

Package: kitems (via r-universe)

October 1, 2026

Title A Framework To Manage Tabular Items

Version 0.8.0.9000

Description A framework to manage tabular items and a set of tools to implement it within R Shiny applications.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0), shiny

Imports DT, dplyr, rlang, iker (>= 0.9.5), ktools (>= 0.9.9), stringr, yaml, shinyjs, bslib, shinyWidgets, crayon, lifecycle

Suggests knitr, rmarkdown, quarto, testthat (>= 3.0.0)

VignetteBuilder quarto

Config/testthat/edition 3

Remotes thekangaroofactory/iker, thekangaroofactory/ktools

URL <https://thekangaroofactory.github.io/kitems/>

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libx11-dev zlib1g-dev

Repository <https://thekangaroofactory.r-universe.dev>

Date/Publication 2026-09-26 09:40:13 UTC

RemoteUrl <https://github.com/thekangaroofactory/kitems>

RemoteRef HEAD

RemoteSha e1bdf18a118effe1f0d6e59b5c7420523352be9a

Contents

adjust	3
admin	4
amend	4
as_default	5

attribute_input	6
backup	7
check	8
compute	9
create_widget	10
date_slider_widget	11
decorate	12
default	12
delete	13
design	14
dialog	15
dm_migrate	16
dm_to_yaml	17
enforce	17
extend	18
extract	19
filter_event	20
form	21
has_date_attribute	21
hide	22
insert	23
is_truthy	24
item_load	25
item_save	25
item_widget	26
items	26
kitems	27
name	29
organize	30
prepare	31
refresh	32
restore	33
reveal	34
runExample	35
shrink	36
skip	37
trigger_event	38
update	39
validate	40
yaml_to_dm	41

adjust	<i>Sort Items</i>
--------	-------------------

Description

Sort Items

Usage

```
adjust(items, config, item = get_context())
```

Arguments

items	a data.frame of the items.
config	the config list.
item	the name (id) of the item group.

Details

The sorting order is given by the sort entry in the config list.

When item is set in the parent frame, then the attribute can be skipped in the function call.

Value

a data.frame.

See Also

[organize\(\)](#)

Examples

```
## Not run:  
items |> adjust(config, item = "foo")  
  
## End(Not run)
```

admin	<i>Admin Console</i>
-------	----------------------

Description

Launches the administration console.

Usage

```
admin()
```

Details

The Admin Console is a standalone Shiny web app delivered along with the package to administrate the items of the project.

It will check the R_KITEMS_PATH environment variable and look for the YAML config file in the provided path.

Examples

```
## Not run:
# set environment (where to find the _kitems.yml)
Sys.setenv("R_KITEMS_PATH" = "D:/data")

# launch the Admin Console
admin()

## End(Not run)
```

amend	<i>Amend Attribute</i>
-------	------------------------

Description

Update specific parameter(s) of specific attribute(s).

Usage

```
amend(config, item = get_context(), ...)
```

Arguments

config	the config list.
item	the name (id) of the item group.
...	one or several attribute instructions.

Details

The function only understands attribute instruction. It can handle multiple instructions in . . .

Note that it is forbidden to update:

- the id attribute (any parameter),
- the name and/or type of an attribute.

When item is set in the parent frame, then the attribute can be skipped in the function call.

Value

a list

See Also

`design()`, `parent.frame()`

Examples

```
## Not run:
# create baseline
config <- design(project = "test",
                 item = "foo") |>
  extend(item = "foo",
        attribute = c(name = "total", type = "integer"))

# skip item argument (in the following calls)
item <- "foo"

# single instruction
config |>
  amend(attribute = c(name = "total", default = 12))
config |>
  amend(attribute = c(name = "total", values = "suggest(12)"))

# multiple instructions
config |>
  amend(attribute = c(name = "total", values = "suggest(12)"),
        attribute = c(name = "total", default = "0"))

## End(Not run)
```

as_default

Turn Item Into Default Value(s)

Description

Turn Item Into Default Value(s)

Usage

```
as_default(item, data.model)
```

Arguments

`item` the data.frame of the item to use as a reference.
`data.model` the data.frame of the data.model (see details).

Details

This function does not accept the data.model element of the config list as an input. It should be turned into a data.frame first, using `yaml_to_dm()`.

data.model should contain the following columns: "name", "type", "default", "values".

item and data.model must have same structure. That means the names of the attributes in the data model are expected to match with the names of the columns in item.

Value

a data.frame to pass to form() function.

Examples

```
## Not run:
# create a config
config <- design(project = "foo", item = "bar") | >
extend(item = "bar", attribute = c(name = "date", type = "Date", default = "Sys.Date()"))

# get the default
as_default(item, data.model = yaml_to_dm(config, "name", "type", "default", "values"))

## End(Not run)
```

attribute_input

Attribute Input

Description

Build input for an attribute.

Usage

```
attribute_input(
  name,
  type,
  value = NULL,
  choices = NULL,
  create = FALSE,
  session = getDefaultReactiveDomain()
)
```

Arguments

name	the name of the attribute.
type	the type of the attribute.
value	the value to be used to initialize the input.
choices	a list of values to select from (see details).
create	a logical (default = FALSE) if user is allowed to create values (see details).
session	optional, the shiny session object.

Details

By default (choices = NULL), the function will return an input driven by the type of the attribute. When choices are provided, it will be replaced by a selectizeInput. User will be allowed to create additional values depending on create (otherwise ignored).

Value

An input that can be added to the UI definition.

Examples

```
## Not run:
# -- create inputs
attribute_input(name = "total", type = "numeric", value = 10)

## End(Not run)
```

 backup

Backup Files

Description

Backup config, data model or items files.

Usage

```
backup(
  type = c("config", "items", "dm"),
  id = NULL,
  max = 1,
  path = Sys.getenv("R_KITEMS_PATH")
)
```

Arguments

type	the type of file to backup (see details).
id	the id of the item or data model.
max	an integer (default = 1) to indicate how many backup files are allowed.
path	optional, the path to the data.

Details

type accepts the following values:

- "config" (the default)
- "items"
- "dm"

"dm" is kept to enable data.model backup before migration to the YAML config.

id will be ignored if type = "config"

The recommended way to define the path argument is to set the R_KITEMS_PATH environment variable.

Backup file will be named as *_kitems_YYYY-MM-DD.yml*, *id_data_model_YYYY-MM-DD.rds* or *id_items_YYYY-MM-DD.csv*. If same file already exists, it will be overwritten.

Examples

```
## Not run:
# backup config
backup()

## End(Not run)
```

check

Check Integrity

Description

Wrapper function to either check the YAML config or items integrity.

Usage

```
check(...)
```

Arguments

... the object(s) to check (see details)

Details

The function is smart enough to detect what is passed to To check the items, it is necessary to also pass the corresponding id in the config (see examples).

Value

a list

Examples

```
## Not run:
# check config (list)
config <- design(project = "test")
check(config)

# check items (data.frame)
check(items, config, id = "foo")

## End(Not run)
```

compute

Attribute Values

Description

Compute attribute values with data-masking support.

Usage

```
compute(x, data = NULL)
```

Arguments

x	a character string of the expression to evaluate.
data	an optional data.frame to use for tidy evaluation of x.

Value

a vector of values.

Examples

```
## Not run:
# baseline
compute(x = "suggest(1, 2)")

# with data masking
compute(x = "suggest(min(value))", data = data.frame(value = c(12, 10)))

## End(Not run)
```

`create_widget`*Item Buttons*

Description

Action button(s) to fire the item dialog.

Usage`create_widget(id)``update_widget(id)``delete_widget(id)``actions_widget(id)`**Arguments**

`id` the server module id.

Details

`id` is the namespace of the module server instance holding the target item.

- `create_widget()` to fire create dialog
- `update_widget()` to fire update dialog
- `delete_widget()` to fire delete dialog
- `actions_widget()` is a wrapper that returns all there buttons.

Value

An HTML tag.

Examples

```
## Not run:  
# assuming the module server has been launched with id = "mydata"  
create_widget(id = "mydata")  
  
## End(Not run)
```

date_slider_widget	<i>Date Slider Widget</i>
--------------------	---------------------------

Description

The UI component to drive the date filter.

Usage

```
date_slider_widget(id, width = "300px", bootstrap = bslib::version_default())
```

Arguments

id	the server module id.
width	the width of the input (passed to <code>shiny::sliderInput()</code>).
bootstrap	optional, a character string for the Bootstrap major version (see details).

Details

`bootstrap` has been added in version 0.8.0 to determine if `bslib::popover()` can be used in the widget. The default is provided by `bslib::version_default()`, but there is no obvious way to know - from the UI side - if the app uses shiny's default or bslib layouts.

In case you are using shiny's layout (ex: `shiny::fluidPage()`), you should set the parameter to `bootstrap = "3"`.

Value

An HTML tag.

Examples

```
## Not run:
# with bslib or Bootstrap 5 in general
date_slider_widget(id = "mydata")

# with Bootstrap 3
date_slider_widget(id = "mydata", bootstrap = "3")

## End(Not run)
```

decorate	<i>Decorate Item Columns</i>
----------	------------------------------

Description

Apply mask on attribute names.

Usage

decorate(x)

Arguments

x a data.frame or named object

Details

The function makes the column names pretty.

Value

A renamed data.frame or object.

Examples

```
## Not run:
decorate(x = dm)

## End(Not run)
```

default	<i>Default Value(s)</i>
---------	-------------------------

Description

Compute default value(s).

Usage

default(data.model)

Arguments

data.model the data.frame of the data model.

Details

This function does not accept the `data.model` element of the config list as an input. It should be turned into a `data.frame` first, using `yaml_to_dm()`.

`data.model` should contain the following columns: "name", "type", "default" and "values".

Value

A `data.frame`.

Examples

```
## Not run:
# create a config
config <- design(project = "foo", item = "bar") | >
extend(item = "bar", attribute = c(name = "date", type = "Date", default = "Sys.Date()"))

# get the default
default(yaml_to_dm(config, "name", "type", "default", "values"))

## End(Not run)
```

delete

Delete Item(s)

Description

Delete Item(s)

Usage

```
delete(items, id)
```

Arguments

<code>items</code>	the <code>data.frame</code> of the items.
<code>id</code>	a vector of <code>id(s)</code> for the item(s) to delete.

Details

The function will drop unmatched `id(s)`.

Value

a `data.frame`.

Examples

```
## Not run:
delete(items = myitems, id = 123456789)

## End(Not run)
```

design

Build YAML Config

Description

Design the project configuration.

Usage

```
design(...)
```

Arguments

... one or several instruction(s) to execute (see details).

Details

design is the core function / verb when it comes to building configuration.

The function understands the following instructions passed to ...:

- project to create a config from scratch
- item to add an item group to the project config
- attribute to add an attribute to a given item group.

An instruction is always formed of a key (project, item or attribute) and a value. The value can either be a length-one character string or a named character vector. The project = "test" instruction is used in almost all function documentation examples to init a project config. item = "foo" or item = c(id = "foo", description = "not bar") are used to add an item group to the project.

attribute = c(item = "foo", name = "value", type = "integer") creates an attribute named 'value' that will accept integers inside the item group 'foo'.

Multiple instructions are supported as recursion is implemented (the function can call itself to execute each instruction and merge the outputs).

As the package grammar is designed for a layered approach, design only covers attribute creation without fine tuning. This means it creates attribute(s) with name & type only.

- use extend to create an attribute with fine tuning.
- use hide, display, skip, include & refresh to set specific attribute behaviors.

In addition to the named instructions, a single unnamed argument is accepted among ... to support piping from the config object (see examples). If multiple unnamed objects are passed, an error will be thrown.

Value

a list.

Examples

```
## Not run:
# create project config from scratch
config <- design(project = "foo")

# add (single) item group
config |> design(item = "foo")
config |> design(item = c(id = "foo", description = "x"))

# add (multiple) item groups
config |> design(item = "foo",
                item = "bar")
config |> design(item = c(id = "foo", description = "x"),
                item = c(id = "bar", description = "z"))
config |> design(item = c(id = "foo", description = "x"),
                item = "bar")
# add (single) attribute to item group
config |> design(item = "foo",
                attribute = c(item = "foo", name = "value", type = "integer"))
# add (multiple) attributes to item group
config |> design(item = "foo",
                attribute = c(item = "foo", name = "value", type = "integer"),
                attribute = c(item = "foo", name = "comment", type = "character"))
# if instruction is not understood
design(dream = "draw me a sheep")

## End(Not run)
```

dialog

Item Modal Dialog(s)

Description

Produces a create / update / delete modal dialog.

Usage

```
dialog(
  ...,
  workflow = c("create", "update", "delete"),
  session = getDefaultReactiveDomain()
)
```

Arguments

... the content to be displayed in the modal dialog.
workflow a character string to indicate workflow (see details).
session optional, the shiny session object.

Details

Possible values for workflow are "create" (default), "update" or "delete".

... is typically the output of the form() function. When workflow = "delete", it will be ignored and replaced by a standard message.

Value

a modal dialog.

Examples

```
## Not run:  
dialog(workflow = "delete")  
  
## End(Not run)
```

dm_migrate

Data Model Migration

Description

Execute the migration procedure on a data model.

Usage

```
dm_migrate(data.model)
```

Arguments

data.model a data.frame of the data model.

Value

The migrated data model or NA if no migration is required.

Examples

```
## Not run:  
dm_migrate(data.model)  
  
## End(Not run)
```

dm_to_yaml	<i>Data Model To YAML</i>
------------	---------------------------

Description

Convert data model to YAML configuration file.

Usage

```
dm_to_yaml(dm)
```

Arguments

dm	a data.frame of the data model
----	--------------------------------

Details

The input data model should have its version = "0.8.0" otherwise an error will be raised. Use dm_migrate() first if needed.

Value

a list.

Examples

```
## Not run:  
dm_to_yaml(dm)  
  
## End(Not run)
```

enforce	<i>Item Migration</i>
---------	-----------------------

Description

Add attribute to existing items.

Usage

```
enforce(items, name, type, fill = NA)
```

Arguments

items	a data.frame of the items.
name	a character string of the attribute name.
type	a character string of the attribute type.
fill	the value (default = NA) to be used to fill the existing rows (see details).

Details

fill will be coerced to the class name provided in type. If a vector is given as input for fill, it will be used to feed the created column. Make sure the vector length is same as the number of rows, otherwise an error will be raised by R.

Value

The updated items data.frame.

Examples

```
## Not run:
enforce(items = myitems, name = "comment", type = "character", fill = "none")

## End(Not run)
```

extend	<i>Extend Item</i>
--------	--------------------

Description

Add specific attribute(s) with fine tuning.

Usage

```
extend(config, item = get_context(), ...)
```

Arguments

- config the config object.
- item the name (id) of the item group.
- ... one or several attribute instructions.

Details

This function is a wrapper around design() to provide a comprehensive layered syntax.
It only understands the attribute instruction inside ...: attribute = c(name = "note", type = "character") Anything else will be filtered.
When item is set in the parent frame, then the parameter can be skipped in the function call.

Value

a list.

See Also

```
design\(\), parent.frame\(\)
```

Examples

```
## Not run:
# create config
config <- design(project = "test",
  item = "foo")

# set item (to skip it in following calls)
item <- "foo"

# extend
config |>
  extend(attribute = c(name = "date", type = "Date", default = "Sys.Date()"))

# is same as
config <- design(project = "test",
  item = "foo",
  attribute = c(name = "date", type = "Date", default = "Sys.Date()"))

## End(Not run)
```

extract

Input Values

Description

Extract the list of values from the input.

Usage

```
extract(input, colClasses)
```

Arguments

input	the input object from the shiny module.
colClasses	a named vector of classes, defining the data model.

Details

The output list will contain as many entries as the colClasses named vector. In case some names have no corresponding item in the input parameter, they will get NULL as value in the output list.

Value

a named list.

Examples

```
## Not run:
values <- extract(input, colClasses = c("date" = "Date", "text" = "character"))

## End(Not run)
```

filter_event

Filter Event

Description

Helper function to create filter events to be passed to kitems module server with the filter argument.

Usage

```
filter_event(layer = c("pre", "main"), ...)
```

Arguments

layer	the filter layer ("pre" or "main").
...	the expression(s) to pass to the filter.

Details

When no expression is passed to ... then the filter layer will be reset.

See the module server function to know how to pass the event to the module server.

For more details see this vignette: `vignette("filtering", package = "kitems")`

Value

A list.

See Also

[kitems\(\)](#)

Examples

```
## Not run:
# set pre-filtering layer
filter_event(layer = "pre", expr = name == Banana)

# reset main filter
filter_event(layer = "main")

## End(Not run)
```

form	<i>Item Form</i>
------	------------------

Description

Builds the create or update item form.

Usage

```
form(attributes, items = NULL)
```

Arguments

- attributes a data.frame of the attributes (see details).
- items an optional data.frame of the items (see details).

Details

attributes is expected to be the output of the default() or as_default() function. The data.frame should have the following columns: "name", "type", "default", "values".

items are required only when the attribute values defined in the data.model should be evaluated with data masking support.

Value

A list of HTML tags.

Examples

```
## Not run:
form(attributes = data.frame(name = "total", type = "numeric"))

## End(Not run)
```

has_date_attribute	<i>Has Date</i>
--------------------	-----------------

Description

Basic check if item has a 'date' attribute.

Usage

```
has_date_attribute(config, item = get_context())
```

Arguments

config	the config list
item	the name (id) of the item group

Details

When item is set in the parent frame, then the attribute can be skipped in the function call.

Value

a logical

See Also

[parent.frame\(\)](#)

Examples

```
has_date_attribute(design(project = "test"))
```

hide

Display Behavior Grammar

Description

A set of verbs to manipulate the display of an attribute.

Usage

```
hide(config, item = get_context(), ...)

display(config, item = get_context(), ...)

hidden(config, item = get_context())

displayed(config, item = get_context())
```

Arguments

config	the config list.
item	the name (id) of the item group.
...	the name of the attribute(s) to manipulate.

Details

hide() and display() are setter functions to tune attribute's behavior. hidden() and displayed() are getter functions to quickly access the attributes to hide or display in the item table.

When item is set in the parent frame, then the attribute can be skipped in the function call.

Value

a list (setters) or a character vector (getters).

See Also

`amend()`, `parent.frame()`

Examples

```
## Not run:
# hide attribute
config <- design(project = "test",
  item = "foo") |>
extend(item = "foo", attribute = c(name = "date", type = "Date")) |>
hide(item = "foo", "date")

# allow skipping argument in following calls
item <- "foo"

# hidden attributes
config |> hidden()

# display attribute
config |> display("date")

# displayed attributes
config |> displayed()

## End(Not run)
```

insert

Insert Item(s)

Description

Insert Item(s)

Usage

```
insert(values, items)
```

Arguments

values	a data.frame of item(s) to insert.
items	a data.frame of items where to insert.

Details

The function is mostly a wrapper around `dplyr::bind_rows()` function.

It does check for potential duplicates among the 'id' column. If so, the operation will be aborted as there is no way to know how to fix the duplicated id(s).

Value

a data.frame.

Examples

```
## Not run:  
insert(values, items)  
## End(Not run)
```

is_truthy	<i>Truthy Value</i>
-----------	---------------------

Description

Inspired by `shiny::isTruthy()`, the purpose here is to determine if a value is considered as valid to be an attribute value. (or if it will need to be replaced by the attribute defaults).

Usage

```
is_truthy(x)
```

Arguments

x an object to test.

Value

A logical.

Examples

```
is_truthy(12)
```

item_load*Load Items*

Description

Load Items

Usage

```
item_load(connector, col.classes)
```

Arguments

connector a list that will be passed to [iker::load_data\(\)](#) function.
col.classes a named vector containing the expected column types.

Value

The data.frame of the items.

Examples

```
## Not run:  
# -- File connector:  
item_load(col.classes = c(id = "numeric", date = "Date", comment = "character"),  
connector = list(type = "file", path = "path/to/my/data", filename = "mydata.csv"))  
  
## End(Not run)
```

item_save*Save Items*

Description

Saves the items data.frame.

Usage

```
item_save(data, connector)
```

Arguments

data a data.frame containing the data to be saved.
connector a list that will be passed to [iker::save_data\(\)](#) function.

Examples

```
## Not run:
# -- File connector:
item_save(data = mydata, connector = list(type = "file", file = "path/to/my/data/mydata.csv"))

## End(Not run)
```

item_widget	<i>Item Widget</i>
-------------	--------------------

Description

The UI component for the item table.

Usage

```
item_widget(id)
```

Arguments

id the id of the module server instance.

Value

An HTML tag.

Examples

```
## Not run:
item_widget(id = "mydata")

## End(Not run)
```

items	<i>Get Items</i>
-------	------------------

Description

[Experimental]
This function is part of the selective loading mechanism to ensure items are loaded only when required.

Usage

```
items(datamart, config, item)
```

Arguments

datamart	a reactiveValues object holding the items.
config	the config list.
item	the name (id) of the item group.

Value

a data.frame of the items or NULL.

Examples

```
## Not run:
# init the datamart
datamart <- reactiveValues()

# get items (it will load them)
datamart |> items("foo")

# get them again (already loaded)
datamart |> items("foo")

## End(Not run)
```

kitems

Kitems Module Server

Description

This is the main component of the package.

Usage

```
kitems(
  id,
  path = Sys.getenv("R_KITEMS_PATH"),
  trigger = NULL,
  filter = NULL,
  options = list(autosave = TRUE, notify = TRUE)
)
```

Arguments

id	the unique id of the module server instance.
path	where the YALM config and items are stored (see details).
trigger	an optional reactive object to pass workflow events (see details).
filter	an optional reactive object to pass filter events (see details).
options	an optional list of options (see details).

Details

Since version 0.8.0, the recommended way to define the path argument is to set the `R_KITEMS_PATH` environment variable. If directly passed as an argument value, a warning will be raised at the console.

Behavior of the module server can be tuned using a list of options:

- `autosave` is a logical whether the item auto save should be activated or not (default = `TRUE`)
- `notify` is a logical if Shiny notifications should be displayed (default = `TRUE`) Partial lists are supported, with missing elements getting the default value.

If `autosave` option is `FALSE`, the `item_save()` function should be used to make the data persistent.

Triggers are the way to send events for the module to execute dedicated actions. `trigger` must be a reactive (or `NULL`, the default). An event is defined as a named list (see examples) and may be created using the `trigger_event()` function. If `NULL`, the trigger manager will not be initialized.

`filter` is a reactive object reference to pass filter expression(s) to the module server. A filter is defined as a named list (see examples). It may be created using the `filter_event()` helper function. If `NULL`, the filter manager will not be initialized.

Value

a list

Details about the elements of this list:

- `id` = the id of the module (same as the input)
- `items` = the reference of the items reactive object
- `data_model` = a list, describing the item's data model
- `filtered_items` = the reference of the filtered items reactive object
- `selected_items` = the reference of the selected items (ids) reactive object
- `clicked_column` = the reference of the clicked column reactive object
- `filters` = the reference of the reactive list with filter expressions.

See Also

`trigger_event()`, `filter_event()`

Examples

```
## Not run:
# baseline:
# launch module server to handle item group named "foo"
kitems(id = "foo")

# read only:
kitems(id = "foo", options = list(autosave = FALSE))

# trigger:
# pass workflow events to the module server
```

```

trigger <- reactiveVal()
kitems(id = "foo", trigger = trigger)

# fire the create item dialog:
trigger(list(workflow = "create", type = "dialog"))

# fire an item creation (no dialog):
# assuming 'foo' item has a 'name' character attribute
trigger(list(workflow = "create", type = "task", values = list(name = "test")))

# filter:
# pass filter events to the module server
filter <- reactiveVal()
kitems(id = "foo", filter = filter)

# apply filter at the pre-filtering layer
# assuming item 'foo' has a 'total' numeric attribute
filter(list(layer = "pre", expr = total > 10))

## End(Not run)

```

name	<i>Kitems Names</i>
------	---------------------

Description

Helper function to homogenize names across the package.

Usage

```

name(
  id = NULL,
  what = c("item", "dm", "config"),
  file = FALSE,
  url = FALSE,
  backup = FALSE
)

```

Arguments

id	the name (id) of the item group.
what	the targeted object (item, dm or config).
file	a logical if file extension should be added.
url	a logical if the url should be returned.
backup	a logical if a timestamp should be added.

Details

what accepts "dm" for backward compatibility reasons (backup & restore old data.model files before migration to YAML config).

When url is TRUE, then file is considered TRUE as well. When backup is TRUE, then file is considered TRUE as well.

Note that from v0.8.0 url for the config is build from the R_KITEMS_PATH environment variable and url for the item groups is contained in the YAML config.

Value

a character string.

Examples

```
# item name
name("foo")
```

organize	<i>Sort Items Grammar</i>
----------	---------------------------

Description

Set or get sorting order.

Usage

```
organize(config, item = get_context(), sort)

organized(config, item = get_context())
```

Arguments

config	the config list.
item	the name (id) of the item group.
sort	a character string to set the order (see details).

Details

The organize function accepts a character string for sort because it is primarily intended to deal with the Admin Console (which returns a character string from the user input).

Setting sort = NULL will reset the item sorting!

Ranking depends on the position of the attribute. Wrap attribute name by "desc()" to set descending order.

When item is set in the parent frame, then the attribute can be skipped in the function call.

Value

organize returns a list and organized returns a character string

See Also

`parent.frame()`

Examples

```
## Not run:
organize(config, item = "foo", sort = "date, desc(total)")
organize(config, item = "foo", sort = NULL)

## End(Not run)
```

prepare

Prepare Values

Description

Turn trigger values into tabular data.

Usage

```
prepare(values, config, update = FALSE, item = get_context())
```

Arguments

values	a named list of values (see details).
config	the config list.
update	a logical if values are used in the update workflow (default FALSE).
item	optional (see details), the name (id) of the item group.

Details

values is a named list. The names are used to check the corresponding values vs the data.model (class, default values if the provided ones are not valid). The elements in the list must have either length one or same length as the id element.

When an element has length one but the id has several values, all items corresponding to these ids will be updated with same value. To do so, values will be turned into a data.frame using `as.data.frame` ; for this reason, it's strongly advised to wrap the call into `tryCatch()` as this may fail.

When item is set in the parent frame, then the attribute can be skipped in the function call.

Value

a data.frame

See Also

`parent.frame()`

Examples

```
## Not run:
prepare(values, config, item = "foo")

## End(Not run)
```

refresh

Refresh Behavior Grammar

Description

A set of verbs to manipulate the refreshed attribute(s) of an item.

Usage

```
refresh(config, item = get_context(), ...)

freeze(config, item = get_context(), ...)

refreshed(config, item = get_context())

frozen(config, item = get_context())
```

Arguments

<code>config</code>	the config list.
<code>item</code>	the name (id) of the item group.
<code>...</code>	the name of the attribute(s) to manipulate.

Details

`refresh()` and `freeze()` are setter functions that allow to add or remove attributes from the refreshed ones. `refreshed()` and `frozen()` are getter functions to quickly access the skipped attributes that are refreshed or not during an item update.

Trying to refresh an attribute that is not skipped will be ignored *without* a warning.

When `item` is set in the parent frame, then the attribute can be skipped in the function call.

Value

a list (setters) or a character vector (getters).

See Also

`amend()`, `parent.frame()`

Examples

```
## Not run:
# refresh attribute
config <- design(project = "test",
  item = "foo") |>
extend(item = "foo", attribute = c(name = "date", type = "Date")) |>
skip(item = "foo", "date") |>
refresh(item = "foo", "date")

# refreshed attributes
config |> refreshed(item = "foo")

# freeze attribute
config |> freeze(item = "foo", "date")

# frozen attributes
config |> frozen(item = "foo")

## End(Not run)
```

restore

Restore Files

Description

Restore config, data model or items files.

Usage

```
restore(
  type = c("config", "items", "dm"),
  id = NULL,
  path = Sys.getenv("R_KITEMS_PATH")
)
```

Arguments

<code>type</code>	the type of file to restore (see details).
<code>id</code>	the id for data.model & items.
<code>path</code>	optional, the path to the data.

Details

type accepts the following values:

- "config" (the default)
- "items"
- "dm"

When "config" is used, id is ignored. "dm" is kept for compatibility purpose.

The recommended way to define the path argument is to set the R_KITEMS_PATH environment variable.

In case several backup files exist, the newest one will be restored.

Value

a logical (see [file.copy\(\)](#))

Examples

```
## Not run:
restore()
restore(type = "items", id = "foo")

## End(Not run)
```

reveal

Reveal Items

Description

Apply data model display mask on the items.

Usage

```
reveal(items, config, item = get_context())
```

Arguments

items	the data.frame of the items.
config	the config list.
item	the name (id) of the item group.

Details

The data model display mask is defined at the data.model level. use [hide\(\)](#) or [display\(\)](#) to tune it.

When item is set in the parent frame, then the attribute can be skipped in the function call.

Value

A data.frame.

See Also

[hide()][display()][parent.frame()]

Examples

```
## Not run:  
# assuming config has been declared and  
# items is the data.frame of the items  
item <- "foo"  
items |> reveal(config)  
  
## End(Not run)
```

runExample

Run Shiny Demo Apps

Description

Run Shiny Demo Apps

Usage

```
runExample(example = NA)
```

Arguments

example a character value, with the name of the demo to be launched.

Details

When example is NA, the function will return the names of available apps.

Examples

```
## Not run:  
runExample()  
  
## End(Not run)
```

`shrink`*Shrink Config*

Description

Shrink Config

Usage`shrink(config, ...)`**Arguments**

<code>config</code>	the config list.
<code>...</code>	one or several item or attribute instruction.

Details

Drop an item or an attribute from the config.

The function understands the following instructions:

- `item = foo`
- `attribute = c(item = "foo", name = "total")` See the examples for more details.

It supports multiple instructions.

Note that it is forbidden to delete the 'id' attribute of an item.

Value

a list.

Examples

```
## Not run:
# build baseline
config <- design(project = "test",
  item = "foo") |>
  extend(item = "foo",
    attribute = c(name = "total", type = "integer"))

# drop item
config |>
  shrink(item = "foo")

# drop attribute
config |>
  shrink(attribute = c(item = "foo", name = "total"))
```

```
# multiple instructions
config |>
  shrink(attribute = c(item = "foo", name = "total"),
        item = "foo")

## End(Not run)
```

skip

Skip Behavior Grammar

Description

A set of verbs to manipulate the skipped attribute(s) of an item.

Usage

```
skip(config, item = get_context(), ...)

include(config, item = get_context(), ...)

skipped(config, item = get_context())

included(config, item = get_context())
```

Arguments

config	the config list.
item	the name (id) of the item group.
...	the name of the attribute(s) to manipulate.

Details

`skip()` and `include()` are setter functions that allow to add or remove attributes from the skipped ones. `skipped()` and `included()` are getter functions to quickly access the attributes that are skipped or included in the item form.

In a scenario where items are created programmatically, `included()` may be used to determine what values can be sent to the trigger.

When `item` is set in the parent frame, then the attribute can be skipped in the function call.

Value

a list (setters) or a character vector (getters).

See Also

[amend\(\)](#), [parent.frame\(\)](#)

Examples

```
## Not run:
# skip attribute
config <- design(project = "test",
  item = "foo") |>
extend(item = "foo", attribute = c(name = "date", type = "Date")) |>
skip(item = "foo", "date")

# get skipped attributes
config |> skipped(item = "foo")

# include attribute
config |> include(item = "foo", "date")

# get included attributes
config |> included(item = "foo")

## End(Not run)
```

trigger_event

Trigger Event

Description

Helper function to create events to be passed to kitems module server with the trigger argument.

Usage

```
trigger_event(
  workflow = c("create", "update", "delete"),
  type = c("dialog", "task"),
  values = NULL
)
```

Arguments

workflow	the name of the workflow (create, update or delete).
type	the name of the action to be performed (dialog or task).
values	optional list of values to create, update or delete an item.

Details

The function also adds an event id to make it unique (otherwise sending two times the same request would fail).

For more details see this vignette: `vignette("workflows", package = "kitems")`

Value

A list.

See Also

[kitems\(\)](#)

Examples

```
## Not run:
# create dialog event
trigger_event()

# delete item event (without dialog)
trigger_event(workflow = "delete", type = "task", values = list(id = 1234))

## End(Not run)
```

update

Update Item(s)

Description

Update Item(s)

Usage

```
update(values, items)
```

Arguments

values	a data.frame used to update the items.
items	a data.frame of the items.

Details

The function will drop unmatched id(s) in values.

Value

a data.frame.

Examples

```
## Not run:
update(values, items)
## End(Not run)
```

validate	<i>Validate Value(s)</i>
----------	--------------------------

Description

Validate values and turn them into item(s).

Usage

```
validate(values, data.model, items = NULL, update = FALSE)
```

Arguments

values	a list of named values.
data.model	the data.frame of the data model (see details).
items	an optional data.frame of the items (see details).
update	whether the id attribute should be checked or not (default FALSE).

Details

This function does not accept the data.model element of the config list as an input. It should be turned into a data.frame first, using `yaml_to_dm()`.

data.model should contain the following columns: "name", "type", "default", "class.arg", "values".

In case the data.model values column contains instruction that require data-masking, items will be used as a context to evaluate those instructions.

Value

A data.frame.

Examples

```
## Not run:
# create config
config <- design(project = "test", item = "foo") |>
  extend(item = "foo",
    attribute = list(name = "name", type = "character"),
    attribute = list(name = "quantity", type = "numeric"))

# values
values <- list(name = c("Banana", "Mango"), quantity = c(1, 12))

# call function
validate(values, data.model = yaml_to_dm(config, "name", "type", "default", "class.arg", "values"))

## End(Not run)
```

yaml_to_dm

*Config List To Table***Description**

Turn a config list into a data.frame.

Usage

```
yaml_to_dm(config, ..., item = get_context())
```

Arguments

<code>config</code>	the config list (see details).
<code>...</code>	the columns to include in the output.
<code>item</code>	the name (id) of the item group.

Details

The function is used as a bridge between the config list structure and the expected data.frame input argument for some functions.

It also helps to keep / select the desired columns. Pass the column names (quoted or unquoted) to `...`. Note that if the function is used inside a package (kitems does), then the column names should be quoted or R-CMD Check will throw a note about 'no visible binding for global variable' since the column names are not declared as symbol in the environment.

By default, `config` is the global project config list. `item` is required in this case unless it is already set in the parent frame.

In case the item level config list is provided, it should be up to the data.model's level. Make sure to explicitly set `item = NULL` if it is set in the parent frame.

Value

a data.frame

Examples

```
## Not run:  
# get data.model with name and default  
yaml_to_dm(config, name, default, item = "foo")  
  
## End(Not run)
```

Index

actions_widget(create_widget), 10
adjust, 3
admin, 4
amend, 4
amend(), 23, 33, 37
as_default, 5
attribute_input, 6

backup, 7

check, 8
compute, 9
create_widget, 10

date_slider_widget, 11
decorate, 12
default, 12
delete, 13
delete_widget(create_widget), 10
design, 14
design(), 5, 18
dialog, 15
display(hide), 22
display(), 34
displayed(hide), 22
dm_migrate, 16
dm_to_yaml, 17
dplyr::bind_rows(), 24

enforce, 17
extend, 18
extract, 19

file.copy(), 34
filter_event, 20
filter_event(), 28
form, 21
freeze(refresh), 32
frozen(refresh), 32

has_date_attribute, 21

hidden(hide), 22
hide, 22
hide(), 34

iker::load_data(), 25
iker::save_data(), 25
include(skip), 37
included(skip), 37
insert, 23
is_truthy, 24
item_load, 25
item_save, 25
item_widget, 26
items, 26

kitems, 27
kitems(), 20, 39

name, 29

organize, 30
organize(), 3
organized(organize), 30

parent.frame(), 5, 18, 22, 23, 31–33, 37
prepare, 31

refresh, 32
refreshed(refresh), 32
restore, 33
reveal, 34
runExample, 35

shiny::isTruthy(), 24
shrink, 36
skip, 37
skipped(skip), 37

trigger_event, 38
trigger_event(), 28

update, 39

`update_widget(create_widget)`, [10](#)

`validate`, [40](#)

`yaml_to_dm`, [41](#)

`yaml_to_dm()`, [6](#), [13](#), [40](#)