGWidgetClass$new()
- SVGWidgetClass$activate()
- SVGWidgetClass$suspend()
- SVGWidgetClass$render()
- SVGWidgetClass$clone()

Method new(): Initialize the object

Usage:

```
SVGWidgetClass$new(..., width = NULL, height = NULL, envir = new.env())
```

Arguments:

`...` Arguments passed to the superclass initializer

`width` A character string, giving the width as a CSS property

`height` A character string, giving the height as a CSS property

`envir` An optional environment within which expressions are evaluated

Method `activate()`**:**

Usage:

```
SVGWidgetClass$activate()
```

Method `suspend()`**:**

Usage:

```
SVGWidgetClass$suspend()
```

Method `render()`**:**

Usage:

```
SVGWidgetClass$render()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Progress` *Progress bars*

Description

Classes and constructor functions for progress bars and styling of them

Usage

```
ProgressStyle(...)

IntProgress(value = 0L, min = 0L, max = 100L, ...)

FloatProgress(value = 0L, min = 0L, max = 100L, ...)
```

Arguments

`...`	Other arguments.
`value`	A floating point number, the initial position of the progress bar
`min`	An floating point number, the minumum value
`max`	An floating point number, the maximum value

Functions

- `ProgressStyle()`: A constructor for a progress bar style
- `IntProgress()`: A constructor for a progress bar style
- `FloatProgress()`: A constructor for a progress bar style

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DescriptionStyle` -> `ProgrssStyle`

Public fields

`_model_name` Name of the Javascript model in the frontend

`bar_color` The colour of the progress bar

Methods

Public methods:

- `ProgressStyleClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::BoundedIntWidget` -> `IntProgress`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`orientation` Orientation of the progress bar, either "horizontal" or "vertical"

`bar_style` General style of the progress bar, either "successs", "info", "warning" or "danger"

`style` Styling of the progress bar, an instance of "ProgressStyleClass"

Methods

Public methods:

- `IntProgressClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::BoundedFloatWidget -> FloatProgress

Public fields

_model_name Name of the Javascript model in the frontend

_view_name Name of the Javascript view in the frontend

orientation Orientation of the progress bar, either "horizontal" or "vertical"

bar_style General style of the progress bar, either "successs", "info", "warning" or "danger"

style Styling of the progress bar, an instance of "ProgressStyleClass"

Methods

Public methods:

- FloatProgressClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

R6Class_ *R6 Objects That Are Not Locked*

Description

This function calls the R6 class constructor in such a way that new members can be added to the created objects.

Usage

```
R6Class_(..., lock_objects = FALSE)
```

Arguments

...	Arguments passed to the superclass constructor
lock_objects	A logical value, indicates whether objects should be locked. See R6Class.

R6Instance *A Generic Constructor for R6 Object Traits*

Description

A Generic Constructor for R6 Object Traits

Usage

```
R6Instance(...)
```

Arguments

...	Arguments passed to the trait instance initializer

R6TraitClass *A Base Class for Traits that are R6 Objects*

Description

A Base Class for Traits that are R6 Objects

A Base Class for Traits that are R6 Objects

Super class

`RKernel::Trait` -> `R6Trait`

Public fields

`value` An R6 object

`class` The R6 class of `value`

Methods

Public methods:

- `R6TraitClass$validator()`
- `R6TraitClass$new()`
- `R6TraitClass$clone()`

Method `validator()`**:** Checks wether `value` has the corret class

Usage:

```
R6TraitClass$validator(value)
```

Arguments:

`value` A value about to be assigned to the trait.

Method `new()`**:** Initialize an object

Usage:

```
R6TraitClass$new(Class, ...)
```

Arguments:

`Class` Class of the object

`...` Values used for initialization

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

Usage:

```
R6TraitClass$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`raw_html` *Send raw HTML code to the frontend*

Description

Send raw HTML code in a character string to the frontend

Usage

```
raw_html(text, id = UUIDgenerate(), update = FALSE)
```

Arguments

`text`	A character string with LaTeX code for math
`id`	A character string with the display id
`update`	A logical value, should an existing display_data option?

`register_magic_handler` *Register a handler for magics*

Description

Similar to ' There are pre-defined magics for LaTeX math, CSS, Javascrippt, HTML, and iframes.

Usage

```
register_magic_handler(magic, handler)
```

Arguments

`magic`	A character string that selects a handler
`handler`	A function that takes at least the argument 'code' and more '...' arguments. The latter are constructed from the arguments of the percentage magic.

remove_displayed_classes *Remove a Class to the 'Displayed' Ones*

Description

Remove a class from those who are output using `display()` when they are autoprinted, i.e. returned as the value of the last expression in a Jupyter notebook cell.

Usage

```
remove_displayed_classes(x)
```

Arguments

x	A character string, the name of a class.

remove_paged_classes *Remove a Class to the 'Paged' Ones*

Description

Remove a class from those who are output using `Page()` when they are autoprinted, i.e. returned as the value of the last expression in a Jupyter notebook cell.

Usage

```
remove_paged_classes(x)
```

Arguments

x	A character string, the name of a class.

`SelectionContainer` *Tabs and Accordions*

Description

Classes and constructor functions for tab and accordion widgets

Usage

```
Accordion(...)

Tab(...)

Stack(...)
```

Arguments

`...`	Arguments passed to the superclass constructor

Functions

- `Accordion()`: The constructor function for accordion widgets.
- `Tab()`: The construction function for accordion widgets.
- `Stack()`: The construction function for accordion widgets.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::Box` -> Box

Public fields

`_titles` A dictionary of strings, for internal use only

`titles` A dictionary of strings, exposed since ipywidgets 8.

`selected_index` An integer, the field currently selected.

Methods

Public methods:

- `SelectionContainerClass$validate_index()`
- `SelectionContainerClass$set_title()`
- `SelectionContainerClass$get_title()`
- `SelectionContainerClass$new()`
- `SelectionContainerClass$clone()`

Method `validate_index()`**:** Validate the index, i.e. check whether it is within range.

Usage:

```
SelectionContainerClass$validate_index(index)
```

Arguments:

`index` An integer number.

Method `set_title()`**:** Set the title of one of the elements.

Usage:

```
SelectionContainerClass$set_title(index, title)
```

Arguments:

`index` The index number of the element to be changed.

`title` A character string, the intended title.

Method `get_title()`**:** Get the title of one of the elements.

Usage:

```
SelectionContainerClass$get_title(index)
```

Arguments:

`index` The index number of the element to be enquired.

Method `new()`**:** An initializer function

Usage:

```
SelectionContainerClass$new(children = list(), ...)
```

Arguments:

`children` A list of widgets

`...` Other arguments, passed to the superclass method

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::Box` -> `Accordion`

Public fields

`_model_name` Name of the Javascript model in the frontend.

`_view_name` Name of the Javascript view in the frontend.

Methods

Public methods:

- AccordionClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::Box -> Tab

Public fields

`_model_name` Name of the Javascript model in the frontend.

`_view_name` Name of the Javascript view in the frontend.

Methods

Public methods:

- TabClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::Box -> Stack

Public fields

`_model_name` Name of the Javascript model in the frontend.

`_view_name` Name of the Javascript view in the frontend.

`required_version` Minimum required ipywidgets version in which the current widget class is supported.

Methods

Public methods:

- `StackClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`SelectionWidget` *Selection widgets*

Description

Classes and constructors for selection widgets, i.e. dropdowns, listboxes etc.

Usage

```
SelectionWidget(options, value, ...)

Dropdown(options, value, ...)

RadioButtons(options, value, ...)

ListBox(options, value, ...)

ToggleButtons(options, value, ...)

SelectionSlider(options, value, ...)

MultipleSelectionWidget(options, value, ...)

ListBoxMultiple(options, value, ...)

SelectionRangeSlider(options, value, ...)
```

Arguments

`options`	A named vector or a vector coerceable into a character vector.
`value`	A trait.
`...`	Any other arguments, ignored.

Functions

- `SelectionWidget()`: A constructor function for selection widgets.
- `Dropdown()`: The construction function for dropdown widgets.
- `RadioButtons()`: The construction function for radiobuttons widgets.
- `ListBox()`: The construction function for listbox-selection widgets.
- `ToggleButtons()`: The construction function for togglebuttons widgets.
- `SelectionSlider()`: The construction function for listbox widgets with multiple selections.
- `MultipleSelectionWidget()`: The construction function for multiple-selection widgets.
- `ListBoxMultiple()`: The construction function for listbox widgets with multiple selections.
- `SelectionRangeSlider()`: The construction function for listbox widgets with multiple selections.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `SelectionWidget`

Public fields

`_options_labels` A unicode vector of option labels.

`index` An integer that refers to currently selected item.

Active bindings

`value` The selected option

Methods

Public methods:

- `SelectionWidgetClass$validate_index()`
- `SelectionWidgetClass$new()`
- `SelectionWidgetClass$clone()`

Method `validate_index()`**:** Validate an index argument.

Usage:

```
SelectionWidgetClass$validate_index(index)
```

Arguments:

`index` The index to be checked.

Method `new()`**:** Initialiser

Usage:

```
SelectionWidgetClass$new(options, value, ...)
```

Arguments:

`options` A named vector or a vector coerceable into a character vector.

`value` Name of a selectable option.

`...` Any other arguments, ignored.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::SelectionWidget` -> `Dropdown`

Public fields

`_model_name` Name of the Javascript model in the frontend.

`_view_name` Name of the Javascript view in the frontend.

Methods

Public methods:

- `DropdownClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::SelectionWidget` -> `RadioButtons`

Public fields

`_model_name` Name of the Javascript model in the frontend.

`_view_name` Name of the Javascript view in the frontend.

Methods

Public methods:

- `RadioButtonsClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::SelectionWidget` -> `ListboxSelect`

Public fields

`_model_name` Name of the Javascript model in the frontend.

`_view_name` Name of the Javascript view in the frontend.

`rows` An integer, the number of rows.

Methods

Public methods:

- `ListboxSelectClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DescriptionStyle` -> `ToggleButtonsStyle`

Public fields

`_model_name` Name of the Javascript model in the frontend.

`button_width` A unicode string, the width in CSS language

`font_weight` A unicode string, the font weight in CSS language

Methods

Public methods:

- `ToggleButtonsStyleClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::SelectionWidget -> ToggleButtons

Public fields

_model_name Name of the Javascript model in the frontend.

_view_name Name of the Javascript view in the frontend.

tooltips A unicode vector with tooltips.

icons A unicode vector with icon specs.

style A TobbleButtonStyle widget

button_style A character string, one of "primary", "success", "info", "warning", "danger", or the empty string.

Methods

Public methods:

- ToggleButtonsClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::SelectionWidget -> SelectionSlider

Public fields

_model_name Name of the Javascript model in the frontend.

_view_name Name of the Javascript view in the frontend.

orientation A Unicode string, either "horizontal" or "vertical"

readout A logical value, whether the value should be showns (read out)

continuous_update A logical value, whether values should be updated as the slider is moved by the user

style A SliderStyle widget

behavior A string that describes the ddragging behavior.

Methods

Public methods:

- SelectionSliderClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> MultipleSelectionWidget

Public fields

`_options_labels` A unicode string vector with labels

`index` An integer vector of indices of currenlty selected elements.

Active bindings

`value` The selected option

Methods

Public methods:

- MultipleSelectionWidgetClass$validate_index()
- MultipleSelectionWidgetClass$new()
- MultipleSelectionWidgetClass$clone()

Method `validate_index()`**:** Validate an index.

Usage:

```
MultipleSelectionWidgetClass$validate_index(index)
```

Arguments:

`index` An index, the index to be checked.

Method `new()`**:** Initialiser

Usage:

```
MultipleSelectionWidgetClass$new(options, value, ...)
```

Arguments:

`options` A named vector or a vector coerceable into a character vector.

`value` Names of selectable options.

`...` Any other arguments, ignored.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::MultipleSelectionWidget` -> `ListboxSelectMultiple`

Public fields

`_model_name` Name of the Javascript model in the frontend.

`_view_name` Name of the Javascript view in the frontend.

`rows` An integer, the number of rows.

Methods

Public methods:

- `ListboxSelectMultipleClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::MultipleSelectionWidget` -> `SelectionRangeSlider`

Public fields

`_model_name` Name of the Javascript model in the frontend.

`_view_name` Name of the Javascript view in the frontend.

`orientation` A Unicode string, either "horizontal" or "vertical"

`readout` A logical value, whether the value should be showns (read out)

`continuous_update` A logical value, whether values should be updated as the slider is moved by the user

`style` A SliderStyle widget

`behavior` A string that describes the ddragging behavior.

Methods

Public methods:

- SelectionRangeSliderClass$validate_index()
- SelectionRangeSliderClass$new()
- SelectionRangeSliderClass$clone()

Method `validate_index()`**:** Validate an index.

Usage:

```
SelectionRangeSliderClass$validate_index(index)
```

Arguments:

`index` An index, the index to be checked.

Method `new()`**:** Initialiser

Usage:

```
SelectionRangeSliderClass$new(options, value, ...)
```

Arguments:

`options` A named vector or a vector coerceable into a character vector.

`value` Names of selectable options.

`...` Any other arguments, ignored.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`sharedHelpServer` *R6 objects that run a help server*

Description

An Object of class "sharedHelper" serves HTML help pages for one or several RKernel processes

Public fields

`port` Integer, the port number

`url` Character string, the base URL of the served help pages

`orig_httpd` A function, set to the original HTTP server function

Methods

Public methods:

- sharedHelpServer$new()
- sharedHelpServer$httpd()
- sharedHelpServer$run()
- sharedHelpServer$publish_port()
- sharedHelpServer$log()
- sharedHelpServer$clone()

Method `new()`**:** Intialize the object

Usage:

```
sharedHelpServer$new(port = 0, prefix = "", use_proxy = FALSE)
```

Arguments:

`port` Integer, a port number

`prefix` A character string, the URL prefix

`use_proxy` A logical value, whether the help server is supposed to be run behind a jupyter proxy

Method `httpd()`**:** The function that serves paths and queries

Usage:

```
sharedHelpServer$httpd(path, query, ...)
```

Arguments:

`path` A character string, the path part of an URL

`query` An optional HTTP query string

`...` Any other arguments, passed on to the original 'httpd' function.

Method `run()`**:** The server loop

Usage:

```
sharedHelpServer$run()
```

Method `publish_port()`**:** Put a port number into a temporary file, for other processes to find

Usage:

```
sharedHelpServer$publish_port(port)
```

Arguments:

`port` An integer, the port number

Method `log()`**:** Put log text into a temporary file, for other processes to read

Usage:

```
sharedHelpServer$log(text)
```

Arguments:

`text` A character string to be added to the log file

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Sidecar` *Sidecar widgets*

Description

Sidecar widgets - work only with Jupyter Lab

Usage

```
Sidecar(...)
```

Arguments

`...`	Arguments passed to the inializer

Functions

- `Sidecar()`: A constructor for sidebar widgets

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::OutputWidget` -> `Sidecar`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_model_module` Name of the Javascript frontend module

`_model_module_version` Version of the Javascript frontend module

`_view_name` Name of the Javascript view in the frontend

`_view_module` Name of the Javascript frontend view module

`_view_module_version` Version of the the Javascript view module

`title` A unicode string, the title of the widget.

`anchor` A string that specifies where the widget s to appear: one of "split-right", "split-left", "split-top", "split-bottom", "tab-before", "tab-after", or "right".

Methods

Public methods:

- SidecarClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Slider` *Sliders*

Description

Classes and constructor functions for sliders (integer and floating-point ones)

Usage

```
IntSlider(value = 0L, min = 0L, max = 100L, ...)

IntRangeSlider(value = c(0L, 50L), min = 0L, max = 100L, ...)

FloatSlider(value = 0, min = 0, max = 100, ...)

FloatRangeSlider(value = c(0, 50), min = 0, max = 100, ...)

FloatLogSlider(value = 1, min = 0, max = 40, base = 10, ...)
```

Arguments

`value`	A floating point number, the current value of the slider
`min`	A floating point number, the minimum value
`max`	A floating point number, the maximum value
`...`	Other arguments.
`base`	A floating point number, the base of the logarithm

Functions

- `IntSlider()`: An integer slider constructor
- `IntRangeSlider()`: An integer range slider constructor
- `FloatSlider()`: A floating point slider constructor
- `FloatRangeSlider()`: A floating point slider range constructor
- `FloatLogSlider()`: A floating point log-slider constructor

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DescriptionStyle -> SliderStyle

Public fields

`_model_name` Name of the Javascript frontend model

`handle_color` Unicode string, the color of the slider handle

Methods

Public methods:

- SliderStyleClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::BoundedIntWidget -> IntSlider

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`step` An Integer traitlet, the minimal step size per slider movement

`orientation` A Unicode string, either "horizontal" or "vertical"

`readout` A logical value, whether the value should be showns (read out)

`readout_format` A unicode string, the format specification

`continuous_update` A logical value, whether values should be updated as the slider is moved by the user

`disabled` A logical value, whether the slider is disabled

`style` A SliderStyle widget

`behavior` A string that describes the ddragging behavior.

Methods

Public methods:

- IntSliderClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::BoundedIntRangeWidget` -> `IntRangeSlider`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`step` An `Integer` traitlet, the minimal step size per slider movement

`orientation` A Unicode string, either "horizontal" or "vertical"

`readout` A logical value, whether the value should be showns (read out)

`readout_format` A logical value, whether values should be updated as the slider is moved by the user

`continuous_update` A logical value, whether values should be updated as the slider is moved by the user

`disabled` A logical value, whether the slider is disabled

`style` A SliderStyle widget

`behavior` A string that describes the ddragging behavior.

Methods

Public methods:

- `IntRangeSliderClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::BoundedFloatWidget` -> `FloatSlider`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`step` A `Float` traitlet, the minimal step size per slider movement

`orientation` A Unicode string, either "horizontal" or "vertical"

`readout` A logical value, whether the value should be showns (read out)

`readout_format` A logical value, whether values should be updated as the slider is moved by the user

`continuous_update` A logical value, whether values should be updated as the slider is moved by the user

`disabled` A logical value, whether the slider is disabled

`style` A SliderStyle widget

`behavior` A string that describes the ddragging behavior.

Methods

Public methods:

- `FloatSliderClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::BoundedFloatRangeWidget` -> `FloatRangeSlider`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`step` A `Float` traitlet, the minimal step size per slider movement

`orientation` A Unicode string, either "horizontal" or "vertical"

`readout` A logical value, whether the value should be showns (read out)

`readout_format` A logical value, whether values should be updated as the slider is moved by the user

`continuous_update` A logical value, whether values should be updated as the slider is moved by the user

`disabled` A logical value, whether the slider is disabled

`style` A SliderStyle widget

`behavior` A string that describes the ddragging behavior.

Methods

Public methods:

- FloatRangeSliderClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::BoundedLogFloatWidget -> FloatLogSlider

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`step` A Float traitlet, the minimal step size per slider movement

`orientation` A Unicode string, either "horizontal" or "vertical"

`readout` A logical value, whether the value should be showns (read out)

`readout_format` A logical value, whether values should be updated as the slider is moved by the user

`continuous_update` A logical value, whether values should be updated as the slider is moved by the user

`disabled` A Boolean traitlet, whether the slider is disabled

`base` A Float traitlet, the base of the logarithm

`style` A SliderStyle widget

`behavior` A string that describes the ddragging behavior.

Methods

Public methods:

- FloatLogSliderClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

StrEnum *An Enumerated String Constructor*

Description

An Enumerated String Constructor

Usage

```
StrEnum(...)
```

Arguments

...	Arguments passed to the trait instance initializer

StrEnumClass *An Enumerated Strings Trait*

Description

An Enumerated Strings Trait

An Enumerated Strings Trait

Super classes

RKernel::Trait -> RKernel::Unicode -> StrEnum

Public fields

`enum` a character vector

`optional` A logical value, whether value can be empty.

Methods

Public methods:

- StrEnumClass$validator()
- StrEnumClass$new()
- StrEnumClass$clone()

Method `validator()`**:** Check whether the assigned vector is one of the allowed enumerated strings.

Usage:

```
StrEnumClass$validator(value)
```

Arguments:

`value` A value to be assigned to the trait

Method `new()`**:** Initialize the trait.

Usage:

```
StrEnumClass$new(enum, default = character(0), optional = FALSE)
```

Arguments:

`enum` A character vector of permitted enumerated strings.

`default` The default value

`optional` Logical can the value be empty?

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`StringWidget` *String widgets*

Description

Classes and constructor functions for string-related widgets (text areas etc.)

Usage

```
LabelStyle(...)

TextStyle(...)

HTMLStyle(...)

HTMLMathStyle(...)

StringWidget(value = character(0), ...)

HTML(value = character(0), ...)

HTMLMath(value = character(0), ...)

Label(value = character(0), ...)

Textarea(value = character(0), ...)

TextWidget(value = character(0), ...)

PasswordWidget(value = character(0), ...)

Combobox(value = character(0), ...)
```

Arguments

...	Arguments passed to the inializer
value	A character vector

Functions

- LabelStyle(): The constructor for Label styles
- TextStyle(): The constructor for Text styles
- HTMLStyle(): The constructor for HTML styles
- HTMLMathStyle(): The constructor for HTMLMath styles
- StringWidget(): A constructor for string widgets
- HTML(): A constructor for HTML widgets
- HTMLMath(): A constructor for HTML widgets with math
- Label(): A constructor for label widgets
- Textarea(): A constructor for text area widgets
- TextWidget(): A constructor for text field widgets
- PasswordWidget(): A constructor for password entry widgets
- Combobox(): A constructor for combo boxes

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DescriptionStyle -> StringStyle

Public fields

_model_name Name of the Javascript model in the frontend

background The background color

font_size The font size

text_color The text color

required_version Minimum required ipywidgets version in which the current widget class is supported.

Methods

Public methods:

- StringStyleClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DescriptionStyle -> RKernel::StringStyle -> LabelStyle

Public fields

_model_name Name of the Javascript model in the frontend

font_family The font family

font_style The font style

font_variant The font variant

font_weight The font weight

text_decoration The text decoration

required_version Minimum required ipywidgets version in which the current widget class is supported.

Methods

Public methods:

- LabelStyleClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DescriptionStyle -> RKernel::StringStyle -> TextStyle

Public fields

_model_name Name of the Javascript model in the frontend

required_version Minimum required ipywidgets version in which the current widget class is supported.

Methods

Public methods:

- TextStyleClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DescriptionStyle -> RKernel::StringStyle -> HTMLStyle

Public fields

_model_name Name of the Javascript model in the frontend

required_version Minimum required ipywidgets version in which the current widget class is supported.

Methods

Public methods:

- HTMLStyleClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DescriptionStyle -> RKernel::StringStyle -> HTMLMathStyle

Public fields

_model_name Name of the Javascript model in the frontend

required_version Minimum required ipywidgets version in which the current widget class is supported.

Methods

Public methods:

- HTMLMathStyleClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> StringWidget

Public fields

`_model_name` Name of the Javascript model in the frontend

`value` A unicode vector

`placeholder` A placeholder character

Methods

Public methods:

- `StringWidgetClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::StringWidget` -> `HTML`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`style` The HTML style, an object of class "HTMLStyleClass"

Methods

Public methods:

- `HTMLClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::StringWidget` -> `HTMLMath`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`style` The HTMLMath style, an object of class "HTMLMathStyleClass"

Methods

Public methods:

- `HTMLMathClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::StringWidget` -> `Label`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`style` The Label style, an object of class "LabelStyleClass"

Methods

Public methods:

- `LabelClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::StringWidget` -> `Textarea`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`disabled` A logical value, whether the slider is disabled

`continuous_update` A logical value, whether values should be updated as the slider is moved by the user

`rows` An integer, the number of rows

`style` The Text style, an object of class "TextStyleClass"

Methods

Public methods:

- `TextareaClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::StringWidget` -> `TextWidget`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`disabled` A logical value, whether the slider is disabled

`continuous_update` A logical value, whether values should be updated as the slider is moved by the user

`style` The Text style, an object of class "TextStyleClass"

Methods

Public methods:

- `TextWidgetClass$clear()`
- `TextWidgetClass$clone()`

Method `clear()`**:** Clear the text area

Usage:

```
TextWidgetClass$clear()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::StringWidget` -> `RKernel::TextWidget` -> `PasswordWidget`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

Methods

Public methods:

- `PasswordWidgetClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::StringWidget` -> `RKernel::TextWidget` -> `Combobox`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`options` A unicode vector, the available options

`ensure_option` A boolean (logical) value, whether at least one option has to be activated

Methods

Public methods:

- `ComboboxClass$clone()`

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

`TagsInput` *Tags Input Widgets*

Description

Classes and constructors to great tag input widgets

Usage

```
TagsInput(...)

ColorsInput(...)

FloatsInput(...)

IntsInput(...)
```

Arguments

...	Arguments passed to the inializer

Functions

- `TagsInput()`: A taginput constructor
- `ColorsInput()`: A color taginput constructor
- `FloatsInput()`: A color taginput constructor
- `IntsInput()`: A color taginput constructor

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `TagsInputBase`

Public fields

`_model_name` Name of the Javascript model in the frontend

`value` A list of tags

`placeholder` A placeholder string

`allowed_tags` Optional list with allowed tags.

`allow_duplicates` Logical, whether duplicate tags are allowed.

`required_version` Minimum required ipywidgets version in which the current widget class is supported.

Methods

Public methods:

- TagsInputBaseClass$validate_value()
- TagsInputBaseClass$new()
- TagsInputBaseClass$clone()

Method `validate_value()`**:** Check value for validity,

Usage:

```
TagsInputBaseClass$validate_value(value)
```

Arguments:

`value` A character string with one or more tags.

Method `new()`**:** Initializer function

Usage:

```
TagsInputBaseClass$new(...)
```

Arguments:

`...` Arguments passed to the superclass initializer

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::TagsInputBase` -> `TagsInput`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend.

`value` A list of tags as unicode strings

`tag_style` The string that describes the tag style

Methods

Public methods:

- TagsInputClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::TagsInputBase -> ColorsInput

Public fields

_model_name Name of the Javascript model in the frontend

_view_name Name of the Javascript view in the frontend.

value A list of tags as unicode strings

Methods

Public methods:

- ColorsInputClass$clone()

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> RKernel::TagsInputBase -> RKernel::TagsInput -> ColorsInput

Public fields

_model_name Name of the Javascript model in the frontend

min A Float traitlet, the minimum allowed value.

max A Float traitlet, the maximum allowed value.

value A list of numeric tags

Methods

Public methods:

- NumbersInputBase$validate_value()
- NumbersInputBase$new()
- NumbersInputBase$clone()

Method validate_value(): Check value for validity,

Usage:

```
NumbersInputBase$validate_value(value)
```

Arguments:

value A character string with one or more tags.

Method `new()`**:** Initializer function

Usage:

```
NumbersInputBase$new(...)
```

Arguments:

`...` Arguments passed to the superclass initializer

`value` An initial value

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::TagsInputBase` -> `RKernel::TagsInput` -> `RKernel::ColorsInput` -> `FloatsInput`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend.

`value` A list of numeric tags

`format` The number format

Methods

Public methods:

- `FloatsInputClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `RKernel::TagsInputBase` -> `RKernel::TagsInput` -> `RKernel::ColorsInput` -> `IntsInput`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend.

`min` A Integer traitlet, the minimum allowed value.

`max` A Integer traitlet, the maximum allowed value.

`value` A list of interger number tags

`format` The number format

Methods

Public methods:

- `IntsInputClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`TimeClass` *Time Traitlets*

Description

A class and constructor of time traitlets.

Super class

`RKernel::Trait` -> `TimeClass`

Public fields

`value` A date.

`coerce` Logical value, whether assignments to the value field should be coerced to the appropriate type.

Methods

Public methods:

- `TimeClass$validator()`
- `TimeClass$new()`
- `TimeClass$clone()`

Method `validator()`**:** Check the value assigned to the traitlet.

Usage:

```
TimeClass$validator(value)
```

Arguments:

`value` The value assigned to the traitlet.

Method `new()`**:** Initialize the traitlet.

Usage:

```
TimeClass$new(initial = as.POSIXct(integer(0)), coerce = TRUE)
```

Arguments:

`initial` An optional POSIXct object or an object coercive into such an object

`coerce` An optional logical value

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`TimePicker` *Time Picker Widgets*

Description

An R6 class and constructor function for time picker widgets

Usage

```
TimePicker(...)
```

Arguments

`...`	Arguments passed to the inializer

Functions

- `TimePicker()`: A constructor for dat picker widgets

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `TimePicker`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript view in the frontend

`value` The date and time. If non-zero length, must have valid timezone info.

`disabled` Boolean, whether the user can make changes

`min` Minimum selectable date and time. If non-zero length, must have valid timezone info.

`max` Maximum selectable date and time. If non-zero length, must have valid timezone info.

Methods

Public methods:

- `TimePickerClass$validate_value()`
- `TimePickerClass$validate_min()`
- `TimePickerClass$validate_max()`
- `TimePickerClass$new()`
- `TimePickerClass$clone()`

Method `validate_value()`**:** Check wether "value" is within range.

Usage:

```
TimePickerClass$validate_value(value)
```

Arguments:

`value` A date and time to be checked for validity

Method `validate_min()`**:** Validate the "min" field after assignment.

Usage:

```
TimePickerClass$validate_min(min)
```

Arguments:

`min` A minimum date and time to be checked for validity

Method `validate_max()`**:** Validate the "max" field after assignment.

Usage:

```
TimePickerClass$validate_max(max)
```

Arguments:

`max` A maximum date and time to be checked for validity

Method `new()`**:**

Usage:

```
TimePickerClass$new(...)
```

Arguments:

`...` Arguments passed to the superclass initializer

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Togglebuttons *Tobble-Button Widgets*

Description

A constructor function and a class to create toggle-button widgets

Usage

```
ToggleButtonStyle(...)

ToggleButton(...)
```

Arguments

...	Arguments passed to the inializer

Functions

- `ToggleButtonStyle()`: The constructor for Togglebuttons styles
- `ToggleButton()`: A toggle-button constructor

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DescriptionStyle` -> `ToggleButtonStyle`

Public fields

`_model_name` Name of the Javascript model in the frontend

`font_family` The font family

`font_size` The font size

`font_style` The font style

`font_variant` The font variant

`font_weight` The font weight

`text_color` The text color

`text_decoration` The text decoration

`required_version` Minimum required ipywidgets version in which the current widget class is supported.

Methods

Public methods:

- ToggleButtonStyleClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::DOMWidget -> RKernel::DescriptionWidget -> RKernel::ValueWidget -> ToggleButton

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript model view in the frontend

`value` Boolean, whether the box is checked

`tooltip` A tooltip

`description` A button description

`disabled` Boolean, whether the button is disabled

`icon` An icon (a fontawesome icon name)

`button_style` The string that describes the button style

`style` The toggle button style, an object of class "ToggleButtonStyleClass"

Methods

Public methods:

- ToggleButtonClass$clone()

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

to_json *A Genereric Converter to JSON*

Description

The function to_json is a generic and idempotent interface to toJSON.

Usage

```
to_json(x, auto_unbox = TRUE, ...)

## Default S3 method:
to_json(x, auto_unbox = TRUE, ...)

## S3 method for class 'list'
to_json(x, auto_unbox = TRUE, ...)

## S3 method for class 'json'
to_json(x, ...)

## S3 method for class 'Trait'
to_json(x, auto_unbox = TRUE, ...)

## S3 method for class 'Bytes'
to_json(x, ...)

## S3 method for class 'Vector'
to_json(x, ...)

## S3 method for class 'List'
to_json(x, auto_unbox = TRUE, ...)

## S3 method for class 'Widget'
to_json(x, ...)
```

Arguments

x	An object to be converted as JSON
auto_unbox	A logical value, whether one-element a JSON list should be changed into JSON scalar.
...	Other arguments, passed on to toJSON.

Methods (by class)

- to_json(default): Default S3 method
- to_json(list): S3 method for lists.

- `to_json(json)`: S3 method for JSON character strings. Returns its argument as is, making `to_json` idempotent.
- `to_json(Trait)`: S3 method for 'TraitClass' objects, i.e. traitlets.
- `to_json(Bytes)`: S3 method for 'BytesClass' objects
- `to_json(Vector)`: S3 method for 'VectorClass' objects
- `to_json(List)`: S3 method for 'ListClass' objects
- `to_json(Widget)`: S3 method for 'WidgetClass' objects, i.e. jupyter widgets

---

`Traitlets` *Traitlets*

Description

The class `TraitClass` brings (some of) the functionality of the traitlets framework on which the ipywidgets framework is based to *R*.

The function `TraitInstance` returns information needed by a `HasTraits` object to construct a `TraitClass` object.

Usage

```
Trait(...)

TraitInstance(Class, ...)
```

Arguments

`...`	Arguments passed to the inializer
`Class`	An R6 Class that inherits from "TraitClass"

Functions

- `Trait()`: A Baseline Trait Constructor
- `TraitInstance()`: A "Delayed Constructor" for Traits, to be used by constructors of derived classes.

Public fields

`value` The value of the trait

`observers` A list of functions to be called as notification callbacks

`validators` A list of functions to check the validity of a

Methods

Public methods:

- TraitClass$set()
- TraitClass$get()
- TraitClass$new()
- TraitClass$clone()

Method `set()`**:** Set the value of the trait

Usage:

```
TraitClass$set(value, notify = FALSE)
```

Arguments:

`value` The value to be set

`notify` Logical; whether to call notification callbacks

Method `get()`**:** Get the trait value

Usage:

```
TraitClass$get()
```

Method `new()`**:** Initialize the trait, i.e. set an initial value

Usage:

```
TraitClass$new(initial)
```

Arguments:

`initial` The initial value

`coerce` Logical; whether to coerce the initial value to the approriate mode.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Unicode` *A Unicode String Trait Constructor*

Description

A Unicode String Trait Constructor

Usage

```
Unicode(...)
```

Arguments

...	Arguments passed to the trait instance initializer

`UnicodeClass` *A Unicode String Vector Trait*

Description

A Unicode String Vector Trait

A Unicode String Vector Trait

Usage

```
## S3 method for class 'Unicode'
as.character(x, ...)
```

Arguments

`x`	A Unicode traitlet
`...`	Other arguments, ignored.

Functions

- `as.character(Unicode)`: Coerce a unicode string trait to a character vector

Super class

`RKernel::Trait` -> `Unicode`

Public fields

`coerce` Logical value, whether values should be coerced to character strings.

`length` Length of the unicode character vector

`value` The value of the unicode character vector

`optional` A logical value, whethe valuer can be empty.

Methods

Public methods:

- `UnicodeClass$validator()`
- `UnicodeClass$new()`
- `UnicodeClass$clone()`

Method `validator()`**:** A validator function

Usage:

```
UnicodeClass$validator(value)
```

Arguments:

`value` The value to be assigned

Method `new()`**:** Initialize an object

Usage:

```
UnicodeClass$new(
  initial = character(0),
  coerce = TRUE,
  optional = length(initial) == 0,
  length = 1L
)
```

Arguments:

`initial` An initial value

`coerce` Logical value, whether values should be coerced to character stringes

`optional` Logical value, whether the value may be empty

`length` Integer, the intended length of the unicode vector

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Valid` *Validity Indicator Widgets*

Description

A constructor function and a class to create toggle-button widgets

Usage

```
Valid(...)
```

Arguments

`...`	Arguments passed to the inializer

Functions

- `Valid()`: A constructor for validity indicator widgets

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `RKernel::ValueWidget` -> `Valid`

Public fields

`_model_name` Name of the Javascript model in the frontend

`_view_name` Name of the Javascript model view in the frontend

`indent` Boolean, whether to indent the indicator widget

`value` Boolean, whether the validity should be indicated

`readout` Text to be shown if the Widget value is FALSE

Methods

Public methods:

- `ValidClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`ValueWidgetClass` *Value widgets*

Description

A base class for widgets that are connected with values

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::DOMWidget` -> `RKernel::DescriptionWidget` -> `ValueWidget`

Public fields

`value` A list or any other vector of values

Methods

Public methods:

- `ValueWidgetClass$new()`
- `ValueWidgetClass$on_change()`
- `ValueWidgetClass$clone()`

Method `new()`**:** A generic initializer function

Usage:

```
ValueWidgetClass$new(value, ...)
```

Arguments:

`value` A value to initialize instance with

`...` Any other arguments, ignored.

Method `on_change()`**:** Add handler function to be called when value is changed

Usage:

```
ValueWidgetClass$on_change(handler, remove = FALSE)
```

Arguments:

`handler` A handler function

`remove` A logical value, whether the handler should be removed or added.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Vector` *Generic Vector Traitles*

Description

A class and a constructor function to create generic vector trait(let)s.

Usage

```
Vector(...)
```

Arguments

`...`	Arguments that are passed to the initialize method of 'VectorClass'

Super class

`RKernel::Trait` -> `Vector`

Public fields

`value` A list

`class` A character string, the common class of the vector elements

Methods

Public methods:

- VectorClass$validator()
- VectorClass$new()
- VectorClass$clone()

Method `validator()`**:** A function that checks the validity of an assigned value, i.e. whether the assigned value is a list with all elements of the same class

Usage:

```
VectorClass$validator(value)
```

Arguments:

`value` A value to be assigned as the traitlet value

Method `new()`**:**

Usage:

```
VectorClass$new(class = NULL, ...)
```

Arguments:

`class` String, optional common class

`...` Arguments passed to the superclass initializer

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`View` *Invoke a Data Viewer*

Description

This is a re-implementation of View that works within Jupyter notebooks by leveraging the Jupyter widgets infrastructure or by using the DataTable Javascript library. The latter is the case if the system option "View.backend" is set to "dataTable" or if this option is not set. Otherwise a 'virtable_widget' is used.

Usage

```
View(x, title = deparse(substitute(x)), ...)

## Default S3 method:
View(x, title = deparse(substitute(x)), ...)

## S3 method for class 'data.frame'
View(x, title = deparse(substitute(x)), ...)
```

Arguments

x	An R object which can be coerced to a data frame with non-zero numbers of rows and columns.
title	A string used as title. Currently unused.
...	Other arguments, ignored.

---

virtable_widget *Widgets to show tabular data*

Description

This is a widget that can be used to potentially large tabular data objects, such as data frames. It is also a potential backend for the View function.

Usage

```
virtable_widget(
  x,
  pagesize = getOption("rkernel_view_size", c(10, 10)),
  continuous_update = TRUE
)

fmt_tab_section(x, i, j)

## Default S3 method:
fmt_tab_section(x, i, j)

## S3 method for class 'tbl_df'
fmt_tab_section(x, i, j)
```

Arguments

x	A tabular object, e.g. a data frame
pagesize	Number of rows and columns that are shown by the widget.
continuous_update	Logical, whether sliders should lead continuous updates.
i	Integer values, indexing rows
j	Integer values, indexing cols

Functions

- `fmt_tab_section()`: Format a section of a tabular object to be used by `virtable_widget`.
- `fmt_tab_section(default)`: Default method
- `fmt_tab_section(tbl_df)`: Dibble method

WidgetLink *Synchronize two Widgets Using a Link*

Description

An R6 class and a constructor function for the creation of a link widget, which links two widgets so that their values are synchronized

Usage

```
WidgetLink(source, target, ...)

DirectionalLink(source, target, ...)
```

Arguments

source	A link with two elements, the first is a widget, the second is one of its traits.
target	A link with two elements, the first is a widget, the second is one of its traits.
...	Other arguments passed to the inializer

Details

The function WidgetLink creates objects of the R6 Class "WidgetLinkClass", which in turn have the S3 class attribute "WidgetLink"

Functions

- WidgetLink(): The WidgetLink constructor function
- DirectionalLink(): The WidgetLink constructor function

Super classes

RKernel::HasTraits -> RKernel::Widget -> RKernel::CoreWidget -> WidgetLink

Public fields

_model_name Name of the Javascript model in the frontend.

source A pair of Unicode strings, the first is the JSON representation of a widget, the second is the name of a trait(let) of the widget.

target A pair of Unicode strings, the first is the JSON representation of a widget, the second is the name of a trait(let) of the widget.

Methods

Public methods:

- `WidgetLinkClass$new()`
- `WidgetLinkClass$clone()`

Method `new()`**:** An initializer method

Usage:

```
WidgetLinkClass$new(source, target, ...)
```

Arguments:

`source` A list with two elements, a widget and the name of a trait(let).

`target` A list with two elements, a widget and the name of a trait(let).

`...` Futher arguments, passed to the superclass initializer.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Super classes

`RKernel::HasTraits` -> `RKernel::Widget` -> `RKernel::CoreWidget` -> `RKernel::WidgetLink` -> `DirectionalLink`

Public fields

`_model_name` Name of the Javascript model in the frontend.

Methods

Public methods:

- `DirectionalLinkClass$clone()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Widgets` *A Widget Base Class*

Description

The base class from which all widget classes are derived

Usage

```
Widget(...)
```

Arguments

`...`	Arguments passed to the inializer

Functions

- `Widget()`: A Widget Constructor Function

Super class

`RKernel::HasTraits` -> `Widget`

Public fields

`_model_id` Identifyer of the frontend Javascript object

`_model_name` Name of the Javascript model in the frontend

`_model_module` Name of the Javascript module with the model

`_model_module_version` Version of the module where the model is defined

`_view_name` Name of the Javascript model view in the frontend

`_view_module` Version of the module where the view is defined

`_view_module_version` Version of the module where the view is defined

`_view_count` Number of views that refer to the same frontend model object

`traits_to_sync` Names of the traits to be synchronized with the frontend

`sync_suspended` Logical value, whether synchronization is suspended

`custom_msg_callbacks` A list of functions to be called on receiving a message

`event_callbacks` A list of functions to be called on an event

`displayed_callbacks` A list of functions to be called when the widget

`_comm` The 'comm' connecting to the frontend or NULL

`required_version` Minimum required ipywidgets version in which the current widget class is supported.

Active bindings

`comm` The 'comm' connecting the frontend (as an active binding)

Methods

Public methods:

- `WidgetClass$new()`
- `WidgetClass$open()`
- `WidgetClass$finalize()`
- `WidgetClass$close()`
- `WidgetClass$get_state()`
- `WidgetClass$set_state()`
- `WidgetClass$send_state()`
- `WidgetClass$send()`
- `WidgetClass$display_data()`
- `WidgetClass$handle_comm_opened()`
- `WidgetClass$handle_comm_msg()`
- `WidgetClass$handle_buffers()`
- `WidgetClass$handle_custom_msg()`
- `WidgetClass$on_msg()`
- `WidgetClass$on_event()`
- `WidgetClass$handle_event()`
- `WidgetClass$on_displayed()`
- `WidgetClass$handle_displayed()`
- `WidgetClass$_send()`
- `WidgetClass$check_version()`
- `WidgetClass$clone()`

Method `new()`**:** Initialize an object

Usage:

```
WidgetClass$new(..., open = TRUE)
```

Arguments:

`...` Values used for initialization

`open` Logical, whether a connection with the frontend should be opened

Method `open()`**:** Open a connection to the frontend

Usage:

```
WidgetClass$open()
```

Method `finalize()`**:** Finalize the object

Usage:

```
WidgetClass$finalize()
```

Method `close()`**:** Close the connection to the frontend

Usage:

```
WidgetClass$close()
```

Method `get_state()`**:** Prepare synchronized traits for sending them to the frontend

Usage:

```
WidgetClass$get_state(keys = NULL)
```

Arguments:

`keys` Keys/names of the traits to be updated in the frontend

Method `set_state()`**:** Update the synchronized states, usually with information from the frontend

Usage:

```
WidgetClass$set_state(state)
```

Arguments:

`state` A list of values for the synchronized traits

Method `send_state()`**:** Send updated traits to the frontend

Usage:

```
WidgetClass$send_state(keys = NULL, drop_defaults = FALSE)
```

Arguments:

`keys` Keys/names of the traits to be updated in the frontend

`drop_defaults` Logical value, not yet used

Method `send()`**:** Send content and binary buffers to the fronend

Usage:

```
WidgetClass$send(content, buffers = NULL)
```

Arguments:

`content` Some user-defined information to be send to the frontend

`buffers` Some raw vector buffers

Method `display_data()`**:** Send display-data of the widget to the frontend

Usage:

```
WidgetClass$display_data()
```

Method `handle_comm_opened()`**:** Handle a 'comm' opened in the frontend

Usage:

```
WidgetClass$handle_comm_opened(comm, data)
```

Arguments:

`comm` The 'comm' object that is opened

`data` Data sent by the frontend

Method `handle_comm_msg()`**:** Handle a message from the frontend

Usage:

```
WidgetClass$handle_comm_msg(comm, msg)
```

Arguments:

`comm` The 'comm' object via which the message is received

`msg` Message sent by the frontend

Method `handle_buffers()`**:** Handle buffers in message. This method should be overwritten by inherting classes that actually process data in buffer components of messages.

Usage:

```
WidgetClass$handle_buffers(msg)
```

Arguments:

`msg` A comm message

Method `handle_custom_msg()`**:** Call the custom message handlers

Usage:

```
WidgetClass$handle_custom_msg(content)
```

Arguments:

`content` The data received

Method `on_msg()`**:** Install a handler for messages being received

Usage:

```
WidgetClass$on_msg(handler, remove = FALSE)
```

Arguments:

`handler` A handler function

`remove` Logical, should the handler be removed?

Method `on_event()`**:** Install a handler for events in the frontend

Usage:

```
WidgetClass$on_event(event, handler, remove = FALSE)
```

Arguments:

`event` A character that describes the event

`handler` A handler function

`remove` Logical, should the handler be removed?

Method `handle_event()`**:** Call the installed event handlers

Usage:

```
WidgetClass$handle_event(event, args)
```

Arguments:

`event` A string that describes the event

`args` A list of argument passed on with the event

Method `on_displayed()`**:** Install a handler to be called when the widget is displayed

Usage:

```
WidgetClass$on_displayed(handler, remove = FALSE)
```

Arguments:

`handler` A handler function

`remove` Logical, should the handler be removed?

Method `handle_displayed()`**:** Call the installed display handlers

Usage:

```
WidgetClass$handle_displayed()
```

Method `_send()`**:** The internal function to send messages to the frontend

Usage:

```
WidgetClass$_send(msg, buffers = NULL)
```

Arguments:

`msg` The message

`buffers` Raw data buffers or NULL

Method `check_version()`**:** Check whether current widget class is supported by ipywidgets

Usage:

```
WidgetClass$check_version()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Index