Package ‘businessPlanR’

August 15, 2023

Type Package

Title Simple Modelling Tools for Business Plans

Description A collection of S4 classes, methods and functions to create and visualize business plans. Different types of cash flows can be defined, which can then be used and tabulated to create profit and loss statements, cash flow plans, investment and depreciation schedules, loan amortization schedules, etc. The methods are designed to produce handsome tables in both PDF and HTML using 'RMarkdown' or 'Shiny'.

Depends R (>= 4.0.0)

Imports methods,kableExtra,knitr

Suggests testthat,rmarkdown,shiny

VignetteBuilder knitr

URL https://www.c3s.cc

BugReports https://github.com/C3S/businessPlanR/issues

License GPL (>= 3)

Encoding UTF-8

LazyLoad yes

Version 0.1-0

Date 2023-08-14

RoxygenNote 7.2.2

Collate '00_environment.R' '01_class_01_operations.R' '01_class_02_transaction.R' '01_class_03_revenue.R' '01_class_04_expense.R' '01_class_05_loan.R' '01_class_06_depreciation.R' '01_class_07_transaction_plan.R' '02_method_barplot.R' '02_method_condense.R' '02_method_get_set_as.R' '02_method_kable_bpR.R' '02_method_kbl_by_types.R' '02_method_model2df.R' '02_method_update_operations.R' '02_method_update_plan.R' 'businessPlanR-package.R' 'businessPlanR_internal.R' 'calc_staff.R' 'fin_needs.R' 'first_last.R' 'growth.R'

'model_node.R' 'nice_numbers.R' 'options.R' 'permalink2list.R'
'regularly.R' 'regularly_delayed.R'

NeedsCompilation no

Author Meik Michalke [aut, cre]

Maintainer Meik Michalke <`meik.michalke@c3s.cc`>

Repository CRAN

Date/Publication 2023-08-15 11:20:09 UTC

R topics documented:

`businessPlanR-package` *Simple Modelling Tools for Business Plans*

Description

A collection of S4 classes, methods and functions to create and visualize business plans. Different types of cash flows can be defined, which can then be used and tabulated to create profit and loss statements, cash flow plans, investment and depreciation schedules, loan amortization schedules, etc. The methods are designed to produce handsome tables in both PDF and HTML using 'RMarkdown' or 'Shiny'.

Details

The DESCRIPTION file:

Package:	businessPlanR
Type:	Package
Version:	0.1-0
Date:	2023-08-14
Depends:	R (>= 4.0.0)
Encoding:	UTF-8
License:	GPL (>= 3)
LazyLoad:	yes
URL:	https://www.c3s.cc

Author(s)

NA

Maintainer: NA

See Also

Useful links:

- `https://www.c3s.cc`
- Report bugs at `https://github.com/C3S/businessPlanR/issues`

`barplot` *Plot business plan transactions*

Description

Plot business plan transactions

Usage

```
barplot(height, ...)

## S4 method for signature 'revenue'
barplot(height, resolution = "month", types = "default", ...)

## S4 method for signature 'expense'
barplot(height, resolution = "month", types = "default", ...)

## S4 method for signature 'operations'
barplot(height, resolution = "month", scope = "profit",
      types = "default", ...)
```

Arguments

`height`	An object of class `operations`, `revenue` or `expense`.
`...`	Any other argument suitable for `barplot()`.
`resolution`	One of `"month"`, `"quarter"`, or `"year"`.
`types`	Character string naming the model types defined by `set_types` to be used.
`scope`	One of `"revenue"`, `"expense"`, `"rev_exp"`, `"profit"`.

Value

See `barplot`.

`calc_staff` *Calculate the number of staff persons necessary to complete a task*

Description

Calculates two values (split by 'boom_months') and returns both in a vector, so that there's never a shortage of staff.

Usage

```
calc_staff(
  task,
  workdays = 205,
  hours = 8,
  rnd = 0.25,
  boom_months = 6,
  boom_pct = 0.5
)
```

Arguments

`task`	The total number of hours to get done in one year.
`workdays`	Numeric, average total working days for a staff person. 205 is the conservative lower end for Germany, see `https://www.deutschlandinzahlen.de/tab/deutschland/arbeitsmarkt/arbeitszeit/arbeitstage`.
`hours`	Number of hours per working day.
`rnd`	Round numbers up to this next fraction of a part-time job.
`boom_months`	Number of months with highest workload, e.g., festival summer
`boom_pct`	Total fraction of `task` that needs to be done during `boom_months`.

Details

Set `boom_months=6` and `boom_pct=.5` to get all hours spread evenly across the year.

Value

A named vector with two elements, `high` (number of staff needed for months with higher workload) and `low` (number of staff needed for months with lower workload).

Examples

```
calc_staff(12328)
```

`condense` *Condense operations objects into neat data frame*

Description

Uses the provided `model` to create a data frame from the `operations` object. Depending on the type of data frame requestet (i.e., default or cashflow) and the temporal resolution (month, quarter or year), various subsets of the overall data in `obj` are returned.

Usage

```
condense(
  obj,
  model = get_model(),
  resolution = c("year", "quarter", "month"),
  keep_types = TRUE,
  cashflow = FALSE,
  cf_init = 0,
  cf_names = c(begin = "Begin", end = "End"),
  years = get_period(obj, years = TRUE),
  digits = 2
)
```

```
## S4 method for signature 'operations'
condense(
  obj,
  model = get_model(),
  resolution = c("year", "quarter", "month"),
  keep_types = TRUE,
  cashflow = FALSE,
  cf_init = 0,
  cf_names = c(begin = "Begin", end = "End"),
  years = get_period(obj, years = TRUE),
  digits = 2
)
```

Arguments

`obj`	An object of class `operations`.
`model`	A named list of named lists describing the stepwise accounting rules for all data in in `obj`.
`resolution`	One of `"month"`, `"quarter"`, or `"year"`.
`keep_types`	Logical, whether the returned data frame should keep the intermediate results for each relevant type of transaction. This will add a column `type` to the data frame.
`cashflow`	Logical, whether the `model` describes a cashflow plan. If `TRUE`, calculations will start with the initial value as specified by `cf_init` and use the result of each period as the starting value of following periods.
`cf_init`	Numeric, used as the initial value for cashflow calculations if `cashflow=TRUE`; i.e., the first beginning cash value.
`cf_names`	Character vector with two entries named `begin` and `end`, used in the resulting table for beginning cash and ending cash.
`years`	Character (or numeric) vector defining the year(s) to be represented in the output. This is intended to be useful for splitting up quarterly or monthly output.
`digits`	Number of digits used for rounding values, disabled if set to `NA`.

Value

A data frame with a subset of the financial transactions of `obj`.

`depreciation,-class` *S4 Class depreciation*

Description

This is a special case of the generic class `transaction`.

Usage

```
## S4 method for signature 'depreciation'
initialize(
  .Object,
  type,
  category,
  name,
  amount,
  obsolete,
  invest_month = format(Sys.Date(), "%Y.%m"),
  method = c("linear", "writedown", "sumofyears", "doubledecline"),
  valid_types = "default",
  value
)
```

Arguments

`.Object`	The object to initialize.
`type`	A character string defining the type of transaction as defined by `valid_types`.
`category`	A character string, custom category for this transaction.
`name`	A character string, custom name or ID for this transaction (i.e., a particular asset that was purchased).
`amount`	Numeric, the amount of money invested into the asset.
`obsolete`	Integer value defining the period (in months) over which the value of the asset diminishes to zero.
`invest_month`	Character string in `YYYY.MM` format, the month of the investment/purchase.
`method`	One of the following, defining the depreciation method: • `"linear"`: The straight line depreciation. This is currently the only implemented option. • `"writedown"`: The written-down value depreciation, not yet implemented. • `"sumofyears"`: The sum-of-years depreciation, not yet implemented. • `"doubledecline"`: The double-declining depreciation, not yet implemented.
`valid_types`	A character string, the model types defined by `set_types` to be used for validation. If `"default"`, pre-defined example types are used.
`value`	A valid data frame to be used as the value slot directly, omitting calculation via `amount`, `obsolete`, `invest_month`, etc.

Details

In contrast to `revenue` or `expense`, the time range of this class of objects is defined by details of the investment as specified. Only when used as an aspect of an `operations` class object, this range is adjusted to fit that particular object.

Slots

`type` A character string, for valid values see `valid_types`. You might use all valid types pre-defined for either `revenue` or `expense`, considering that you might be the depreciation giver or receiver.

`category` A character string, custom category for this depreciation.

`name` A character string, custom name or ID for this depreciation.

`value` Data frame containing an investment plan and allowance for depreciation balance, each month in a row named `YYYY.MM`. The columns are `investment`, `depreciation`, and remaining `value`.

`valid_types` A character string, the model types defined by `set_types` to be used for validation.

Constructor function

Should you need to manually generate objects of this class, the constructor function `depreciation(...)` can be used instead of `new("depreciation", ...)`.

NA

Should you need to manually generate objects of this class, the constructor function `depreciation(...)` can be used instead of `new("depreciation", ...)`.

Examples

```
depreciation_printer <- depreciation(
    type="Depreciation",
    category="Office",
    name="Printer",
    amount=100,
    obsolete=36,
    invest_month="2019.04"
)

# turn depreciation object into an expense
depreciation_as_expense_printer <- as_transaction(
    depreciation_printer,
    to="expense",
    aspect="depreciation"
)
```

`expense,-class` *S4 Class expense*

Description

This is a special case of the generic class `transaction`.

Usage

```
## S4 method for signature 'expense'
initialize(
  .Object,
  type,
  category,
  name,
  per_use,
  missing = c("rep", "interpol", "0"),
  due_month = NA,
  valid_types = "default",
  ...,
  .list = list()
)
```

Arguments

`.Object`	The object to initialize.
`type`	A character string defining the type of transaction as defined by `valid_types`.
`category`	A character string, custom category for this transaction.
`name`	A character string, custom name or ID for this transaction.
`per_use`	If given, the numbers provided via `...` (or `.list`) are not interpreted as the monetary value, but as number of transactions in that month, and the actual fiscal value is calculated by multiplying it with the value given here.
`missing`	One of `"rep"`, `"interpol"`, or `"0"`. This defines how gaps are filled: If `"rep"`, present values are repeated until the next valid value; if `"interpol"`, missing values are interpolated using `approx`; if `"0"`, missing values are set to zero.
`due_month`	Character vector to define months where transactions are due. This argument causes previous amounts to be cumulated and thereby postponed to the given month of a year. Combined with e.g. `.list` this makes it easier to turn monthly amounts into quarterly ones.
`valid_types`	A character string, the model types defined by `set_types` to be used for validation. If `"default"`, pre-defined example types are used.
`...`	Numeric values named in `YYYY.MM` format, defining the transaction amount for a particular month. The resulting object will automatically cover all months from the earliest to the latest among all given values.
`.list`	An alternative to `...` if the values are already present as a list. If both are given, their values will be merged into one list.

Slots

`type` A character string, for valid values see `valid_types`.

`category` A character string, custom category for this expense.

`name` A character string, custom name or ID for this expense.

`value` Data frame containing all expenses, each month in a column named `YYYY.MM`.

`valid_types` A character string, the model types defined by `set_types` to be used for validation.

Constructor function

Should you need to manually generate objects of this class, the constructor function `expense(...)` can be used instead of `new("expense", ...)`.

Examples

```
exp_2019_2021 <- expense(
   type="Goods",
   category="Merch",
   name="T-Shirts",
   "2019.03"=65,
   "2019.07"=170,
   "2020.02"=210,
   "2020.08"=312,
   "2021.01"=450,
   "2021.06"=600,
   "2021.10"=720
)
```

`fin_needs` *Estimate capital requirement from cash flow*

Description

To avoid cash flow issues, this function takes a data frame as returned by `condense` with `cashflow=TRUE` to calculate the amount of financial needs per time resolution.

Usage

```
fin_needs(
  cashflow_df,
  resolution = c("year", "quarter", "month"),
  row_names = c("Financial needs", "Cumulative")
)
```

Arguments

`cashflow_df`	Data frame as returned by `condense` with `cashflow=TRUE`.
`resolution`	One of `"month"`, `"quarter"`, or `"year"`. Must be identical to the value used with the call to condense!
`row_names`	Character vector of two, names for the rows of the resulting data frame. The first represents financial need per time period (column), the second is cumulated over all columns.

Details

Only negative values are returned, so the row sum can be used as an estimate of the overall financial demand for the given period of time.

Value

A data frame with two rows and columns depending on `resolution` and period covered by `cashflow_df`.

`first_last` *Shortcut for lists with steady transactions.*

Description

Generates a list of two elements, first and last month of the full years range, both with the same value specified.

Usage

```
first_last(years, value)
```

Arguments

`years`	Integer vector, at least two elements, the range of years to cover.
`value`	The transaction amount that is assumed to remain unchanged over all `years`.

Details

You can use this in combination with the `.list` argument of `expense`, `revenue`, and `transaction`.

Value

A list with two elements named after the first and last month of the `years`' range in `YYYY.MM` format.

Examples

```
expense(
    type="Operation",
    category="Bank",
    name="Accounting",
    missing="rep",
    .list=first_last(2022:2025, value=20)
)
```

get_revenue *Getter/setter methods for businessPlanR objects*

Description

These methods return the requested slots from objects of class `operations`, `revenue`, `expense`, `transaction_plan`, `loan` or `depreciation`, or, in case of their `<-` counterparts, replace slots with a given `value`.

Usage

```
get_revenue(
  obj,
  drop_nonyear_cols = FALSE,
  resolution = c("month", "quarter", "year"),
  only_type,
  not_type
)

## S4 method for signature 'operations'
get_revenue(
  obj,
  drop_nonyear_cols = FALSE,
  resolution = c("month", "quarter", "year"),
  only_type,
  not_type
)

get_expense(
  obj,
  drop_nonyear_cols = FALSE,
  resolution = c("month", "quarter", "year"),
  only_type,
  not_type
)

## S4 method for signature 'operations'
get_expense(
  obj,
  drop_nonyear_cols = FALSE,
  resolution = c("month", "quarter", "year"),
  only_type,
  not_type
)

get_value(
  obj,
  drop_nonyear_cols = FALSE,
  resolution = c("month", "quarter", "year")
)

## S4 method for signature 'transaction_plan'
get_value(
  obj,
  drop_nonyear_cols = FALSE,
  resolution = c("month", "quarter", "year")
)

## S4 method for signature 'loan'
get_value(
  obj,
  drop_nonyear_cols = FALSE,
  resolution = c("month", "quarter", "year")
)

## S4 method for signature 'depreciation'
get_value(
  obj,
  drop_nonyear_cols = FALSE,
  resolution = c("month", "quarter", "year")
)

## S4 method for signature 'revenue'
get_value(
  obj,
  drop_nonyear_cols = FALSE,
  resolution = c("month", "quarter", "year")
)

## S4 method for signature 'expense'
get_value(
  obj,
  drop_nonyear_cols = FALSE,
  resolution = c("month", "quarter", "year")
)

get_loans(obj, as_data_frame = TRUE)

## S4 method for signature 'operations'
get_loans(obj, as_data_frame = TRUE)

get_plan(obj, type, category, name, valid_types = "default")

## S4 method for signature 'transaction_plan'
get_plan(obj, type, category, name, valid_types = "default")
```

```
get_period(obj, years = FALSE)

## S4 method for signature 'operations'
get_period(obj, years = FALSE)

## S4 method for signature 'transaction_plan'
get_period(obj, years = FALSE)

## S4 method for signature 'loan'
get_period(obj, years = FALSE)

## S4 method for signature 'depreciation'
get_period(obj, years = FALSE)

get_depreciation_plan(obj, as_data_frame = TRUE)

## S4 method for signature 'operations'
get_depreciation_plan(obj, as_data_frame = TRUE)

get_plan_type(obj)

## S4 method for signature 'transaction_plan'
get_plan_type(obj)

get_misc(obj, name)

## S4 method for signature 'operations'
get_misc(obj, name)

set_misc(obj, name) <- value

## S4 replacement method for signature 'operations'
set_misc(obj, name) <- value

list_plans(obj)

## S4 method for signature 'transaction_plan'
list_plans(obj)

get_sum(obj)

## S4 method for signature 'revenue'
get_sum(obj)

## S4 method for signature 'expense'
get_sum(obj)
```

```
as_transaction(obj, to, aspect, valid_types = "default", type)

## S4 method for signature 'loan'
as_transaction(
  obj,
  to = c("revenue", "expense"),
  aspect = c("interest", "balance_start", "principal", "total", "cumsum",
    "balance_remain"),
  valid_types = "default",
  type
)

## S4 method for signature 'depreciation'
as_transaction(
  obj,
  to = c("revenue", "expense"),
  aspect = c("investment", "depreciation", "value"),
  valid_types = "default",
  type
)
```

Arguments

`obj`	An object of class `operations`, `revenue`, `expense`, `transaction_plan`, `loan` or `depreciation`.
`drop_nonyear_cols`	Logical, whether to drop or keep columns specifying type, category or name or rows.
`resolution`	One of `"month"`, `"quarter"`, or `"year"`.
`only_type`	Optional character vector, if given, only rows with matching type are returned. Overrides `not_type` if both are provided.
`not_type`	Optional character vector, if given, only rows with types not matching the vector entries are returned.
`as_data_frame`	Logical, if `FALSE` returns an object of class `transaction_plan` instead of a data frame.
`type`	Character string, a valid type name for the resulting object.
`category`	A character string, custom category for this transaction.
`name`	Character or integer, specifying which element to get/set. If missing, the whole list is returned/replaced.
`valid_types`	A character string, the model types defined by `set_types` to be used for validation. If `"default"`, pre-defined example types are used.
`years`	Logical, if `TRUE` doesn't return the period vector but a vector of all years in the period.
`value`	A value to assign to the object.
`to`	Character string, the transaction class to coerce into.
`aspect`	Character string, the row/column of the input objects's `value` data frame to use in the resulting object. All additional data are silently dropped.

Details

If `as_transaction(..., aspect="balance_start")` is being called on a loan object, only the initial value (and perhaps growth instead of declining values) is used, e.g. as revenue for calculations.

Value

Depending on the method, either a data frame or a numeric value.

`growth` *Growth of a numeric vector*

Description

Calculates the differences between consecutive values in a numeric vector.

Usage

```
growth(x, round = c("round", "ceiling", "floor"), digits = 0, init = x[1])
```

Arguments

`x`	A numeric vector.
`round`	One of `"round"` (invokes `round` on `x` before calculation), `"ceiling"` (calling `ceiling`), or `"floor"` (calling `floor` instead of `round`, respectively).
`digits`	Integer, passed to `round` if `round="round"`.
`init`	Numeric, the initial value to compare the first element of `x` to.

Value

A numeric vector the same length as `x`.

Examples

```
growth(c(1,10,12,15,122))
```

`kable_bpR` *Format table from condensed objects*

Description

This method uses the `kableExtra` package for table formatting.

Usage

```
kable_bpR(
  obj,
  model = get_model(),
  resolution = c("year", "quarter", "month"),
  keep_types = TRUE,
  detailed = FALSE,
  cashflow = FALSE,
  currency = "€",
  DIY = FALSE,
  longtable_clean_cut = TRUE,
  font_size = NULL,
  latex_options = "striped",
  stripe_color = "gray!6",
  years = get_period(obj, years = TRUE),
  detail_names = c(revenue = "Revenue", expense = "Exepense"),
  detail_colors = c(color = "white", background = "grey"),
  cf_init = 0,
  cf_names = c(begin = "Begin", end = "End"),
  space = c(html = "&#8239;", latex = "\\,"),
  detail_width,
  ...
)

## S4 method for signature 'operations'
kable_bpR(
  obj,
  model = get_model(),
  resolution = c("year", "quarter", "month"),
  keep_types = TRUE,
  detailed = FALSE,
  cashflow = FALSE,
  currency = "€",
  DIY = FALSE,
  longtable_clean_cut = TRUE,
  font_size = NULL,
  latex_options = "striped",
  stripe_color = "gray!6",
  years = get_period(obj, years = TRUE),
  detail_names = c(revenue = "Revenue", expense = "Exepense"),
  detail_colors = c(color = "white", background = "grey"),
  cf_init = 0,
  cf_names = c(begin = "Begin", end = "End"),
  space = c(html = "&#8239;", latex = "\\,"),
  detail_width,
  ...
)

## S4 method for signature 'loan'
kable_bpR(
  obj,
  resolution = c("month", "quarter", "year"),
  currency = "€",
  DIY = FALSE,
  font_size = NULL,
  latex_options = "striped",
  stripe_color = "gray!6",
  loan_names = c(balance_start = "Balance start", interest = "Interest", principal =
    "Principal", total = "Total", cumsum = "Cumulated", balance_remain =
    "Balance remain"),
  space = c(html = "&#8239;", latex = "\\,"),
  ...
)

## S4 method for signature 'transaction_plan'
kable_bpR(
  obj,
  resolution = c("month", "quarter", "year"),
  keep_types = FALSE,
  currency = "€",
  DIY = FALSE,
  longtable_clean_cut = TRUE,
  font_size = NULL,
  latex_options = "basic",
  stripe_color = "gray!6",
  years = get_period(obj, years = TRUE),
  dep_names = c(investment = "Investment", depreciation = "Depreciation", value =
    "Value", sum = "Sum"),
  loan_names = c(balance_start = "Balance start", interest = "Interest", principal =
    "Principal", total = "Total", cumsum = "Cumulated", balance_remain =
    "Balance remain", sum = "Sum"),
  space = c(html = "&#8239;", latex = "\\,"),
  zeroes = c(html = "#C0C0C0", latex = "gray!25"),
  ...
)
```

Arguments

`obj`	An object of class `operations` or `loan`.
`model`	A named list of named lists describing the stepwise accounting rules for all data in in `obj`.
`resolution`	One of `"month"`, `"quarter"`, or `"year"`.
`keep_types`	Logical, whether the returned data frame should keep the intermediate results for each relevant type of transaction. This will add a column `type` to the data frame.
`detailed`	Logical, supersedes `keep_types`. If `TRUE`, the table includes detailed information all the way down to types, categories, and transaction names.
`cashflow`	Logical, whether the `model` describes a cash flow plan. If `TRUE`, calculations will start with the initial value as specified by `cf_init` and use the result of each period as the starting value of following periods. This only works if `detailed=FALSE`.
`currency`	Character defining a currency symbol.
`DIY`	Logical, if `TRUE` returns the `kable` object prior to any row collapsing, column specs or kable styling, so you can apply all of those as you wish.
`longtable_clean_cut`	Passed to `collapse_rows`.
`font_size`	Passed to `kable_styling`.
`latex_options`	Passed to `kable_styling`.
`stripe_color`	Passed to `kable_styling`.
`years`	Character (or numeric) vector defining the year(s) to be represented in the output. This is intended to be useful for splitting up quarterly or monthly output.
`detail_names`	A named character vector with two entries, `revenue` and `expense`, defining the global names used for the two transaction classes in the data frame if `detailed=TRUE`.
`detail_colors`	A named character vector with two entries, `color` and `background`, defining the color scheme for position headlines (revenue and expense). Only relevant if `detailed=TRUE`.
`cf_init`	Numeric, used as the initial value for cash flow calculations if `cashflow=TRUE`; i.e., the first beginning cash value.
`cf_names`	Character vector with two entries named `begin` and `end`, used in the resulting table for beginning cash and ending cash.
`space`	Character, a space definition to put between currency and value.
`detail_width`	Optional vector of length 3, if given defined the width of the three categorial columns, `Type`, `Category`, and `Name`.
`...`	Additional arguments passed on to `kbl`.
`loan_names`	Like `dep_names` but with seven named entries, `balance_start`, `interest`, `principal`, `total`, `cumsum`, `balance_remain`, and `sum`, for loan plans, respectively.
`dep_names`	A named character vector with four entries, `investment`, `depreciation`, `value`, and `sum`, used in table to describe the rows of each depreciation item, with `sum` only being used in the final set of rows showing a summary over all items.
`zeroes`	Named character vector defining the text color to use for zero amounts, for both LaTeX and HTML format.

Value

An object of class `kable`.

`kbl_by_types` *Format table from collection of types of operations objects*

Description

This method uses the `kableExtra` package for table formatting.

Usage

```
kbl_by_types(
  obj,
  types,
  resolution = c("year", "quarter", "month"),
  currency = "€",
  digits = 0,
  DIY = FALSE,
  font_size = NULL,
  latex_options = "striped",
  stripe_color = "gray!6",
  years = get_period(obj, years = TRUE),
  sum_names = c(subtotal = "Subtotal", total = "Total"),
  type_colors = c(color = "white", background = "grey"),
  space = c(html = "&#8239;", latex = "\\,"),
  ...
)

## S4 method for signature 'operations'
kbl_by_types(
  obj,
  types,
  resolution = c("year", "quarter", "month"),
  currency = "€",
  digits = 0,
  DIY = FALSE,
  font_size = NULL,
  latex_options = "striped",
  stripe_color = "gray!6",
  years = get_period(obj, years = TRUE),
  sum_names = c(subtotal = "Subtotal", total = "Total"),
  type_colors = c(color = "white", background = "grey"),
  space = c(html = "&#8239;", latex = "\\,"),
  ...
)
```

Arguments

obj	An object of class operations or loan.
types	A named character vector of types to fetch from obj and print in the resulting table. Names must be the type names, their value must be one of "revenue" or "expense" so the method knows what to use in case identical type names are defined for both.
resolution	One of "month", "quarter", or "year".
currency	Character defining a currency symbol.
digits	Integer, round values to number of digits.
DIY	Logical, if TRUE returns the kable object prior to any row packing, specs or kable styling, so you can apply all of those as you wish.
font_size	Passed to kable_styling.
latex_options	Passed to kable_styling.
stripe_color	Passed to kable_styling.
years	Character (or numeric) vector defining the year(s) to be represented in the output. This is intended to be useful for splitting up quarterly or monthly output.
sum_names	A named character vector with two entries, subtotal and total, to be used in the resulting table for those values.
type_colors	A named character vector with two entries, color and background, defining the color scheme for type headlines.
space	Character, a space definition to put between currency and value.
...	Additional arguments passed on to kbl.

Value

An object of class kable.

---

loan,-class *S4 Class loan*

Description

This is a special case of the generic class transaction.

Usage

```
## S4 method for signature 'loan'
initialize(
  .Object,
  type,
  category,
  name,
  amount,
  period,
  interest,
  first_month = format(Sys.Date(), "%Y.%m"),
  schedule = c("annuity", "amortization", "maturity"),
  due_month = NA,
  valid_types = "default",
  value
)
```

Arguments

`.Object`	The object to initialize.
`type`	A character string defining the type of transaction as defined by `valid_types`.
`category`	A character string, custom category for this transaction.
`name`	A character string, custom name or ID for this transaction.
`amount`	Numeric, the amount of money loaned.
`period`	Integer, number of months to fully repay the loan.
`interest`	Numeric, the nominal interest rate per annum (a value between 0 and 1).
`first_month`	Character string in `YYYY.MM` format, defining the initial date of the loan.
`schedule`	One of the following, defining the repayment schedule: • `"annuity"`: Equal rates of total repayment over `period`, thereby interest is relatively higher and principal payment relatively lower at the beginning. • `"amortization"`: Repayment of equal rates of principal payment with decreasing interest and total payments over `period`. • `"maturity"`: Repayment of the full loan amount at the end of `period`, until then only payment of interest.
`due_month`	Integer value defining the first month of principal repayment. The selected `schedule` will not begin before this month, until then only interest rates are due. Beware that this is a different behaviour of this argument compared to `transaction`.
`valid_types`	A character string, the model types defined by `set_types` to be used for validation. If `"default"`, pre-defined example types are used.
`value`	A valid data frame to be used as the value slot directly, omitting calculation via `amount`, `period`, `interest`, etc.

Details

In contrast to `revenue` or `expense`, the time range of this class of objects is defined by details of the loan as specified. Only when used as an aspect of an `operations` class object, this range is adjusted to fit that particular object.

Slots

`type` A character string, for valid values see `valid_types`. You might use all valid types predefined for either `revenue` or `expense`, considering that you might be the loan giver or receiver.

`category` A character string, custom category for this loan.

`name` A character string, custom name or ID for this loan.

`value` Data frame containing an amortization schedule for the loan, each month in a row named `YYYY.MM`. It has a row for each month and the columns `balance_start`, `interest`, `principal`, `total`, `cumsum`, and `balance_remain`.

`valid_types` A character string, the model types defined by `set_types` to be used for validation.

Constructor function

Should you need to manually generate objects of this class, the constructor function `loan(...)` can be used instead of `new("loan", ...)`.

NA

Should you need to manually generate objects of this class, the constructor function `loan(...)` can be used instead of `new("loan", ...)`.

Examples

```
loan_2019 <- loan(
    type="Interest",
    category="Bank",
    name="New office",
    amount=10000,
    period=60,
    interest=0.075,
    first_month="2019.04",
    schedule=c("amortization")
)

# turn loan object into an expense
loan_as_expense_2019 <- as_transaction(
    loan_2019,
    to="expense",
    aspect="interest"
)
```

`model2df` *Convert model from list to data frame*

Description

Converting a model from list format into a data frame makes it easier to work with nested sub-positions, and to check the model for completeness.

Usage

```
model2df(model = get_model(), factorize = TRUE)

## S4 method for signature 'list'
model2df(model = get_model(), factorize = TRUE)
```

Arguments

`model`	A named list describing a transaction model.
`factorize`	Logical, whether columns not representing a transaction type should be returned as a factor.

Details

The list provided must have named entries which form the top level of the transaction model. Values are in turn a list with optional named arguments:

- `subpos` A named list, nested sub-position to this level, structured like any higher level position.
- `carry` Name of a previous position of the same level, its value is used as the starting value of this position.
- `revenue` Character vecotor of valid revenue types, their values are added to the position total.
- `expense` Character vecotor of valid expense types, their values are subtracted from the position total.

Value

A data frame, representing the model structure that was defined with `table_model`.

`nice_numbers` *Format numbers in nice layout*

Description

Uses `format` with some customized defaults. It's being called by `kable_bpR`.

Usage

```
nice_numbers(
  x,
  prefix,
  suffix,
  digits = 0L,
  width = NULL,
  nsmall = digits,
  space = c(html = "&#8239;", latex = "\\,")
)
```

Arguments

`x`	The numeric value to format. Can be a single number, numeric vector, matrix, or data frame.
`prefix`	An optional symbol to prepend, ignored if missing.
`suffix`	An optional symbol to append, ignored if missing.
`digits`	See `round`.
`width`	See `format`.
`nsmall`	See `format`.
`space`	Named character vector, a space definition to put between prefix/suffix and value. Defaults to a thin space for both, LaTeX and HTML. If you use provide one character, that one is used regardless of the output environment.

Value

A formatted character string.

Examples

```
nice_numbers(12345.6789, suffix="€", digits=2)
```

`operations,-class` *S4 Class operations*

Description

This class is used for objects that contain all transactions of the business plan.

Slots

`period` A character vector defining beginning and end of the time period covered by the business plan. Values can either be a vector of two in `YYYY.MM` format, or a numeric vector of full fiscal years which will automatically be transformed into character.

`revenue` Data frame containing `type`, `category`, `name`, and all revenues, each month in a column named `YYYY.MM`. If these are not covering `period` exactly, missing values will be set to zero.

`expense` Data frame containing all expenses, data structure like the `revenue` slot.

`loan` Data frame, basically the `plan` slot as in `transaction_plan` with `plan_type="loan"`.

`depreciation` Data frame, like `loan`, but with `plan_type="depreciation"`, respectively.

`misc` A list to keep miscellaneous data or information for documentation or re-use.

Constructor function

Should you need to manually generate objects of this class, the constructor function `operations(...)` can be used instead of `new("operations", ...)`.

Examples

```
rev_2019_2021_merch <- revenue(
   type="Sale",
   category="Merch",
   name="T-Shirts",
   "2019.01"=100,
   "2019.08"=267,
   "2020.03"=344,
   "2020.09"=549,
   "2021.02"=770,
   "2021.07"=1022,
   "2021.12"=1263
)
rev_2019_2021_rec <- revenue(
   type="Sale",
   category="Records",
   name="Albums",
   "2019.01"=220,
   "2019.08"=234,
   "2020.03"=221,
   "2020.09"=354,
   "2021.02"=276,
   "2021.07"=285,
   "2021.12"=311
)
rev_2019_2021_inv <- revenue(
   type="Invest income",
   category="Rent",
   name="Studio",
   "2019.01"=120,
   "2019.08"=234,
   "2020.03"=321,
   "2020.09"=454,
   "2021.02"=376,
   "2021.07"=385,
   "2021.12"=211
)
exp_2019_2021_merch <- expense(
   type="Goods",
   category="Merch",
   name="T-Shirts",
   "2019.01"=65,
   "2019.07"=170,
   "2020.02"=210,
   "2020.08"=312,
   "2021.01"=450,
   "2021.06"=600,
   "2021.12"=720
)
exp_2019_2021_rec <- expense(
   type="Goods",
   category="Records",
    name="Pressing",
    "2019.01"=1860,
    "2019.02"=0,
    "2020.08"=600,
    "2020.09"=0,
    "2021.12"=0
)

op_2019_2021 <- operations(
    period=c("2019.01", "2021.12") # alternative: 2019:2021
)
update_operations(op_2019_2021) <- rev_2019_2021_merch
update_operations(op_2019_2021) <- exp_2019_2021_merch
update_operations(op_2019_2021) <- rev_2019_2021_rec
update_operations(op_2019_2021) <- exp_2019_2021_rec
update_operations(op_2019_2021) <- rev_2019_2021_inv
```

`permalink2list` *Turn a Shiny permalink into a list*

Description

The Shiny package can generate permalinks of its web apps, making it possible to share individual configurations of the app with others. This function translated such a permalink into a named list, so you can use the configuration also in R code.

Usage

```
permalink2list(permalink, prefix = ".*\\?_inputs_&")
```

Arguments

`permalink`	Character string, the actual URL with arguments copied from the Shiny app as-is.
`prefix`	Character string or regular expression, should capture everything up to the first argument name. This is the part that will be discarded.

Details

When this package was written, we also wrote a Shiny web app for it but separated the actual calculations from the app's code. This allowed us to use the same functions and objects in RMarkdown. We were discussing the numbers in the web tool using permalinks, and finally transferred the calculations to the PDF version.

To transfer the configuration from the web app to the markdown document, this function discards the URL prefix and splits the arguments into a named list that behaves like the input object commonly used in Shiny apps.

Value

A named list with one element for each argument in `permalink`.

Examples

```
permalink2list(
  paste0(
    "https://example.com/businessPlanR/?_inputs_&salary=50000",
    "&loan_interest=3.22&loan_period=7&loan_due=2&years=%5B%222022%22%2C%222026%22%5D"
  )
)
```

`regularly` *Generate list of repeating financial transactions*

Description

For all years defined, generates a list of values as defined by `pa` and due at the given month. The result can be used as input for the `.list` argument of `expense`, `revenue`, and `transaction`.

Usage

```
regularly(
  years,
  pa,
  month = "01",
  last = 0,
  first = 0,
  merge = list(),
  digits = 2
)
```

Arguments

`years`	Integer vector, the range of years to cover.
`pa`	A vector with values for each year. This amounts to the total sum for the respective year.
`month`	Character, but numeric description of a month in "MM" format when to account the values of `pa`. If you provide more than a single month here, e.g., quarterly payments, the amounts defined by `pa` are divided the number of months.
`last`	Defines the final entry, last month of the last year. It can be either a numeric value (taken as-is), `"rep"` (repeats the last value of pa), or `"none"` to omit adding a last month (e.g., to later merge with results of another call to this function). Only used if month is not "12".
`first`	Defines how to treat years if January was included in in `month`. This could be desired for merging, but problematic if you want to create a new transaction object. Valid values are the same as for `last` except `"rep"`.

`merge`	Another list of values to be merged with the results, can be used for nested calls of this function to generate more complex patterns.
`digits`	Number of digits used for rounding when `month` is more than one entry.

Value

A list of monthly transactions named in "YYYY.MM" scheme (`regularly_delayed`).

Examples

```
expense(
    type="Operation",
    category="Insurance",
    name="Electronics",
    missing="0",
    .list=regularly(
        years=2021:2025,
        pa=rep(111.11, 5),
        month="01",
        last=0
    )
)
```

`regularly_delayed` *Generate list of repeating financial transactions with delayed starting month*

Description

In case you only know the annual sum of transactions for given years but also that they don't begin in January of the first year, you can use the function `regularly_delayed` to split the sums to be used in revenue or expense objects that acknowledge the delay. It extends `regularly`.

Usage

```
regularly_delayed(years, pa, start_month = 1)

delayed(pa, start_month = 1)
```

Arguments

`years`	See `regularly`.
`pa`	See `regularly`.
`start_month`	Integer number, the month of the first revenue/expense. All earlier monthly transactions will be 0 and the sum for the respective year divided by the number months left for that year.

Details

The delayed function assumes pa to be a total value for a full year, but does not distribute it evenly over the active months, but rather subtracts any amount that would have been due before start_month.

Value

Either a list of monthly transactions named in "YYYY.MM" scheme (regularly_delayed), or vector of the same length as pa (delayed).

Examples

```
# say you earn 3000 each year, but payment starts in September
# calculate payment sums
delayed_2019_2021 <- delayed(
   pa=rep(3000, 3),
   start_month=9
)

# now use the result to caclulate monthly amounts
delayed_monthly_2019_2021 <- regularly_delayed(
   years=2019:2021,
   pa=delayed_2019_2021,
   start_month=9
)
```

---

revenue,-class *S4 Class revenue*

Description

This is a special case of the generic class transaction.

Usage

```
## S4 method for signature 'revenue'
initialize(
  .Object,
  type,
  category,
  name,
  per_use,
  missing = c("rep", "interpol", "0"),
  due_month = NA,
  valid_types = "default",
  ...,
  .list = list()
)
```

Arguments

`.Object`	The object to initialize.
`type`	A character string defining the type of transaction as defined by `valid_types`.
`category`	A character string, custom category for this transaction.
`name`	A character string, custom name or ID for this transaction.
`per_use`	If given, the numbers provided via `...` (or `.list`) are not interpreted as the monetary value, but as number of transactions in that month, and the actual fiscal value is calculated by multiplying it with the value given here.
`missing`	One of `"rep"`, `"interpol"`, or `"0"`. This defines how gaps are filled: If `"rep"`, present values are repeated until the next valid value; if `"interpol"`, missing values are interpolated using `approx`; if `"0"`, missing values are set to zero.
`due_month`	Character vector to define months where transactions are due. This argument causes previous amounts to be cumulated and thereby postponed to the given month of a year. Combined with e.g. `.list` this makes it easier to turn monthly amounts into quarterly ones.
`valid_types`	A character string, the model types defined by `set_types` to be used for validation. If `"default"`, pre-defined example types are used.
`...`	Numeric values named in `YYYY.MM` format, defining the transaction amount for a particular month. The resulting object will automatically cover all months from the earliest to the latest among all given values.
`.list`	An alternative to `...` if the values are already present as a list. If both are given, their values will be merged into one list.

Slots

`type` A character string, for valid values see `valid_types`.

`category` A character string, custom category for this revenue.

`name` A character string, custom name or ID for this revenue.

`value` Data frame containing all revenues, each month in a column named `YYYY.MM`.

`valid_types` A character string, the model types defined by `set_types` to be used for validation.

Constructor function

Should you need to manually generate objects of this class, the constructor function `revenue(...)` can be used instead of `new("revenue", ...)`.

Examples

```
rev_2019_2021 <- revenue(
   type="Sale",
   category="Merch",
   name="T-Shirts",
   "2019.03"=100,
   "2019.08"=267,
   "2020.03"=344,
   "2020.09"=549,
   "2021.02"=770,
   "2021.07"=1022,
   "2021.10"=1263
)
```

set_types *Define valid types of revenues and expenses*

Description

These functions change the globally available options of the running R session. Its values define types of transactions you want to be able to use in your business plan.

Usage

```
set_types(types, class = c("revenue", "expense"), name = "default")

get_types(
  name = "default",
  class = c("revenue", "expense"),
  names_only = FALSE
)

get_model()
```

Arguments

`types`	Named list, one entry for each type. Values define the color to use in plots.
`class`	One of `"revenue"` or `"expense"`.
`name`	Character string, giving the set of types a name. You can use this to have multiple sets of types simultaneously in the same session.
`names_only`	Logical, whether the full list or only the names of defined types should be returned.

Details

The getter functions return a list of default types if none have been defined so far.

Value

`set_types` is a wrapper for `options` and adds/replaces a list called `name` to the `businessPlanR` option of the running session. `get_types` returns the list from the `businessPlanR` option. `get_model` just returns the internal definition of default operations model as a list.

table_model — *Define a model node for business plan tables*

Description

Tool to define a (possibly nested) model for generating tables for our business plan. The "model" is in fact a nested list.

Usage

```
table_model(..., valid_types, check_carry = TRUE)

model_node(carry, ..., revenue, expense)
```

Arguments

`...`	Optional named lists of nodes (`table_model`) or nested sub-nodes (`model_node`), like subsections of this section. You can use `model_node` recursive to define these named nodes. Just don't forget to give each a unique name.
`valid_types`	Optional character string, the name of the type set to use for checking if all used revenue and expense names are actually valid.
`check_carry`	Logical, if `TRUE` all node names used und the nested list will be looked up if they are referenced by `carry` somewhere down the line.
`carry`	Optional character string, the name of another already defined named list, probably at the same level. The sum of that list will then be used as the initial value for the calculation of this node.
`revenue`	Optional character vector defining names defined as class revenue via `set_types`.
`expense`	Optional character vector defining names defined as class expense via `set_types`.

Details

If you define nested levels, you want to probably only want to combine this node with `carry` and neither `revenue` nor `expense`.

Value

A nested, named list.

Examples

```
my_model <- table_model(
  "Basic Income"=model_node(
    revenue="Sale"
  ),
  "Basic Costs"=model_node(
    carry="Basic Income",
    expense=c(
      "Goods",
      "Operation"
    )
  ),
  valid_types="default",
  check_carry=TRUE
)
```

transaction,-class *S4 Class transaction*

Description

This is a generic class used by subclasses revenue and expense.

Usage

```
## S4 method for signature 'transaction'
initialize(
  .Object,
  type,
  category,
  name,
  per_use,
  missing = c("rep", "interpol", "0"),
  due_month = NA,
  valid_types = "default",
  ...,
  .list = list()
)
```

Arguments

`.Object`	The object to initialize.
`type`	A character string defining the type of transaction as defined by `valid_types`.
`category`	A character string, custom category for this transaction.
`name`	A character string, custom name or ID for this transaction.
`per_use`	If given, the numbers provided via `...` (or `.list`) are not interpreted as the monetary value, but as number of transactions in that month, and the actual fiscal value is calculated by multiplying it with the value given here.
`missing`	One of `"rep"`, `"interpol"`, or `"0"`. This defines how gaps are filled: If `"rep"`, present values are repeated until the next valid value; if `"interpol"`, missing values are interpolated using `approx`; if `"0"`, missing values are set to zero.
`due_month`	Character vector to define months where transactions are due. This argument causes previous amounts to be cumulated and thereby postponed to the given month of a year. Combined with e.g. `.list` this makes it easier to turn monthly amounts into quarterly ones.

`valid_types`	A character string, the model types defined by `set_types` to be used for validation. If `"default"`, pre-defined example types are used.
`...`	Numeric values named in `YYYY.MM` format, defining the transaction amount for a particular month. The resulting object will automatically cover all months from the earliest to the latest among all given values.
`.list`	An alternative to `...` if the values are already present as a list. If both are given, their values will be merged into one list.

Slots

`type` A character string, valid values are defined by the subclasses.

`category` A character string, custom category for this transaction.

`name` A character string, custom name or ID for this transaction.

`value` Data frame containing all transactions, each month of each year in a column named `YYYY.MM`.

`valid_types` A character string, the model types defined by `set_types` to be used for validation.

Constructor function

Should you need to manually generate objects of this class, the constructor function `transaction(...)` can be used instead of `new("transaction", ...)`. It uses the same arguments like the `initialize()` method.

You should either provide exactly one named value for each month of the full scope of the respective business plan, or at least two, representing the first and last value.

Missing values

How missing values are dealt with depends on the value of the `missing` parameter. By default (`missing="rep"`) a given value will be repeated until a later value comes, which will then be repeated further on. That is, you can define a staring value and only have to provide updated values for months that differ from the previous value. Alternatively, `missing="interpol"` will interpolate missing values linearly, and `missing="0"` fills missing values with zeroes.

`transaction_plan,-class`

S4 Class transaction_plan

Description

This is a container class for multiple objects of either class `depreciation` or `loan`, similar to `operations` for revenues and expenses. Its main data frame stores each transaction object in multiple rows. Investment have three rows, `investment`, `depreciation`, and remaining `value`, while loans have six named `balance_start`, `interest`, `principal`, `total`, `cumsum`, and `balance_remain`, repectively. This makes it easier to create nice overview tables via `kable_bpR`.

Details

The data frame has four meta data columns, `type`, `category`, `name`, and `part`, followed by a column for each month covered by any of the contained `transaction` objects. The first three columns take their values from the respective object, while the fourth, `part`, defines the rows as explained earlier.

Slots

`plan_type` One of `"depreciation"` or `"loan"`, defining which type of transactions are accumulated in the object.

`plan` A data frame with three rows for each `depreciation` or six for each `loan` class object added to it, e.g., via `update_plan`.

Constructor function

Should you need to manually generate objects of this class, the constructor function `transaction_plan(...)` can be used instead of `new("transaction_plan", ...)`.

Examples

```
depreciation_printer <- depreciation(
    type="Depreciation",
    category="Office",
    name="Printer",
    amount=100,
    obsolete=36,
    invest_month="2019.04"
)
depreciation_laptop <- depreciation(
    type="Depreciation",
    category="Office",
    name="Laptop",
    amount=1200,
    obsolete=36,
    invest_month="2019.02"
)
# initialize an empty plan
dep_plan <- transaction_plan()
# add your assets to the plan
update_plan(dep_plan) <- depreciation_printer
update_plan(dep_plan) <- depreciation_laptop
```

`update_operations<-` *Update operations objects*

Description

You can use this method to add or replace transactions to an existing object of class `operations`.

Usage

```
update_operations(obj, cut_to_period = TRUE, warning = FALSE, as_transaction) <- value

## S4 replacement method for signature 'operations'
update_operations(obj, cut_to_period = TRUE, warning = FALSE,
      as_transaction) <- value
```

Arguments

`obj`	An object of class `operations`.
`cut_to_period`	Logical, whether to adjust the data of `value` to the period covered by `obj`. This means that missing months will be added with zero values, and months that lie beyond the covered period will be dropped. This only affects objects of class `revenue` and `expense`.
`warning`	Logical, if `TRUE` shows a warning when `cut_to_period=TRUE` and months are adjusted.
`as_transaction`	Optional list of vectors of arguments for `value` of class `loan` or `depreciation`, as used by `as_transaction`. If given, the object provided as `value` will also be processed as if `as_transaction` was also called. This is repeated for each vector of arguments.
`value`	An object of either class `revenue`, `expense`, `loan`, `depreciation`, or `transaction_plan`.

Value

An updated object of class `operations`.

`update_plan<-` *Update transaction_plan objects*

Description

You can use this method to add or replace `depreciation` or `loan` class objects to/in an existing object of class `transaction_plan`.

Usage

```
update_plan(obj) <- value

## S4 replacement method for signature 'transaction_plan'
update_plan(obj) <- value
```

Arguments

`obj`	An object of class `transaction_plan`.
`value`	An object of class `depreciation` or `loan`.

Value

An updated object of class `transaction_plan`.

Examples

```
depreciation_printer <- depreciation(
    type="Depreciation",
    category="Office",
    name="Printer",
    amount=100,
    obsolete=36,
    invest_month="2019.04"
)
depreciation_laptop <- depreciation(
    type="Depreciation",
    category="Office",
    name="Laptop",
    amount=1200,
    obsolete=36,
    invest_month="2019.02"
)
# initialize an empty plan
dep_plan <- transaction_plan()
# add your assets to the plan
update_plan(dep_plan) <- depreciation_printer
update_plan(dep_plan) <- depreciation_laptop
```

Index