Package 'rb3'

May 9, 2026

Title Download and Parse Public Data Released by B3 Exchange

Description Download and parse public files released by B3 and convert them into useful formats and data structures common to data analysis practitioners.

Version 0.1.0

License MIT + file LICENSE

Depends R (>= 4.1.0),

Imports bizdays, stringr, stringi, cli, readr, dplyr, arrow, httr, jsonlite, purrr, rlang, yaml, digest, base64enc, XML, R.utils, lubridate, DBI, RSQLite

Suggests testthat, knitr, rmarkdown, ggplot2, covr, scales, magrittr, tibble, withr, tidyr, vcr (>= 0.6.0)

Collate 'types.R' 'fields.R' 'registry.R' 'template.R' 'meta.R' 'read-marketdata.R' 'fetch-marketdata.R' 'download-marketdata.R' 'scraper-futures.R' 'scraper-cotahist.R' 'scraper-yc.R' 'scraper-indexes.R' 'readers.R' 'downloaders.R' 'rb3-package.R' 'zzz.R'

BugReports https://github.com/ropensci/rb3/issues

URL https://github.com/ropensci/rb3, https://ropensci.github.io/rb3/

VignetteBuilder knitr

RoxygenNote 7.3.2

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation no

Author Wilson Freitas [aut, cre], Marcelo Perlin [aut]

Maintainer Wilson Freitas <wilson.freitas@gmail.com>

Repository CRAN

Date/Publication 2025-07-07 09:20:02 UTC

Contents

`code2month` *Convert Maturity Code to Corresponding Month*

Description

This function takes a character string representing the maturity code of a futures contract and returns the corresponding month as an integer. It supports both the new and old maturity code formats used in futures contracts.

Usage

```
code2month(x)
```

Arguments

`x`	A character vector with the maturity code(s) of futures contracts. The codes can be either a single letter (e.g., "F", "G", "H", ...) representing the new code format or a three-letter abbreviation (e.g., "JAN", "FEV", "MAR", ...) representing the old code format.

Details

The function distinguishes between two maturity code formats:

- The **new code format** uses a single letter (e.g., "F" = January, "G" = February, etc.).
- The **old code format** uses a three-letter abbreviation (e.g., "JAN" = January, "FEV" = February, etc.).

Value

A vector of integers corresponding to the months of the year, where 1 = January, 2 = February, ..., 12 = December.

Examples

```
code2month(c("F", "G", "H", "J", "K", "M", "N", "Q", "U", "V", "X", "Z"))
code2month(c("JAN", "FEV", "MAR", "NOV", "DEZ"))
```

---

cotahist-extracts *Filtering data from COTAHIST datasets*

Description

A set of functions that implement filters to obtain organized and useful data from the COTAHIST datasets.

Usage

```
cotahist_filter_equity(x)

cotahist_filter_etf(x)

cotahist_filter_bdr(x)

cotahist_filter_unit(x)

cotahist_filter_fii(x)

cotahist_filter_fidc(x)

cotahist_filter_fiagro(x)

cotahist_filter_index(x)

cotahist_filter_equity_options(x)

cotahist_filter_index_options(x)

cotahist_filter_fund_options(x)
```

Arguments

x	A cotahist dataset

Details

The functions bellow return data from plain instruments, stocks, funds and indexes.

- `cotahist_filter_equity()` returns data for stocks and UNITs.
- `cotahist_filter_etf()` returns data for ETFs.
- `cotahist_filter_bdr()` returns data for BDRs.
- `cotahist_filter_unit()` returns data exclusively for UNITs.
- `cotahist_filter_index()` returns data for indices. The index data returned by `cotahist_filter_index()` corresponds to option expiration days, meaning there is only one index quote per month.
- `cotahist_filter_fii()`, `cotahist_filter_fidc()`, and `cotahist_filter_fiagro()` return data for funds.

The functions bellow return data related to options, from equities, indexes and funds (ETFs).

- `cotahist_filter_equity_options()` returns data for stock options.
- `cotahist_filter_index_options()` returns data for index options, currently only for IBOVESPA.
- `cotahist_filter_funds_options()` returns data for fund options, currently only for ETFs.

The dataset provided must have at least the columns `isin`, `instrument_market`, `bdi_code` and `specification_code`. A combination of these columns is used to filter the desired data.

Value

A dataframe containing the requested market data.

Examples

```
## Not run:
df <- cotahist_get() |> cotahist_filter_equity()

## End(Not run)
## Not run:
df <- cotahist_get() |> cotahist_filter_etf()

## End(Not run)
## Not run:
df <- cotahist_get() |> cotahist_filter_bdr()

## End(Not run)
## Not run:
df <- cotahist_get() |> cotahist_filter_unit()

## End(Not run)
## Not run:
df <- cotahist_get() |> cotahist_filter_fii()
```

```
## End(Not run)
## Not run:
df <- cotahist_get() |> cotahist_filter_fidc()

## End(Not run)
## Not run:
df <- cotahist_get() |> cotahist_filter_fiagro()

## End(Not run)
## Not run:
df <- cotahist_get() |> cotahist_filter_index()

## End(Not run)
## Not run:
df <- cotahist_get() |> cotahist_filter_equity_options()

## End(Not run)
## Not run:
df <- cotahist_get() |> cotahist_filter_index_options()

## End(Not run)
## Not run:
df <- cotahist_get() |> cotahist_filter_fund_options()

## End(Not run)
```

`cotahist_get` *Access COTAHIST datasets*

Description

The COTAHIST files are available with daily, monthly, and yearly data. Therefore, the datasets correspond to these periods (`daily`, `monthly`, `yearly`). See `download_marketdata` and `read_marketdata` for instructions on how to download the files and create the datasets.

Usage

```
cotahist_get(type = c("yearly", "monthly", "daily"))
```

Arguments

`type`	A string specifying the dataset to be used: `"daily"`, `"monthly"`, or `"yearly"`.

Details

The COTAHIST files contain historical quotation data (*Cotações Históricas*) for stocks, stock options, stock forward contracts, ETFs, ETF options, BDRs, UNITs, REITs (FIIs - *Fundos Imobiliários*), FIAGROs (*Fundos da Agroindústria*), and FIDCs (*Fundos de Direitos Creditórios*). These

files from B3 hold the oldest available information. The earliest annual file available dates back to 1986. However, it is not recommended to use data prior to 1995 due to the monetary stabilization process in 1994 (Plano Real).

Note that the prices in the files are not adjusted for corporate actions. As a result, only ETF series can be used without issues.

Before using the dataset, it is necessary to download the files using the `download_marketdata` function and create the datasets with the `read_marketdata` function.

Value

An `arrow_dplyr_query` or `ArrowObject`, representing a lazily evaluated query. The underlying data is not collected until explicitly requested, allowing efficient manipulation of large datasets without immediate memory usage. To trigger evaluation and return the results as an R `tibble`, use `collect()`.

Examples

```
## Not run:
# get all data to the year of 2001
meta <- download_marketdata("b3-cotahist-yearly", year = 2001)
read_marketdata(meta)
ds_yearly <- cotahist_get()

# Earliest available annual file: 1986
# Recommended starting point: 1995 (after Plano Real)

## End(Not run)
## Not run:
# To obtain data from January 2, 2014, the earliest available date:
meta <- download_marketdata("b3-cotahist-daily", refdate = as.Date("2014-01-02"))
read_marketdata(meta)
ds_daily <- cotahist_get("daily")

## End(Not run)
## Not run:
# Once you download more dates, the data downloaded before remains stored and you can filter
# any date you want.
meta <- download_marketdata("b3-cotahist-daily", refdate = as.Date("2014-01-03"))
read_marketdata(meta)
df_daily <- cotahist_get("daily") |>
  filter(refdate == "2014-01-03") |>
  collect()

## End(Not run)
```

`download_marketdata` *Download Raw Market Data Files from B3*

Description

Downloads and caches financial market datasets from B3 (Brazilian Stock Exchange) based on predefined templates. Handles file downloading and caching.

Usage

```
download_marketdata(meta)
```

Arguments

`meta`	A metadata object.

Details

The function follows this workflow:

1. Checks if requested data exists in cache
2. Downloads data if needed (based on template specifications)
3. Manages file compression and storage
4. Maintains metadata for tracking and verification

Files are organized in the `rb3.cachedir` as follows:

- Data: Gzipped files in 'raw/' directory, named by file's checksum

Templates are YAML documents that define:

- Download parameters and methods
- Data reading instructions
- Dataset structure (columns, types)

Templates can be found using `list_templates()` and retrieved with `template_retrieve()`.

Value

Returns a meta object containing the downloaded file's metadata:

- template - Name of the template used
- download_checksum - Unique hash code for the download
- download_args - Arguments used for the download
- downloaded - Path to the downloaded file
- created - Timestamp of file creation
- is_downloaded - Whether the file was successfully downloaded

- is_processed - Whether the file was successfully processed
- is_valid - Whether the file is valid

The `meta` object can be interpreted as a ticket for the download process. It contains all the necessary information to identify the data, if it has been downloaded, if it has been processed, and ince it is processed, if the downloaded file is valid.

See Also

- `template_meta_create_or_load` for creating a metadata object
- `list_templates` for listing available data templates
- `template_retrieve` for retrieving specific template details

`read_marketdata`, `rb3.cachedir`

Examples

```
## Not run:
# Create metadata for daily market data
meta <- template_meta_create_or_load("b3-cotahist-daily",
  refdate = as.Date("2024-04-05")
)
# Download using the metadata
meta <- download_marketdata(meta)

# For reference rates
meta <- template_meta_create_or_load("b3-reference-rates",
  refdate = as.Date("2024-04-05"),
  curve_name = "PRE"
)
# Download using the metadata
meta <- download_marketdata(meta)

## End(Not run)
```

`fetch_marketdata` *Fetch and process market data*

Description

Downloads market data based on a template and parameter combinations, then reads the data into a database.

Usage

```
fetch_marketdata(
  template,
  force_download = FALSE,
  reprocess = FALSE,
  throttle = FALSE,
  ...
)
```

Arguments

template	A character string specifying the market data template to use
force_download	A logical value indicating whether to force downloading files even if they already exist in the cache (default is FALSE). If TRUE, the function will download files again even if they were previously downloaded.
reprocess	A logical value indicating whether to reprocess files even if they are already processed (default is FALSE). If TRUE, the function will reprocess the files in the input layer, even if they were previously processed.
throttle	A logical value indicating whether to throttle the download requests (default is FALSE). If TRUE, a 1-second delay is introduced between requests to avoid overwhelming the server.
...	Named arguments that will be expanded into a grid of all combinations to fetch data for

Details

This function performs three main steps:

1. Downloads market data files by creating all combinations of the provided parameters.
2. Processes the downloaded files by reading them into the input layer of the database.
3. Creates the staging layer if configured in the template.

Progress indicators are displayed during both steps, and warnings are shown for combinations that failed to download or produced invalid files.

The throttle parameter is useful for avoiding server overload and ensuring that the requests are sent at a reasonable rate. If set to TRUE, a 1-second delay is introduced between each download request.

The force_download parameter allows you to re-download files even if they already exist in the cache. This can be useful if you want to ensure that you have the latest version of the data or if the files have been modified on the server.

The reprocess parameter allows you to reprocess files even if they have already been processed. This can be useful if you want to ensure that the data is up-to-date.

Examples

```
## Not run:
fetch_marketdata("b3-cotahist-yearly", year = 2020:2024)
fetch_marketdata("b3-cotahist-daily", refdate = bizseq("2025-01-01", "2025-03-10", "Brazil/B3"))
fetch_marketdata("b3-reference-rates",
  refdate = bizseq("2025-01-01", "2025-03-10", "Brazil/B3"),
  curve_name = c("DIC", "DOC", "PRE")
)
fetch_marketdata("b3-indexes-historical-data",
  throttle = TRUE, index = c("IBOV", "IBXX", "IBXL"),
  year = 2000:2025
)

## End(Not run)
```

---

futures_get *Retrieves B3 Futures Settlement Prices*

Description

This function fetches settlement price data for B3 futures contracts. This function retrieves futures settlement prices from the B3 dataset, adding a `symbol` column that combines `commodity` and `maturity_code`.

Usage

```
futures_get()
```

Value

An `arrow_dplyr_query` or `ArrowObject`, representing a lazily evaluated query. The underlying data is not collected until explicitly requested, allowing efficient manipulation of large datasets without immediate memory usage. To trigger evaluation and return the results as an R `tibble`, use `collect()`.

The returned data includes the following columns:

- `refdate`: Reference date for the prices.
- `symbol`: Futures contract symbol, created by concatenating the commodity code and the maturity code.
- `commodity`: Commodity code of the futures contract.
- `maturity_code`: Maturity code of the futures contract.
- `previous_price`: Closing price from the previous trading day.
- `price`: Current price of the futures contract.
- `price_change`: Price variation compared to the previous day.
- `settlement_value`: Settlement value of the futures contract.

Source

B3 Market Data

Examples

```
## Not run:
df_fut <- futures_get() |> filter(refdate == Sys.Date()) |> collect()
head(df_fut)

## End(Not run)
```

`indexes-portfolio` *Retrieve Portfolio of B3 Indexes*

Description

These functions fetch the current and theoretical portfolio of B3 indexes using predefined dataset templates. The data is retrieved from the datasets "b3-indexes-current-portfolio" and "b3-indexes-theoretical-portfolio".

Usage

```
indexes_current_portfolio_get()

indexes_theoretical_portfolio_get()
```

Value

An `arrow_dplyr_query` or `ArrowObject`, representing a lazily evaluated query. The underlying data is not collected until explicitly requested, allowing efficient manipulation of large datasets without immediate memory usage. To trigger evaluation and return the results as an R `tibble`, use `collect()`.

Examples

```
## Not run:
template_dataset("b3-indexes-current-portfolio", layer = 2) |>
  filter(index %in% c("SMLL", "IBOV", "IBRA")) |>
  collect()

## End(Not run)

## Not run:
template_dataset("b3-indexes-theoretical-portfolio", layer = 2) |>
  filter(index == "IBOV") |>
  collect()

## End(Not run)
```

indexes_composition_get *Retrieve Composition of B3 Indexes*

Description

This function fetches the composition of B3 indexes. It uses the template dataset "b3-indexes-composition" to retrieve the data.

Usage

```
indexes_composition_get()
```

Value

A data frame containing the columns:

update_date The date when the data was last updated.

symbol The symbol of the asset.

indexes The indexes associated with the asset.

An `arrow_dplyr_query` or `ArrowObject`, representing a lazily evaluated query. The underlying data is not collected until explicitly requested, allowing efficient manipulation of large datasets without immediate memory usage. To trigger evaluation and return the results as an R `tibble`, use `collect()`.

Examples

```
## Not run:
  indexes_composition <- indexes_composition_get()
  head(indexes_composition)

## End(Not run)
```

indexes_get *Get B3 indexes available*

Description

Gets B3 indexes available.

Usage

```
indexes_get()
```

Value

a character vector with symbols of indexes available

Examples

```
## Not run:
indexes_get()

## End(Not run)
```

`indexes_historical_data_get` *Get historical data from B3 indexes*

Description

Fetches historical data from B3 indexes.

Usage

```
indexes_historical_data_get()
```

Value

An `arrow_dplyr_query` or `ArrowObject`, representing a lazily evaluated query. The underlying data is not collected until explicitly requested, allowing efficient manipulation of large datasets without immediate memory usage. To trigger evaluation and return the results as an R `tibble`, use `collect()`.

Examples

```
## Not run:
fetch_marketdata("b3-indexes-historical-data", index = "IBOV", year = 2001:2010)
indexes_historical_data_get() |>
 filter(index == "IBOV", refdate >= as.Date("2001-01-01"), refdate <= as.Date("2010-12-31"))

## End(Not run)
```

list_templates *List Available Templates*

Description

Retrieves all templates registered in the template registry and returns their properties as a tibble.

Usage

```
list_templates()
```

Value

A tibble with the following columns:

Description The description of the template

Template The template identifier

Examples

```
list_templates()
```

maturitycode2date *Convert Maturity Code to Date*

Description

This function converts a vector of maturity codes into actual dates.

Usage

```
maturitycode2date(x, expr = "first day", refdate = NULL)
```

Arguments

x	a character vector with three letters string that represent maturity of futures contracts.
expr	a string which indicates the day to use in maturity date, default is "first day". See `bizdays::getdate` for more details on this argument
refdate	reference date to be passed. It is necessary to convert old maturities like JAN0, that can be Jan/2000 or Jan/2010. If `refdate` is greater that 2001-01-01 JAN0 is converted to Jan/2010, otherwise, Jan/2000.

Value

A vector of dates corresponding to the input maturity codes. Convert Maturity Code to Date

This function converts a vector of maturity codes into actual dates. Get the corresponding maturity date for the three characters string that represent maturity of futures contracts.

a Date vector with maturity dates

Examples

```
maturitycode2date(c("F22", "F23", "G23", "H23", "F45"), "first day")
maturitycode2date(c("F23", "K35"), "15th day")
maturitycode2date(c("AGO2", "SET2"), "first day")
```

`meta_db_connection` *Returns a SQLite Database Connection for the RB3 Package Metadata*

Description

This function provides a consistent way to connect to the SQLite database used for RB3 package metadata. It returns an existing connection if one is already established and valid, or creates a new connection if needed.

Usage

```
meta_db_connection()
```

Details

The function first checks if a valid connection already exists in the package registry. If not, it establishes a new connection to a SQLite database located in the configured database folder and stores this connection in the package registry.

Value

A SQLite connection object for metadata storage

Examples

```
# Get a connection to the RB3 metadata database
con <- meta_db_connection()
```

rb3.cachedir rb3.cachedir *Option*

Description

The rb3.cachedir option is used to specify the directory where cached data will be stored when using the rb3 package. This option allows users to define a custom directory for caching, which can improve performance by avoiding repeated downloads or computations.

Details

Setting the rb3.cachedir **Option:**

To set the rb3.cachedir option, use the options() function and provide the desired directory path as a string. For example:

```
# Set the cache directory to a custom path
options(rb3.cachedir = "/path/to/your/cache/directory")
```

Replace "/path/to/your/cache/directory" with the actual path where you want the cached data to be stored.

Viewing the Current Value of rb3.cachedir**:**

To check the current value of the rb3.cachedir option, use the getOption() function:

```
# View the current cache directory
getOption("rb3.cachedir")
```

This will return the path to the directory currently set for caching, or NULL if the option has not been set.

Notes:

- Ensure that the specified directory exists and is writable.
- If the rb3.cachedir option is not set, the package use a temporary directory (base::tempdir()).

Examples

```
# Set the cache directory
options(rb3.cachedir = "~/rb3_cache")

# Verify the cache directory
cache_dir <- getOption("rb3.cachedir")
print(cache_dir)

# In this example, the cache directory is set to `~/rb3_cache`, and the value
# is then retrieved and printed to confirm the setting.
```

rb3_bootstrap *Initialize the rb3 package cache folders*

Description

This function sets up the necessary directory structure for caching rb3 data. It creates a main cache folder and three subfolders: 'raw', 'meta', and 'db'. The folder paths are stored in the rb3 registry for later use.

Usage

```
rb3_bootstrap()
```

Details

The function first checks if the 'rb3.cachedir' option is set. If not, it uses a subfolder in the temporary directory. It creates the main cache folder and the three subfolders if they don't already exist, then stores their paths in the rb3 registry.

The cache structure includes:

- raw folder - for storing raw downloaded data
- db folder - for database files

Examples

```
## Not run:
options(rb3.cachedir = "~/rb3-cache")
rb3_bootstrap()

## End(Not run)
```

read_marketdata *Read and parse raw market data files downloaded from the B3 website.*

Description

B3 provides various files containing valuable information about traded assets on the exchange and over-the-counter (OTC) market. These files include historical market data, trading data, and asset registration data for stocks, derivatives, and indices. This function reads these files and parses their content according to the specifications defined in a template.

Usage

```
read_marketdata(meta)
```

Arguments

meta	A metadata object.

Details

This function reads the downloaded file and parses its content according to the specifications and schema defined in the template associated with the meta object. The template specifies the file format, column definitions, and data types.

The parsed data is then written to a partitioned dataset in Parquet format, stored in a directory structure based on the template name and data layer. This directory is located within the db subdirectory of the rb3.cachedir directory. The partitioning scheme is also defined in the template, allowing for efficient querying of the data using the arrow package.

If an error occurs during file processing, the function issues a warning, removes the downloaded file and its metadata, and returns NULL.

Value

Returns a meta object containing the downloaded file's metadata:

- template - Name of the template used
- download_checksum - Unique hash code for the download
- download_args - Arguments used for the download
- downloaded - Path to the downloaded file
- created - Timestamp of file creation
- is_downloaded - Whether the file was successfully downloaded
- is_processed - Whether the file was successfully processed
- is_valid - Whether the file is valid

The meta object can be interpreted as a ticket for the download process. It contains all the necessary information to identify the data, if it has been downloaded, if it has been processed, and ince it is processed, if the downloaded file is valid.

See Also

list_templates

rb3.cachedir

download_marketdata

Examples

```
## Not run:
# Create metadata for daily market data
meta <- template_meta_create_or_load("b3-cotahist-daily",
  refdate = as.Date("2024-04-05")
)
# Download using the metadata
meta <- download_marketdata(meta)
```

```
meta <- read_marketdata(meta)

# For reference rates
meta <- template_meta_create_or_load("b3-reference-rates",
  refdate = as.Date("2024-04-05"),
  curve_name = "PRE"
)
# Download using the metadata
meta <- download_marketdata(meta)
meta <- read_marketdata(meta)

## End(Not run)
```

`superdataset` *Enhanced Dataset Creation*

Description

To maximize the utility of B3's existing datasets, several functions integrate data from multiple sources to generate specialized datasets for specific analytical needs. For instance, `cotahist_equity_options_superset()` combines data from COTAHIST datasets (`b3-cotahist-yearly`, `b3-cotahist-monthly`, and `b3-cotahist-daily`) and Reference Rates (`b3-reference-rates`) to construct a dataset containing stock options data. This dataset includes details such as the closing price of the underlying stock, its ticker symbol, and the applicable interest rate at option expiration. This comprehensive data enables users to perform option pricing and calculate implied volatility.

Usage

```
cotahist_options_by_symbols_get(symbols)

yc_brl_with_futures_get(refdate)

yc_usd_with_futures_get(refdate)

yc_ipca_with_futures_get(refdate)
```

Arguments

`symbols`	list of symbols to extract market data from the COTAHIST dataset.
`refdate`	A Date object specifying the reference date for which to retrieve data

Details

The functions `cotahist_equity_options_superset()`, `cotahist_funds_options_superset()`, `cotahist_index_options_superset()`, and `cotahist_options_by_symbol_superset()` use information from the COTAHIST datasets (`b3-cotahist-yearly`, `b3-cotahist-monthly`, and `b3-cotahist-daily`) and Reference Rates (`b3-reference-rates`) and return a dataframe containing stock option data,

including the closing price of the underlying stocks, the ticker of the underlying asset, and the interest rate at the option's expiration. The returned dataframe contains the following columns: "refdate", "symbol", "type", "symbol_underlying", "strike_price", "maturity_date", "r_252", "close", "close_underlying", "volume", "trade_quantity", and "traded_contracts".

`cotahist_options_by_symbol_superset()` returns the same dataset but filtered for the specified asset ticker.

Returned objects preserve lazy evaluation whenever possible and avoid being collected until the last possible moment. Exceptions occur when operations cannot be performed using Arrow's operators — in such cases, data will be collected and `data.frames` will be returned. Please refer to the documentation to identify the situations where this behavior applies.

These functions retrieve yield curve data merged with corresponding futures contract information:

- `yc_brl_with_futures_get()`: BRL nominal rates with DI1 futures contracts
- `yc_usd_with_futures_get()`: USD rates (Cupom Cambial) with DDI futures contracts
- `yc_ipca_with_futures_get()`: Real (inflation-indexed) rates with DAP futures contracts

These functions combine data from B3 Reference Rates (`b3-reference-rates`) and Futures Settlement Prices (`b3-futures-settlement-prices`) to create comprehensive yield curve datasets. The resulting data highlights key vertices along the curve with their corresponding futures contracts, providing insight into the term structure of interest rates.

Each function requires a specific reference date to prevent excessive memory usage and ensure optimal performance.

Value

The function `cotahist_options_by_symbol_superset()` return an object that inherits from a `arrow_dplyr_query` since it tries to preserve the lazy evaluation and avoids collecting the data before its return.

The functions `yc_brl_with_futures_get()`, `yc_usd_with_futures_get()` and `yc_ipca_with_futures_get()` return a `data.frame` containing the yield curve data merged with futures contract information. The data is pre-collected (not lazy) and includes all columns from the respective yield curve function plus a `symbol` column identifying the corresponding futures contract.

Examples

```
## Not run:
date <- preceding(Sys.Date() - 1, "Brazil/ANBIMA")
bova_options <- cotahist_options_by_symbols_get("BOVA11") |> filter(refdate == date)
petr_options <- cotahist_options_by_symbols_get(c("PETR4", "PETR3")) |> filter(refdate == date)

## End(Not run)

## Not run:
# Get data for the last business day
date <- preceding(Sys.Date() - 1, "Brazil/ANBIMA")

# Retrieve BRL yield curve with DI1 futures
brl_curve <- yc_brl_with_futures_get(date)
head(brl_curve)
```

```
# Retrieve USD yield curve with DDI futures
usd_curve <- yc_usd_with_futures_get(date)

# Retrieve inflation-indexed yield curve with DAP futures
ipca_curve <- yc_ipca_with_futures_get(date)

## End(Not run)
```

`template_dataset` *Access a Dataset for a Template*

Description

This function provides access to a dataset associated with a specific template. It retrieves the dataset stored in the database folder for the given template and layer, using the schema defined in the template configuration.

Usage

```
template_dataset(template, layer = NULL)
```

Arguments

`template`	The template identifier or template object. This specifies the dataset to retrieve.
`layer`	The layer of the dataset to access (e.g., "input" or "staging"). If `NULL`, the layer `"input` is used.

Details

The `template_dataset()` function is a generic function that dispatches to specific methods based on the type of the `template` argument. It retrieves the dataset by resolving the template using `template_retrieve()` if the input is a template identifier.

Value

An Arrow dataset object representing the data for the specified template and layer.

Examples

```
## Not run:
# Access the dataset for the "b3-reference-rates" template
ds <- template_dataset("b3-reference-rates")

# Access the dataset for the "b3-reference-rates" template in the staging layer
ds <- template_dataset("b3-reference-rates", layer = "staging")

# Query the dataset
```

```
ds |>
  dplyr::filter(refdate > as.Date("2023-01-01")) |>
  dplyr::collect()

## End(Not run)
```

`template_meta_load` *Create or Load Template Metadata*

Description

These functions attempt to create new template metadata objects or load existing ones.

Usage

```
template_meta_load(template, ...)

template_meta_new(template, ...)

template_meta_create_or_load(template_name, ...)
```

Arguments

`template`	An object representing the template. Can be of class `character` (template ID) or `template` (template object).
`...`	Additional arguments to be passed to the metadata creation process. These should include all required arguments specified in the template definition.
`template_name`	Character string specifying the template ID.

Details

The `template_meta_load` function checks that all required arguments for the template are provided. If any required arguments are missing, it will abort with an error of class "error_template_missing_args". It raises an error if the template is not found or if the required arguments are not provided. The error message will indicate which arguments are missing.

The `template_meta_new` function checks that all required arguments for the template are provided. If any required arguments are missing, it will abort with an error of class "error_template_missing_args". It raises an error if the template already exists in the database.

The `template_meta_create_or_load` function is a safe way to create or load template metadata. It first attempts to create a new metadata object using the provided arguments. If that fails (e.g., if the template already exists), it will attempt to load the existing metadata object. This is useful for ensuring that you always have access to the latest metadata for a given template.

Value

The created or loaded template metadata object

Examples

```
## Not run:
# Create or load metadata for a template
meta <- template_meta_create_or_load("b3-reference-rates",
  refdate = as.Date("2024-04-05"),
  curve_name = "PRE"
)

## End(Not run)
```

---

template_retrieve *Retrieve a template by its name*

Description

This function retrieves a template identified by its name.

Usage

```
template_retrieve(template_name)
```

Arguments

template_name	The name identifying the template to retrieve.

Value

The template associated with the given name.

---

yc_xxx_get *Retrieve Yield Curve Data*

Description

These functions retrieve yield curve data, either for all available curves (`yc_get`) or specifically for:

- the nominal rates curve (`yc_brl_get`).
- the nominal rates curve for USD in Brazil - Cupom Cambial Limpo (`yc_usd_get`).
- the real rates curve (`yc_ipca_get`).

Usage

```
yc_get()

yc_brl_get()

yc_ipca_get()

yc_usd_get()
```

Details

The yield curve data is downloaded from the B3 website https://www2.bmf.com.br/pages/portal/bmfbovespa/lumis/lum-taxas-referenciais-bmf-ptBR.asp. See the Curve Manual in this link https://www.b3.com.br/data/files/8B/F5/11/68/5391F61043E561F6AC094EA8/Manual_de_Curvas.pdf for more details.

Value

An `arrow_dplyr_query` or `ArrowObject`, representing a lazily evaluated query. The underlying data is not collected until explicitly requested, allowing efficient manipulation of large datasets without immediate memory usage. To trigger evaluation and return the results as an R `tibble`, use `collect()`.

The returned data includes the following columns:

- `curve_name`: Identifier of the yield curve (e.g., "PRE", "DOC", "DIC").
- `refdate`: Reference date of the curve.
- `forward_date`: Maturity date associated with the interest rate.
- `biz_days`: Number of business days between `refdate` and `forward_date`.
- `cur_days`: Number of calendar days between `refdate` and `forward_date`.
- `r_252`: Annualized interest rate based on 252 business days.
- `r_360`: Annualized interest rate based on 360 calendar days.

Examples

```
## Not run:
df <- yc_get() |>
  filter(curve_name == "PRE") |>
  collect()

## End(Not run)
## Not run:
df_yc <- yc_brl_get() |>
  filter(refdate == Sys.Date()) |>
  collect()
head(df_yc)

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

```
df_yc_ipca <- yc_ipca_get() |>
  filter(refdate == Sys.Date()) |>
  collect()
head(df_yc_ipca)

## End(Not run)
## Not run:
df_yc_usd <- yc_usd_get() |>
  filter(refdate == Sys.Date()) |>
  collect()
head(df_yc_usd)

## End(Not run)
```

Index