

Package ‘tradesimr’

October 8, 2026

Title Execution and Simulation Engine for Trading Strategies

Version 0.18.7

Description An R-native trading simulation package with a C++ execution core that turns strategy intentions and explicit orders into simulated trades, positions, cash, profit and loss, risk, and performance outputs under configurable execution, margin, funding, and cost assumptions. The package provides historical replay, incremental exchange stepping, durable event tables, append-only agent command logs, registered assets, per-agent shared-cash cross-margin live accounts, AI agent competitors, scheduled live-feed stepping, strategy-backed AI agents with diagnostics, calibrated and coordinated multi-asset market simulation with static covariance, AR-GARCH, factor, and regime models, durable per-feed simulation state, profile-aware heterogeneous inventory and margin execution with atomic mixed-profile order groups, optional portfolio-margin enforcement through a multi-asset C++ step kernel, local live-service APIs, import/export helpers, separate replay, live-state, and agent dashboard exports, and installed local orchestration scripts. It is designed to consume signals, order intents, or target exposure decisions from compatible strategy packages and market data from compatible adapters.

Depends R ($\geq 4.2.0$)

Imports data.table, Rcpp, R6

LinkingTo Rcpp

Suggests testthat, fst, ggplot2, jsonlite, lubridate, plumber, strategyr, zoo

URL <https://github.com/OliverLDS/tradesimr>

BugReports <https://github.com/OliverLDS/tradesimr/issues>

License MIT + file LICENSE

Encoding UTF-8

Config/roxygen2/version 7.2.3

RoxygenNote 7.2.3

NeedsCompilation yes

Author Oliver Zhou [aut, cre]

Maintainer Oliver Zhou <oliver.yxzhou@gmail.com>

Repository CRAN

Date/Publication 2026-10-08 10:40:02 UTC

Contents

as_market_bars	5
as_target_positions	6
PaperTrader	6
PaperTradingPlatform	8
sim_account	12
sim_agents_step	13
sim_agent_add	13
sim_agent_command_schema	14
sim_agent_dashboard_export	14
sim_agent_dashboard_open	15
sim_agent_rankings	15
sim_agent_remove	16
sim_agent_set_status	16
sim_assets	17
sim_asset_add	17
sim_asset_remove	18
sim_backtest	19
sim_bond_schedules	21
sim_bond_schedule_add	21
sim_calendar_expected_bars	22
sim_calendar_holidays	23
sim_calendar_is_open	23
sim_calendar_settlement_timestamp	24
sim_calendar_spec	24
sim_cancel_order	25
sim_cash_ledger	26
sim_cross_asset_risk	26
sim_dashboard_export	27
sim_dashboard_open	27
sim_events	28
sim_exchange_account	28
sim_exchange_account_state	29
sim_exchange_accrue_carry	29
sim_exchange_add_bars	30
sim_exchange_calendarize_bars	30
sim_exchange_calendar_exception	31
sim_exchange_cancel_order	32
sim_exchange_cash_adjust	32
sim_exchange_cash_balances	33
sim_exchange_convert_cash	34

sim_exchange_corporate_action	34
sim_exchange_dashboard	35
sim_exchange_export_events	36
sim_exchange_future_roll	36
sim_exchange_fx_rate	37
sim_exchange_load	38
sim_exchange_new	38
sim_exchange_new_events	39
sim_exchange_orders	39
sim_exchange_place_order	40
sim_exchange_positions	41
sim_exchange_process_commands	41
sim_exchange_run	42
sim_exchange_save	42
sim_exchange_settle	43
sim_exchange_set_carry_rates	43
sim_exchange_step	44
sim_exchange_validate_cadence	44
sim_export	45
sim_feed_config	45
sim_feed_configure	46
sim_feed_start	47
sim_feed_status	47
sim_feed_step	48
sim_feed_stop	48
sim_feed_warmup	49
sim_fills	50
sim_heterogeneous_account_step	50
sim_heterogeneous_order_batch_schema	51
sim_import	52
sim_instrument_profile	52
sim_instrument_profiles	53
sim_live_service	53
sim_live_service_run	54
sim_live_state_dashboard_open	54
sim_manifest	55
sim_market_events	55
sim_market_model_calibrate	56
sim_market_model_calibrate_exchange	56
sim_market_model_config	57
sim_market_model_configure	58
sim_market_model_status	59
sim_metrics	59
sim_orders	60
sim_portfolio_decision_policy	60
sim_portfolio_execution	61
sim_portfolio_execution_quality	62
sim_portfolio_export	62

sim_portfolio_market_step	63
sim_portfolio_step	64
sim_portfolio_target_replay	65
sim_portfolio_target_step	67
sim_portfolio_target_submit	68
sim_portfolio_target_submit_batch	69
sim_positions	70
sim_read_account	71
sim_read_events	71
sim_read_manifest	72
sim_read_table	72
sim_replay	73
sim_replay_dashboard_export	73
sim_risk	74
sim_run_from_events	74
sim_schemas	75
sim_schema_migrate	75
sim_schema_version	76
sim_spot_step	76
sim_spot_target_submit	77
sim_state	78
sim_state_dashboard_export	79
sim_step	79
sim_strategy_list	80
sim_strategy_register	81
sim_strategy_unregister	82
sim_strategy_validate_config	82
sim_submit_order	83
sim_trading_calendars	84
TRADESIMR_ACCOUNT_SCHEMA_VERSION	84
TRADESIMR_SCHEMA_VERSION	85
validate_intents	85
validate_market_data	86
vec_batch_run_simulations	86
vec_sim_gen_plot	87
vec_sim_gen_summary	87
vec_sim_gen_summary_table	88
vec_sim_run_backtest	88

as_market_bars	<i>Normalize market bars for tradesimr</i>
----------------	--

Description

Normalize market bars for tradesimr

Usage

```
as_market_bars(
  data,
  timestamp_col = "timestamp",
  symbol_col = NULL,
  asset_id_col = NULL,
  symbol = NULL,
  asset_id = NULL,
  open_col = "open",
  high_col = "high",
  low_col = "low",
  close_col = "close",
  observation_timestamp_col = NULL,
  bar_start_col = NULL,
  bar_end_col = NULL,
  market_timezone = "UTC"
)
```

Arguments

data	A table-like object.
timestamp_col, open_col, high_col, low_col, close_col	Column names.
symbol_col	Optional input symbol column name.
asset_id_col	Optional input asset identifier column name.
symbol	Optional scalar symbol when the input has no symbol column.
asset_id	Optional scalar asset identifier when the input has no asset identifier column.
observation_timestamp_col, bar_start_col, bar_end_col	Optional source columns for explicit market-time metadata. timestamp remains the completed-bar decision boundary for compatibility.
market_timezone	Time zone label when the input has no timezone column.

Value

A data.table with canonical timestamp, open, high, low, close columns.

as_target_positions	<i>Normalize target-position intents for tradesimr</i>
---------------------	--

Description

Normalize target-position intents for tradesimr

Usage

```
as_target_positions(
  data,
  timestamp_col = NULL,
  tgt_pos_col = "tgt_pos",
  pos_strat_col = NULL,
  tol_pos_col = NULL,
  strat = 0L,
  tol_pos = 0
)
```

Arguments

data	A table-like object.
timestamp_col	Optional timestamp column name.
tgt_pos_col	Target-position column name.
pos_strat_col	Optional strategy-id column name.
tol_pos_col	Optional tolerance column name.
strat	Default strategy id.
tol_pos	Default target-position tolerance.

Value

A data.table with canonical intent columns.

PaperTrader	<i>Paper trader demo client</i>
-------------	---------------------------------

Description

Legacy R6 demonstration client for PaperTradingPlatform. The production simulation path is `sim_backtest()` and the C++ execution engine.

Public fields

user_id	User identifier registered on the demo platform.
platform	Reference to a PaperTradingPlatform instance.

Methods

Public methods:

- `PaperTrader$new()`
- `PaperTrader$place_order()`
- `PaperTrader$cancel_order()`
- `PaperTrader$get_all_orders()`
- `PaperTrader$get_order()`
- `PaperTrader$get_wallet()`
- `PaperTrader$get_position()`
- `PaperTrader$clone()`

`PaperTrader$new()`: Create a paper trader demo client.

Usage:

`PaperTrader$new(user_id, platform)`

Arguments:

`user_id` User identifier.

`platform` A PaperTradingPlatform instance.

`PaperTrader$place_order()`: Place a demo order through the platform.

Usage:

`PaperTrader$place_order(inst_id, type, pos, size, price, pricing_method, tag)`

Arguments:

`inst_id` Instrument identifier.

`type` Order type label.

`pos` Position side label.

`size` Order size.

`price` Order price.

`pricing_method` Pricing method label.

`tag` Optional order tag.

Returns: Order id.

`PaperTrader$cancel_order()`: Cancel a demo order.

Usage:

`PaperTrader$cancel_order(order_id)`

Arguments:

`order_id` Order id.

Returns: Platform cancel result.

`PaperTrader$get_all_orders()`: Get all orders for this trader.

Usage:

`PaperTrader$get_all_orders()`

Returns: A data.table of orders.

PaperTrader\$get_order(): Get one order for this trader.

Usage:

PaperTrader\$get_order(order_id)

Arguments:

order_id Order id.

Returns: A data.table with matching order rows.

PaperTrader\$get_wallet(): Get this trader's wallet balance.

Usage:

PaperTrader\$get_wallet()

Returns: Wallet balance or account cash.

PaperTrader\$get_position(): Get this trader's position.

Usage:

PaperTrader\$get_position()

Returns: A data.table of positions.

PaperTrader\$clone(): The objects of this class are cloneable with this method.

Usage:

PaperTrader\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

PaperTradingPlatform *Paper trading platform demo*

Description

Legacy R6 demonstration of a paper trading platform. The production simulation path is `sim_backtest()` and the C++ execution engine.

Public fields

inst_info Instrument metadata used by the demo platform.

bar_info Latest market bar data by instrument.

user_data Demo user wallet and position state.

exchange Backing tradesimr_exchange object.

order_pool Demo order table.

order_id_counter Next demo order id counter.

Methods**Public methods:**

- `PaperTradingPlatform$new()`
- `PaperTradingPlatform$register_user()`
- `PaperTradingPlatform$place_user_order()`
- `PaperTradingPlatform$cancel_user_order()`
- `PaperTradingPlatform$get_user_orders()`
- `PaperTradingPlatform$get_user_order()`
- `PaperTradingPlatform$get_live_orders()`
- `PaperTradingPlatform$get_order()`
- `PaperTradingPlatform$update_bar()`
- `PaperTradingPlatform$process_order()`
- `PaperTradingPlatform$fill_order()`
- `PaperTradingPlatform$get_user_wallet()`
- `PaperTradingPlatform$get_user_position()`
- `PaperTradingPlatform$update_user_wallet()`
- `PaperTradingPlatform$update_user_position()`
- `PaperTradingPlatform$clone()`

`PaperTradingPlatform$new()`: Create a paper trading platform demo.

Usage:

```
PaperTradingPlatform$new(config = list())
```

Arguments:

`config` Simulation config passed to `sim_exchange_new()`.

`PaperTradingPlatform$register_user()`: Register a demo user and return a trader client.

Usage:

```
PaperTradingPlatform$register_user(user_id, initial_balance = 10000)
```

Arguments:

`user_id` User identifier.

`initial_balance` Initial wallet balance.

Returns: A `PaperTrader` instance.

`PaperTradingPlatform$place_user_order()`: Place a demo user order.

Usage:

```
PaperTradingPlatform$place_user_order(
  user_id,
  inst_id,
  type,
  pos,
  size,
  price,
  pricing_method,
  tag
)
```

Arguments:

`user_id` User identifier.
`inst_id` Instrument identifier.
`type` Order type label.
`pos` Position side label.
`size` Order size.
`price` Order price.
`pricing_method` Pricing method label.
`tag` Optional order tag.

Returns: Order id.

`PaperTradingPlatform$cancel_user_order()`: Cancel a demo user order.

Usage:

`PaperTradingPlatform$cancel_user_order(user_id, order_id)`

Arguments:

`user_id` User identifier.
`order_id` Order id.

`PaperTradingPlatform$get_user_orders()`: Get all orders for a user.

Usage:

`PaperTradingPlatform$get_user_orders(user_id)`

Arguments:

`user_id` User identifier.

Returns: A data.table of orders.

`PaperTradingPlatform$get_user_order()`: Get a user order.

Usage:

`PaperTradingPlatform$get_user_order(user_id, order_id)`

Arguments:

`user_id` User identifier.
`order_id` Order id.

Returns: A data.table with matching order rows.

`PaperTradingPlatform$get_live_orders()`: Get live demo orders.

Usage:

`PaperTradingPlatform$get_live_orders()`

Returns: A data.table of live orders.

`PaperTradingPlatform$get_order()`: Get an order by id.

Usage:

`PaperTradingPlatform$get_order(order_id)`

Arguments:

order_id Order id.

Returns: A data.table with matching order rows.

PaperTradingPlatform\$update_bar(): Update the latest market bar and append it to the exchange.

Usage:

PaperTradingPlatform\$update_bar(inst_id, timestamp, open, high, low, close)

Arguments:

inst_id Instrument identifier.

timestamp Bar timestamp.

open, high, low, close OHLC prices.

PaperTradingPlatform\$process_order(): Process one demo order against the latest bar.

Usage:

PaperTradingPlatform\$process_order(order)

Arguments:

order Order row.

PaperTradingPlatform\$fill_order(): Fill a demo order and pass target exposure to the exchange.

Usage:

PaperTradingPlatform\$fill_order(order_id, fill_price)

Arguments:

order_id Order id.

fill_price Fill price.

PaperTradingPlatform\$get_user_wallet(): Get a user's wallet balance.

Usage:

PaperTradingPlatform\$get_user_wallet(user_id)

Arguments:

user_id User identifier.

Returns: Wallet balance or account cash.

PaperTradingPlatform\$get_user_position(): Get a user's position.

Usage:

PaperTradingPlatform\$get_user_position(user_id)

Arguments:

user_id User identifier.

Returns: A data.table of positions.

PaperTradingPlatform\$update_user_wallet(): Update a user's demo wallet balance.

Usage:

PaperTradingPlatform\$update_user_wallet(user_id, delta_wallet_balance)

Arguments:

user_id User identifier.

delta_wallet_balance Wallet balance delta.

PaperTradingPlatform\$update_user_position(): Update a user's demo position after a fill.

Usage:

PaperTradingPlatform\$update_user_position(order)

Arguments:

order Filled order row.

Returns: Wallet balance delta.

PaperTradingPlatform\$clone(): The objects of this class are cloneable with this method.

Usage:

PaperTradingPlatform\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

sim_account

Extract account snapshots from a simulation

Description

Extract account snapshots from a simulation

Usage

```
sim_account(sim)
```

Arguments

sim A simulation result returned by `sim_backtest()`.

Value

A data.table of bar-level account snapshots.

sim_agents_step	<i>Step active AI agents and append their order commands</i>
-----------------	--

Description

Step active AI agents and append their order commands

Usage

```
sim_agents_step(exchange, bar = NULL)
```

Arguments

exchange	A tradesimr_exchange.
bar	Optional current bar used for decision timestamps and prices.

Value

A data.table of generated decisions.

sim_agent_add	<i>Add an AI or human agent to a live exchange</i>
---------------	--

Description

AI agents generate ordinary order requests; they do not bypass the exchange command/execution path.

Usage

```
sim_agent_add(
  exchange,
  agent_id = NULL,
  agent_type = c("chaos", "momentum", "contrarian", "mean_reversion", "strategy",
    "human"),
  config = list(),
  status = c("active", "paused")
)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Agent identifier. If omitted, one is generated.
agent_type	Agent type: human, chaos, momentum, contrarian, mean_reversion, or strategy.
config	Named list of agent settings. Supported values include qty, order_type, lookback, asset_policy, and for strategy agents strategy_id or strategy_fun. Strategy parameters can be supplied with param_ prefixes, for example param_fast = 10.
status	Initial status: active or paused.

Value

The agent id.

sim_agent_command_schema
<i>Get live-agent command schemas</i>

Description

Get live-agent command schemas

Usage

sim_agent_command_schema()

Value

A named list of empty data.tables for append-only agent commands, order requests, and order cancellations.

sim_agent_dashboard_export
<i>Export an agent-facing live dashboard</i>

Description

Export an agent-facing live dashboard

Usage

sim_agent_dashboard_export(exchange, path)

Arguments

exchange	A tradesimr_exchange.
path	Output directory.

Value

Invisibly returns a named character vector of written files.

sim_agent_dashboard_open	<i>Open an agent-facing live dashboard</i>
--------------------------	--

Description

Open an agent-facing live dashboard

Usage

```
sim_agent_dashboard_open(  
  exchange = sim_exchange_new(),  
  path = tempfile("tradesimr-live-agent-")  
)
```

Arguments

exchange	A tradesimr_exchange.
path	Output directory.

Value

Invisibly returns the dashboard index path.

sim_agent_rankings	<i>Compute current agent rankings</i>
--------------------	---------------------------------------

Description

Rankings combine current per-agent account equity with order activity.

Usage

```
sim_agent_rankings(exchange)
```

Arguments

exchange	A tradesimr_exchange.
----------	-----------------------

Value

A data.table of agent rankings.

sim_agent_remove	<i>Remove an agent from a live exchange</i>
------------------	---

Description

Removed agents stay in the durable registry with status removed.

Usage

```
sim_agent_remove(exchange, agent_id)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Agent identifier.

Value

Invisibly returns TRUE when an agent was updated.

sim_agent_set_status	<i>Set an agent status</i>
----------------------	----------------------------

Description

Set an agent status

Usage

```
sim_agent_set_status(
  exchange,
  agent_id,
  status = c("active", "paused", "removed")
)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Agent identifier.
status	New status: active, paused, or removed.

Value

Invisibly returns TRUE when an agent was updated.

sim_assets	<i>List registered exchange assets</i>
------------	--

Description

List registered exchange assets

Usage

```
sim_assets(exchange)
```

Arguments

exchange A tradesimr_exchange.

Value

A data.table of assets.

sim_asset_add	<i>Register a tradable asset on an exchange</i>
---------------	---

Description

Register a tradable asset on an exchange

Usage

```
sim_asset_add(  
  exchange,  
  symbol,  
  asset_id = NULL,  
  status = c("active", "paused", "delisted", "expired", "removed"),  
  asset_class = "other",  
  instrument_profile = NULL,  
  contract_size = 1,  
  tick_size = NA_real_,  
  qty_step = 1,  
  base_ccy = NA_character_,  
  quote_ccy = NA_character_,  
  calendar_id = NULL,  
  timezone = NULL,  
  settlement_lag_days = NULL,  
  margin_model = NULL,  
  bar_cadence_seconds = NA_real_,  
  metadata = list()  
)
```

Arguments

exchange	A tradesimr_exchange.
symbol	Asset symbol, for example "BTC-USDT-SWAP".
asset_id	Optional integer asset id. Defaults to a stable id derived from symbol.
status	Asset status: active, paused, delisted, expired, or removed.
asset_class	Asset class label, such as crypto_perp, stock, bond, etf, commodity_future, fx, or other.
instrument_profile	Canonical accounting/calendar profile. Defaults to the profile implied by asset_class.
contract_size	Contract multiplier used by execution/accounting.
tick_size	Minimum price increment.
qty_step	Minimum order quantity increment.
base_ccy, quote_ccy	Optional currency labels.
calendar_id, timezone	Optional market-calendar metadata. Defaults are supplied by the selected instrument profile.
settlement_lag_days	Optional settlement lag override.
margin_model	Optional margin-model override.
bar_cadence_seconds	Optional expected completed-bar cadence in seconds.
metadata	Optional named list of durable profile metadata.

Value

Invisibly returns the registered asset row.

sim_asset_remove	<i>Remove an asset from an exchange registry</i>
------------------	--

Description

Removed assets remain in the durable registry with status removed.

Usage

```
sim_asset_remove(exchange, symbol = NULL, asset_id = NULL)
```

Arguments

exchange	A tradesimr_exchange.
symbol	Optional symbol.
asset_id	Optional integer asset id.

Value

Invisibly returns TRUE when an asset was updated.

sim_backtest

Run a stateful trading simulation backtest

Description

sim_backtest() executes target-position intentions through the package's C++ exchange/accounting engine. Orders are planned at bar close and market orders are filled on the next bar open. Target-derived opening and increase orders are fee-aware: they are clipped at the executable price to the largest step-rounded quantity satisfying $\text{equity} - \text{fee} \geq \text{initial_margin}$. Thus a tgt_pos of 1 at lev = 1 produces the largest near-100%-notional position that reserves its transaction fee rather than a failed order.

Usage

```
sim_backtest(
  data,
  timestamp_col = "timestamp",
  open_col = "open",
  high_col = "high",
  low_col = "low",
  close_col = "close",
  tgt_pos_col = "tgt_pos",
  pos_strat_col = NULL,
  tol_pos_col = NULL,
  order_type_col = NULL,
  limit_price_col = NULL,
  strat = 0L,
  asset = 0L,
  init_cash = 10000,
  ctr_size = 1,
  ctr_step = 1,
  lev = 10,
  fee_rt = 0,
  maker_fee_rt = NA_real_,
  taker_fee_rt = NA_real_,
  fund_rt = 0,
  funding_interval_hours = 8,
  mmr = 0.02,
  fill_model = c("next_open", "same_close"),
  slippage = 0,
  spread = 0,
  tol_pos = 0,
  record = TRUE
)
```

Arguments

<code>data</code>	A data frame/data.table with timestamp, open, high, low, close, and target-position columns.
<code>timestamp_col, open_col, high_col, low_col, close_col, tgt_pos_col</code>	Column names in data.
<code>pos_strat_col</code>	Optional strategy-id column. If absent, <code>strat</code> is used.
<code>tol_pos_col</code>	Optional target-position tolerance column. If absent, <code>tol_pos</code> is used.
<code>order_type_col</code>	Optional order type column using market or limit.
<code>limit_price_col</code>	Optional limit price column.
<code>strat, asset</code>	Integer identifiers for the simulation and asset.
<code>init_cash</code>	Initial account cash.
<code>ctr_size</code>	Contract size.
<code>ctr_step</code>	Minimum contract increment.
<code>lev</code>	Leverage used for initial margin.
<code>fee_rt</code>	Trading fee rate on notional. Fees are reserved by target-derived opening/increase actions at their fill boundary; explicit contract orders continue to fail if their requested quantity is infeasible.
<code>maker_fee_rt, taker_fee_rt</code>	Optional maker/taker fee rates. Missing values fall back to <code>fee_rt</code> .
<code>fund_rt</code>	Funding rate per 8 hours on notional.
<code>funding_interval_hours</code>	Funding interval in hours.
<code>mmr</code>	Maintenance margin rate.
<code>fill_model</code>	Fill timing model: <code>next_open</code> or <code>same_close</code> .
<code>slippage</code>	Absolute slippage added against trade direction.
<code>spread</code>	Absolute bid/ask spread; half spread is added against trade direction.
<code>tol_pos</code>	Scalar default target-position tolerance used when <code>tol_pos_col</code> is absent.
<code>record</code>	Whether to attach the execution recorder.

Value

A data.table with timestamp and equity. If `record = TRUE`, an execution recorder is attached as attribute `orders`.

sim_bond_schedules	<i>List durable bond schedules</i>
--------------------	------------------------------------

Description

List durable bond schedules

Usage

```
sim_bond_schedules(exchange)
```

Arguments

exchange	A tradesimr_exchange.
----------	-----------------------

Value

A data.table of bond schedule state.

sim_bond_schedule_add	<i>Register a calendar-driven bond schedule</i>
-----------------------	---

Description

The schedule uses a fixed ACT/day-count convention and equal coupon periods. At each eligible heterogeneous account boundary, C++ emits non-cash accrual events, books due coupons into settled cash, and redeems remaining inventory at maturity. A sparse replay boundary crossing multiple coupon dates is processed coupon-by-coupon before any remaining partial-period accrual. The cursor fields are durable exchange state, so resumed replay continues from the same coupon boundary.

Usage

```
sim_bond_schedule_add(  
  exchange,  
  symbol,  
  coupon_rate,  
  coupon_frequency = 2L,  
  issue_timestamp,  
  maturity_timestamp,  
  face_value = 100,  
  accrual_day_count = 365,  
  currency = NULL  
)
```

Arguments

exchange	A tradesimr_exchange.
symbol	Registered bond symbol.
coupon_rate	Annual decimal coupon rate.
coupon_frequency	Number of equal coupon payments per 365-day year.
issue_timestamp	Schedule start timestamp.
maturity_timestamp	Maturity/redemption timestamp after issue_timestamp.
face_value	Redemption value per inventory unit.
accrual_day_count	Positive ACT denominator used for accrual events.
currency	Coupon and redemption currency. Defaults to the asset quote currency.

Value

Invisibly returns the registered schedule row.

sim_calendar_expected_bars

Generate expected completed bar timestamps from a calendar

Description

Generate expected completed bar timestamps from a calendar

Usage

```
sim_calendar_expected_bars(
    calendar_id,
    start,
    end,
    cadence_seconds,
    exceptions = NULL
)
```

Arguments

calendar_id	Built-in calendar identifier.
start, end	Timestamp bounds.
cadence_seconds	Positive bar cadence in seconds.
exceptions	Optional exception table with session_date, action, and close_time fields.

Value

A data.table of expected completed-bar timestamps.

sim_calendar_holidays *List deterministic built-in calendar holidays*

Description

List deterministic built-in calendar holidays

Usage

```
sim_calendar_holidays(calendar_id, start, end)
```

Arguments

calendar_id	Built-in calendar identifier.
start, end	Date/POSIXct bounds.

Value

A data.table of session dates, labels, and optional early close time.

sim_calendar_is_open *Test whether timestamps fall in a built-in tradable session*

Description

This deliberately provides deterministic session rules, not a vendor holiday database. XNYS includes fixed-date observed holidays; callers may mark exceptional closures with `is_tradable = FALSE` in market bars.

Usage

```
sim_calendar_is_open(timestamp, calendar_id = "ALWAYS_OPEN", exceptions = NULL)
```

Arguments

timestamp	POSIXct timestamps.
calendar_id	Built-in calendar identifier.
exceptions	Optional calendar-exception rows from sim_exchange_calendar_exception() .

Value

A logical vector.

sim_calendar_settlement_timestamp	<i>Calculate a calendar-aware settlement timestamp</i>
-----------------------------------	--

Description

Settlement lags count tradable calendar dates, not raw 24-hour periods. This gives FX spot its conventional weekday progression while preserving same-day settlement for 24/7 instruments. Exchange-specific closed-date exceptions are respected.

Usage

```
sim_calendar_settlement_timestamp(
    calendar_id,
    timestamp,
    settlement_lag_days = 0L,
    exceptions = NULL
)
```

Arguments

calendar_id	Built-in settlement calendar identifier.
timestamp	Trade timestamp.
settlement_lag_days	Non-negative whole settlement days.
exceptions	Optional calendar-exception rows.

Value

A UTC POSIXct settlement timestamp.

sim_calendar_spec	<i>Get a trading-calendar specification</i>
-------------------	---

Description

Get a trading-calendar specification

Usage

```
sim_calendar_spec(calendar_id)
```

Arguments

calendar_id	Built-in calendar identifier.
-------------	-------------------------------

Value

A one-row data.table with timezone, session rule, and holiday policy.

sim_cancel_order	<i>Submit an agent order cancellation command</i>
------------------	---

Description

Submit an agent order cancellation command

Usage

```
sim_cancel_order(  
  exchange,  
  agent_id,  
  order_id,  
  client_order_id = NA_character_,  
  timestamp = Sys.time(),  
  process = TRUE  
)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Agent identifier.
order_id	Order id to cancel.
client_order_id	Optional client order id for audit purposes.
timestamp	Command timestamp.
process	Whether to process pending commands immediately.

Value

The generated command id.

sim_cash_ledger	<i>Extract cash ledger entries from a simulation</i>
-----------------	--

Description

Extract cash ledger entries from a simulation

Usage

```
sim_cash_ledger(sim)
```

Arguments

sim	A simulation result returned by <code>sim_backtest()</code> .
-----	---

Value

A data.table of event-level cash changes.

sim_cross_asset_risk	<i>Compute cross-asset risk for live exchange agents</i>
----------------------	--

Description

Compute cross-asset risk for live exchange agents

Usage

```
sim_cross_asset_risk(exchange, stress_sigma = 2)
```

Arguments

exchange	A tradesimr_exchange.
stress_sigma	Multiplier applied to portfolio return volatility for the stress-loss estimate.

Value

A data.table with one row per agent and exposed asset.

sim_dashboard_export	<i>Export a static simulation dashboard</i>
----------------------	---

Description

Compatibility alias for `sim_replay_dashboard_export()`.

Usage

```
sim_dashboard_export(sim, path)
```

Arguments

sim	A simulation result returned by <code>sim_backtest()</code> or <code>sim_exchange_step()</code> .
path	Output directory.

Value

Invisibly returns a named character vector of written files.

sim_dashboard_open	<i>Open an exported static dashboard</i>
--------------------	--

Description

Open an exported static dashboard

Usage

```
sim_dashboard_open(path)
```

Arguments

path	Directory created by a dashboard export helper.
------	---

Value

Invisibly returns the dashboard index path.

sim_events	<i>Convert a simulation recorder into an event table</i>
------------	--

Description

Convert a simulation recorder into an event table

Usage

sim_events(x)

Arguments

x A simulation result returned by sim_backtest() or a raw recorder list from the C++ engine.

Value

A data.table of recorded simulation events.

sim_exchange_account	<i>Get simulated exchange account state</i>
----------------------	---

Description

Get simulated exchange account state

Usage

sim_exchange_account(exchange)

Arguments

exchange A tradesimr_exchange.

Value

A one-row data.table with the latest account snapshot.

`sim_exchange_account_state`*Get the durable heterogeneous account state*

Description

Returns the typed, profile-aware account projection used by the heterogeneous execution engine. Cash is valued in the exchange base currency; fully paid inventory contributes marked market value, while margin positions contribute marked unrealized P&L and maintenance margin. `sim_exchange_account()` remains the compatibility account snapshot API.

Usage

```
sim_exchange_account_state(exchange, agent_id = NULL)
```

Arguments

<code>exchange</code>	A <code>tradesimr_exchange</code> .
<code>agent_id</code>	Optional account identifier.

Value

A named list containing `account`, `cash_balances`, `inventory_positions`, `margin_positions`, and `events` `data.tables`.

`sim_exchange_accrue_carry`*Accrue profile-aware borrow and cash interest*

Description

The function is idempotent at a timestamp and is called automatically before every executable exchange boundary. Call it explicitly to establish an initial accrual cursor or to accrue a durable account without new bars.

Usage

```
sim_exchange_accrue_carry(exchange, timestamp)
```

Arguments

<code>exchange</code>	A <code>tradesimr_exchange</code> .
<code>timestamp</code>	Accrual boundary.

Value

A `data.table` of booked carry events.

sim_exchange_addBars *Append market bars to a simulated exchange*

Description

Append market bars to a simulated exchange

Usage

```
sim_exchange_addBars(exchange, bars)
```

Arguments

exchange	A tradesimr_exchange.
bars	Market bars coercible by as_market_bars().

Value

The exchange, invisibly.

sim_exchange_calendarizeBars
Calendarize registered-asset market bars

Description

Calendarize registered-asset market bars

Usage

```
sim_exchange_calendarizeBars(exchange, bars, strict = FALSE)
```

Arguments

exchange	A tradesimr_exchange.
bars	Market bars.
strict	Whether a supplied tradable bar outside its registered session should error rather than be converted to valuation-only.

Value

Canonical market bars with calendar-derived is_tradable values.

sim_exchange_calendar_exception
<i>Add an exchange-specific calendar exception</i>

Description

Add an exchange-specific calendar exception

Usage

```
sim_exchange_calendar_exception(  
  exchange,  
  session_date,  
  action = c("closed", "early_close"),  
  calendar_id = NULL,  
  symbol = NULL,  
  asset_id = NULL,  
  close_time = NULL,  
  message = ""  
)
```

Arguments

exchange	A tradesimr_exchange.
session_date	Local session date.
action	"closed" or "early_close".
calendar_id	Optional calendar identifier.
symbol, asset_id	Optional registered-asset scope.
close_time	Required "HH:MM" for an early close.
message	Public-safe description.

Value

Invisibly returns the durable exception row.

`sim_exchange_cancel_order`*Cancel an intent-level order in a simulated exchange*

Description

Cancel an intent-level order in a simulated exchange

Usage

```
sim_exchange_cancel_order(exchange, order_id)
```

Arguments

<code>exchange</code>	A <code>tradesimr_exchange</code> .
<code>order_id</code>	Order id returned by <code>sim_exchange_place_order()</code> .

Value

Invisibly returns `TRUE` if an order was cancelled.

`sim_exchange_cash_adjust`*Deposit or withdraw a profile-aware currency balance*

Description

Deposit or withdraw a profile-aware currency balance

Usage

```
sim_exchange_cash_adjust(  
  exchange,  
  agent_id,  
  amount,  
  currency = NULL,  
  timestamp = Sys.time(),  
  message = "Manual cash adjustment"  
)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Account identifier.
amount	Signed amount.
currency	Currency code. Defaults to the exchange base currency.
timestamp	Ledger timestamp.
message	Public-safe ledger description.

Value

Invisibly returns the resulting currency balance.

sim_exchange_cash_balances
Get profile-aware cash balances

Description

Get profile-aware cash balances

Usage

```
sim_exchange_cash_balances(exchange, agent_id = NULL)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Optional account identifier.

Value

A data.table where amount is settled cash (kept for compatibility), unsettled is pending settlement cash, and the base-value columns report settled-only and total cash valuation respectively.

sim_exchange_convert_cash

Convert cash between currencies at an authoritative exchange FX mark

Description

Convert cash between currencies at an authoritative exchange FX mark

Usage

```
sim_exchange_convert_cash(
    exchange,
    agent_id,
    amount,
    from_ccy,
    to_ccy,
    timestamp = Sys.time()
)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Account identifier.
amount	Source-currency amount to convert.
from_ccy, to_ccy	Currency codes.
timestamp	Ledger timestamp.

Value

Invisibly returns the destination amount.

sim_exchange_corporate_action

Register a durable inventory corporate action

Description

Actions are applied at the first exchange step at or after their effective timestamp and retained in a durable audit table.

Usage

```
sim_exchange_corporate_action(
  exchange,
  symbol,
  action_type = c("dividend", "split", "coupon", "bond_accrual", "redemption",
    "delisting", "future_expiry", "future_roll"),
  amount,
  effective_timestamp,
  currency = NULL
)
```

Arguments

exchange	A tradesimr_exchange.
symbol	Registered symbol.
action_type	One of dividend, split, coupon, bond_accrual, redemption, delisting, future_expiry, or future_roll.
amount	Cash per inventory unit for dividend/coupon/accrual/redemption, split ratio for split, or the per-unit cash settlement price for delisting.
effective_timestamp	Action timestamp.
currency	Action currency. Defaults to the asset quote currency.

Value

Invisibly returns the action id.

sim_exchange_dashboard

Export and open a simulated exchange dashboard

Description

Export and open a simulated exchange dashboard

Usage

```
sim_exchange_dashboard(exchange, path)
```

Arguments

exchange	A tradesimr_exchange with a simulation result.
path	Output directory.

Value

Invisibly returns written dashboard files.

`sim_exchange_export_events`*Export exchange simulation events*

Description

Export exchange simulation events

Usage

```
sim_exchange_export_events(exchange, path, format = c("csv", "fst"))
```

Arguments

<code>exchange</code>	A <code>tradesimr_exchange</code> .
<code>path</code>	Output directory.
<code>format</code>	File format, either <code>csv</code> or <code>fst</code> .

Value

Invisibly returns written file paths.

`sim_exchange_future_roll`*Register a futures expiry or contract roll*

Description

Expiry settles the old contract's marked P&L into settled quote-currency cash and terminates its open margin position. A roll additionally opens the same signed quantity in a registered successor contract at `roll_price`. The successor must not already have an open margin position for an affected account; callers should make any independent successor adjustment first.

Usage

```
sim_exchange_future_roll(  
  exchange,  
  symbol,  
  effective_timestamp,  
  settlement_price,  
  successor_symbol = NULL,  
  roll_price = NULL  
)
```

Arguments

exchange	A tradesimr_exchange.
symbol	Expiring registered futures symbol.
effective_timestamp	Lifecycle boundary.
settlement_price	Cash-settlement price for the expiring contract.
successor_symbol	Optional registered successor futures symbol.
roll_price	Required successor reference price when rolling.

Value

Invisibly returns the corporate action id.

sim_exchange_fx_rate *Set a foreign-exchange conversion rate*

Description

Rates express units of to_ccy per one unit of from_ccy. They are used only for account valuation and explicit currency conversion; they never alter an execution price.

Usage

```
sim_exchange_fx_rate(
  exchange,
  from_ccy,
  to_ccy,
  rate,
  timestamp = Sys.time(),
  source = "manual"
)
```

Arguments

exchange	A tradesimr_exchange.
from_ccy	Source currency.
to_ccy	Destination currency.
rate	Positive conversion rate.
timestamp	Valuation timestamp.
source	Public-safe source label.

Value

Invisibly returns the added rate row.

sim_exchange_load	<i>Load exchange state from disk</i>
-------------------	--------------------------------------

Description

Load exchange state from disk

Usage

```
sim_exchange_load(path)
```

Arguments

path	Directory produced by sim_exchange_save().
------	--

Value

A tradesimr_exchange.

sim_exchange_new	<i>Create an in-memory simulated exchange state</i>
------------------	---

Description

Create an in-memory simulated exchange state

Usage

```
sim_exchange_new(config = list())
```

Arguments

config	Named simulation parameters. Set execution_engine to "heterogeneous_v2" to route portfolio boundaries through the typed multi-profile C++ account kernel; it is the default. "legacy_v1" is a deprecated compatibility route for downstream consumers that still require the legacy snapshot/event projection. calendar_mode is "raw" (default), "calendarize" (closed or cadence-misaligned bars become valuation-only), or "strict" (such bars are rejected).
--------	---

Value

A mutable environment containing market, intent, order, and result tables.

`sim_exchange_new_events`*Get new events since the previous exchange run*

Description

Get new events since the previous exchange run

Usage

```
sim_exchange_new_events(exchange)
```

Arguments

`exchange` A `tradesimr_exchange`.

Value

A `data.table` of newly observed simulation events.

`sim_exchange_orders` *Get simulated exchange orders*

Description

Get simulated exchange orders

Usage

```
sim_exchange_orders(exchange)
```

Arguments

`exchange` A `tradesimr_exchange`.

Value

A `data.table` of accepted/cancelled intent-level orders.

sim_exchange_place_order

Place an order into a simulated exchange

Description

Explicit orders use qty_type = "contracts" by default for buy/sell/flat orders. Use qty_type = "target_pos" or side = "target" for exposure targets consumed by replay-style backtests.

Usage

```
sim_exchange_place_order(
  exchange,
  agent_id,
  timestamp,
  symbol = NULL,
  asset_id = NULL,
  tgt_pos = NULL,
  tol_pos = 0,
  order_type = c("market", "limit"),
  side = c("target", "buy", "sell", "flat"),
  qty_type = NULL,
  qty = NULL,
  limit_price = NA_real_,
  time_in_force = "gtc",
  atomic_group_id = NULL,
  client_order_id = NA_character_
)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Agent identifier.
timestamp	Order timestamp.
symbol	Registered asset symbol.
asset_id	Registered asset identifier. Provide symbol, asset_id, or both when they identify the same asset.
tgt_pos	Target exposure. Kept for compatibility with earlier intent-level calls.
tol_pos	Target-position tolerance.
order_type	Order type: market or limit.
side	Order side: target, buy, sell, or flat.
qty_type	Quantity semantics: contracts or target_pos.
qty	Order quantity. Meaning is controlled by qty_type.
limit_price	Optional limit price for limit orders.

time_in_force Time-in-force label.
atomic_group_id Optional atomic execution group. Explicit orders in the same group either commit together or are rejected together.
client_order_id Optional client order id.

Value

The generated order id.

sim_exchange_positions
<i>Get simulated exchange positions</i>

Description

Get simulated exchange positions

Usage

sim_exchange_positions(exchange)

Arguments

exchange A tradesimr_exchange.

Value

A one-row data.table with the latest position snapshot.

sim_exchange_process_commands
<i>Process pending agent commands</i>

Description

Converts pending append-only order request and cancellation commands into exchange orders and cancellation attempts.

Usage

sim_exchange_process_commands(exchange)

Arguments

exchange A tradesimr_exchange.

Value

A data.table of processed command rows.

sim_exchange_run	<i>Run or refresh a simulated exchange replay</i>
------------------	---

Description

Run or refresh a simulated exchange replay

Usage

```
sim_exchange_run(exchange)
```

Arguments

exchange A tradesimr_exchange.

Value

A simulation result returned by sim_backtest().

sim_exchange_save	<i>Export exchange events and state</i>
-------------------	---

Description

Export exchange events and state

Usage

```
sim_exchange_save(exchange, path, format = c("csv", "fst"))
```

Arguments

exchange A tradesimr_exchange.
path Output directory.
format File format, either csv or fst.

Value

Invisibly returns written file paths.

sim_exchange_settle	<i>Settle due profile-aware cash movements</i>
---------------------	--

Description

Settle due profile-aware cash movements

Usage

```
sim_exchange_settle(exchange, timestamp = Sys.time())
```

Arguments

exchange	A tradesimr_exchange.
timestamp	Settlement cutoff.

Value

A data.table of settled ledger rows.

sim_exchange_set_carry_rates	<i>Configure borrow and cash interest rates</i>
------------------------------	---

Description

Rates are annualized simple rates keyed by registered symbol for borrow and by currency for settled-cash interest. Positive cash earns the configured rate; negative cash is charged it. Short inventory is charged its symbol's borrow rate from the inventory quote-currency balance.

Usage

```
sim_exchange_set_carry_rates(  
  exchange,  
  borrow_rates = numeric(),  
  cash_interest_rates = numeric()  
)
```

Arguments

exchange	A tradesimr_exchange.
borrow_rates	Named numeric annualized rates keyed by symbol.
cash_interest_rates	Named numeric annualized rates keyed by currency.

Value

Invisibly returns the configured rates.

sim_exchange_step	<i>Step a simulated exchange with one or more bars</i>
-------------------	--

Description

Step a simulated exchange with one or more bars

Usage

```
sim_exchange_step(exchange, bars)
```

Arguments

exchange	A tradesimr_exchange.
bars	Market bars coercible by as_market_bars().

Value

The incremental simulation snapshots.

sim_exchange_validate_cadence	<i>Validate registered-asset bar cadence</i>
-------------------------------	--

Description

Validate registered-asset bar cadence

Usage

```
sim_exchange_validate_cadence(exchange, bars, strict = FALSE)
```

Arguments

exchange	A tradesimr_exchange.
bars	Market bars.
strict	Whether cadence violations should error.

Value

A data.table with calendar_open and cadence_ok columns.

sim_export	<i>Export simulation tables to durable files</i>
------------	--

Description

Export simulation tables to durable files

Usage

```
sim_export(  
  sim,  
  path,  
  format = c("csv", "fst"),  
  tables = c("simulation", "market_events", "events", "orders", "fills", "positions",  
            "cash_ledger", "account", "risk")  
)
```

Arguments

sim	A simulation result returned by <code>sim_backtest()</code> .
path	Output directory.
format	File format, either csv or fst.
tables	Names of tables to export.

Value

Invisibly returns a named character vector of written file paths.

sim_feed_config	<i>Default live feed configuration</i>
-----------------	--

Description

Default live feed configuration

Usage

```
sim_feed_config(  
  symbol = "BTC-USDT-SWAP",  
  asset_id = NULL,  
  timeframe = "4h",  
  tz = "UTC",  
  feed_mode = c("simulation", "external"),  
  feed_adapter = NULL,  
  start_time = NULL,
```

```
simulation_model = c("random_walk", "ar", "garch11", "ar_garch", "regime"),
random_walk = list(start_price = 100, drift = 0, vol = 0.02, seed = 1L),
simulation = list()
)
```

Arguments

symbol	Instrument symbol.
asset_id	Optional integer asset id.
timeframe	Bar interval, such as "4h", "1h", or "15m".
tz	Time zone used to align completed bar boundaries.
feed_mode	Feed mode: simulation or external.
feed_adapter	Optional external adapter function with signature function(symbol, timeframe, start, end, tz = "UTC", feed_mode = "simulation", feed_adapter = NULL)
start_time	Optional first completed boundary to process.
simulation_model	Simulation model: random_walk, ar, garch11, ar_garch, or regime.
random_walk	List of random-walk simulation settings: start_price, drift, vol, and seed.
simulation	Advanced simulation settings. Supported nested lists are ar, garch11, and ohlc.

Value

A list suitable for sim_feed_configure().

sim_feed_configure	<i>Configure a live exchange feed</i>
--------------------	---------------------------------------

Description

Configure a live exchange feed

Usage

```
sim_feed_configure(exchange, config = sim_feed_config())
```

Arguments

exchange	A tradesimr_exchange.
config	Feed configuration list. A list with configs may be used to configure all registered assets in one call. A market_model element may be used to configure synchronized multi-asset simulation.

Value

The feed configuration, invisibly.

sim_feed_start	<i>Start a configured live feed</i>
----------------	-------------------------------------

Description

Start a configured live feed

Usage

```
sim_feed_start(exchange, now = Sys.time(), symbol = NULL, asset_id = NULL)
```

Arguments

exchange	A tradesimr_exchange.
now	Current time used to initialize the schedule.
symbol, asset_id	Optional feed asset selector. If omitted, all configured active asset feeds are started.

Value

Feed status.

sim_feed_status	<i>Get live feed status</i>
-----------------	-----------------------------

Description

Get live feed status

Usage

```
sim_feed_status(exchange)
```

Arguments

exchange	A tradesimr_exchange.
----------	-----------------------

Value

A list describing feed state.

sim_feed_step	<i>Step a live feed through completed bars</i>
---------------	--

Description

Generates or fetches all completed bars after the last processed feed boundary and appends them through `sim_exchange_step()`.

Usage

```
sim_feed_step(
  exchange,
  now = Sys.time(),
  maxBars = Inf,
  symbol = NULL,
  asset_id = NULL
)
```

Arguments

exchange	A <code>tradesimr_exchange</code> .
now	Current time.
maxBars	Maximum bars to append in one call.
symbol, asset_id	Optional feed asset selector. If omitted, all configured active asset feeds are stepped.

Value

A `data.table` of bars appended by this call.

sim_feed_stop	<i>Stop a configured live feed</i>
---------------	------------------------------------

Description

Stop a configured live feed

Usage

```
sim_feed_stop(exchange, symbol = NULL, asset_id = NULL)
```

Arguments

exchange A tradesimr_exchange.
 symbol, asset_id Optional feed asset selector. If omitted, all configured active asset feeds are stopped.

Value

Feed status.

sim_feed_warmup	<i>Generate historical simulation bars before starting a live feed</i>
-----------------	--

Description

Appends n_bars simulated OHLC bars ending at the latest completed boundary. This is a market-history warmup: it does not step AI agents or process pending orders.

Usage

```
sim_feed_warmup(
  exchange,
  n_bars = 100L,
  now = Sys.time(),
  symbol = NULL,
  asset_id = NULL
)
```

Arguments

exchange A tradesimr_exchange.
 n_bars Number of historical bars to append.
 now Current time used to align the latest completed boundary.
 symbol, asset_id Optional feed asset selector. If omitted, all configured active asset feeds are warmed up.

Value

A data.table of appended market bars.

sim_fills

Extract fill events from a simulation

Description

Extract fill events from a simulation

Usage

```
sim_fills(sim)
```

Arguments

sim A simulation result returned by `sim_backtest()`.

Value

A data.table of filled trade events.

sim_heterogeneous_account_step

Step a heterogeneous profile-aware account kernel

Description

Marks inventory, settles futures/perpetual variation margin, and evaluates a normalized order batch without mutating the supplied R input tables.

Usage

```
sim_heterogeneous_account_step(
  base_currency,
  cash_balances,
  inventory_positions = data.frame(),
  margin_positions,
  bars,
  fx_rates,
  settlements = data.frame(),
  corporate_actions = data.frame(),
  orders = data.frame(),
  timestamp = Sys.time()
)
```

Arguments

base_currency	Account reporting currency.
cash_balances	Data frame with currency, settled, and unsettled.
inventory_positions	Data frame with inventory units and valuation.
margin_positions	Data frame with margin positions and settlement prices.
bars	Profile-tagged market bars.
fx_rates	Data frame with currency and rate_to_base.
settlements	Durable engine settings and settlement inputs.
corporate_actions	A durable input table. Rows with asset_i, asset_j, and covariance provide covariance-margin inputs. Explicit bond lifecycle rows use asset_id, action_type (coupon, bond_accrual, or redemption), amount, currency, and optional effective_timestamp; they mutate settled cash and emit typed events at the eligible account boundary. Calendar rows use schedule_type = "bond" with coupon, accrual-cursor, and maturity fields from sim_bond_schedule_add() . C++ carries accrued interest on the typed inventory position, emits non-cash accrual events, settles due coupons, and redeems inventory at maturity.
orders	A normalized heterogeneous order batch.
timestamp	Market-boundary timestamp.

Value

Updated account state, typed events, fills, and group outcomes.

sim_heterogeneous_order_batch_schema

Empty normalized heterogeneous order-batch schema

Description

The schema carries generic order identity and eligibility fields plus derivative compatibility fields: action/direction codes, strategy/action identifiers, target-derived admission flag, per-asset quantity step, and funding settings. Inventory adapters may leave the compatibility fields at their typed defaults.

Usage

```
sim_heterogeneous_order_batch_schema()
```

Value

A typed empty data.table accepted by [sim_heterogeneous_account_step\(\)](#).

sim_import	<i>Import exported simulation tables</i>
------------	--

Description

Import exported simulation tables

Usage

sim_import(path)

Arguments

path Export directory created by sim_export().

Value

A named list containing manifest, simulation, and available durable tables.

sim_instrument_profile	<i>Resolve an instrument profile</i>
------------------------	--------------------------------------

Description

Resolve an instrument profile

Usage

sim_instrument_profile(instrument_profile)

Arguments

instrument_profile
 Supported profile name.

Value

A one-row data.table of profile defaults.

sim_instrument_profiles

List supported instrument profiles

Description

Instrument profiles define accounting and calendar defaults. They are intentionally declarative: the current C++ execution kernel still uses the registered contract multiplier and quantity step as its numeric inputs. The `synthetic_price_return` profile models signed marked price exposure only. It makes no custody, borrow availability or cost, dividend, funding, carry, settlement, or corporate-action claims, and is therefore suitable for stated price-return simulations rather than brokerage execution. Target-derived fee scaling preserves target ratios subject to contract-step rounding and records `fee_scaled` fills as durable partial execution-quality outcomes. Explicit contract orders remain all-or-nothing.

Usage

```
sim_instrument_profiles()
```

Value

A `data.table` of supported profiles and their defaults.

sim_live_service

Create a local live exchange service

Description

Builds an optional plumber app exposing local HTTP endpoints for agent commands. The service is a thin API over append-only command logs and the existing exchange APIs.

Usage

```
sim_live_service(exchange = sim_exchange_new())
```

Arguments

`exchange` A `tradesimr_exchange`.

Value

A plumber router.

sim_live_service_run *Run a local live exchange service*

Description

Run a local live exchange service

Usage

```
sim_live_service_run(
  exchange = sim_exchange_new(),
  host = "127.0.0.1",
  port = 8080
)
```

Arguments

exchange	A tradesimr_exchange.
host	Host interface.
port	Port.

Value

The result of plumber's run() method.

sim_live_state_dashboard_open
Open a live-state dashboard

Description

Open a live-state dashboard

Usage

```
sim_live_state_dashboard_open(
  exchange = sim_exchange_new(),
  path = tempfile("tradesimr-live-state-")
)
```

Arguments

exchange	A tradesimr_exchange.
path	Output directory.

Value

Invisibly returns the dashboard index path.

sim_manifest	<i>Build an export manifest</i>
--------------	---------------------------------

Description

Build an export manifest

Usage

```
sim_manifest(paths, tables, format, config = list())
```

Arguments

paths	Named character vector of exported files.
tables	Named list of exported tables.
format	Export file format.
config	Optional simulation configuration.

Value

A data.table manifest.

sim_market_events	<i>Extract market bars from a simulation</i>
-------------------	--

Description

Extract market bars from a simulation

Usage

```
sim_market_events(sim)
```

Arguments

sim	A simulation result returned by <code>sim_backtest()</code> or <code>sim_exchange_step()</code> .
-----	---

Value

A data.table with timestamp, open, high, low, and close.

sim_market_model_calibrate

Calibrate a market model from historical OHLC bars

Description

Estimates per-asset drift, volatility, AR coefficients, GARCH-like variance persistence, covariance/correlation, one-factor loadings, and simple low/high-volatility regime covariance from historical closes.

Usage

```
sim_market_model_calibrate(
  bars,
  model = c("multi_asset_random_walk", "multi_asset_ar_garch", "factor_random_walk",
    "regime_random_walk"),
  ar_order = 1L,
  seed = 1L,
  timeframe = NULL,
  tz = "UTC"
)
```

Arguments

bars	Market bars coercible by <code>as_market_bars()</code> .
model	Market model to configure from the calibration.
ar_order	Number of autoregressive lags to estimate per asset.
seed	Simulation seed recorded in the returned config.
timeframe, tz	Optional scheduling metadata for the returned config.

Value

A `sim_market_model_config()` list with calibration metadata.

sim_market_model_calibrate_exchange

Configure an exchange market model from historical bars

Description

Configure an exchange market model from historical bars

Usage

```
sim_market_model_calibrate_exchange(
  exchange,
  bars,
  model = c("multi_asset_random_walk", "multi_asset_ar_garch", "factor_random_walk",
    "regime_random_walk"),
  ar_order = 1L,
  seed = 1L,
  timeframe = NULL,
  tz = "UTC"
)
```

Arguments

exchange	A tradesimr_exchange.
bars	Market bars coercible by as_market_bars().
model	Market model to configure from the calibration.
ar_order	Number of autoregressive lags to estimate per asset.
seed	Simulation seed recorded in the returned config.
timeframe, tz	Optional scheduling metadata for the returned config.

Value

The calibrated market model config, invisibly.

sim_market_model_config

Configure market-level multi-asset simulation

Description

Configure market-level multi-asset simulation

Usage

```
sim_market_model_config(
  model = c("independent", "multi_asset_random_walk", "multi_asset_ar_garch",
    "factor_random_walk", "regime_random_walk"),
  corr = NULL,
  cov = NULL,
  factors = NULL,
  regimes = NULL,
  calibration = NULL,
  seed = 1L,
  timeframe = NULL,
```

```

    tz = NULL,
    start_time = NULL
)

```

Arguments

model	Market model. independent keeps per-asset feeds independent. Other values generate synchronized multi-asset return batches.
corr	Optional static correlation matrix.
cov	Optional static covariance matrix. If supplied, it takes precedence over corr and per-asset vol.
factors	Optional factor model settings for factor_random_walk.
regimes	Optional regime settings for regime_random_walk.
calibration	Optional calibration object from sim_market_model_calibrate().
seed	Base random seed for synchronized draws.
timeframe	Optional common timeframe. If omitted, all selected feeds must share the same timeframe.
tz	Optional common time zone. If omitted, all selected feeds must share the same time zone.
start_time	Optional first completed boundary to process.

Value

A market model configuration list.

```
sim_market_model_configure
```

Configure a market-level simulation model

Description

Configure a market-level simulation model

Usage

```
sim_market_model_configure(exchange, config = sim_market_model_config())
```

Arguments

exchange	A tradesimr_exchange.
config	A list from sim_market_model_config().

Value

The market model configuration, invisibly.

`sim_market_model_status`*Get market-level simulation model status*

Description

Get market-level simulation model status

Usage

```
sim_market_model_status(exchange)
```

Arguments

`exchange` A `tradesimr_exchange`.

Value

A list describing the market-level simulation model.

`sim_metrics`*Calculate core performance metrics from a simulation result*

Description

Calculate core performance metrics from a simulation result

Usage

```
sim_metrics(sim)
```

Arguments

`sim` A simulation result from `sim_backtest()`.

Value

A one-row `data.table` with return, drawdown, and event counts.

sim_orders	<i>Convert a simulation recorder into an order/event table</i>
------------	--

Description

Convert a simulation recorder into an order/event table

Usage

```
sim_orders(x)
```

Arguments

x	A simulation result returned by <code>sim_backtest()</code> or a raw recorder list from the C++ engine.
---	---

Value

A data.table of recorded execution events.

sim_portfolio_decision_policy	<i>Define a portfolio decision policy</i>
-------------------------------	---

Description

A policy controls which completed market observations may create a target-weight decision. It never changes execution eligibility: every order still fills only on a strictly later, tradable bar for its own asset.

Usage

```
sim_portfolio_decision_policy(
  mode = c("complete_universe", "as_of_valuation", "per_asset_decision"),
  max_staleness = Inf
)
```

Arguments

mode	Decision policy. "complete_universe" requires a fresh, tradable completed bar for every allowed asset. "as_of_valuation" permits carried marks for absent assets, subject to max_staleness. "per_asset_decision" permits partial decisions, but every named target must have a fresh, tradable completed bar in the submitted batch.
max_staleness	Maximum age in seconds of a carried valuation for "as_of_valuation". Defaults to Inf; use a finite value in production.

Value

A validated decision-policy list.

sim_portfolio_execution

Build execution assumptions for a target-weight portfolio replay

Description

Build execution assumptions for a target-weight portfolio replay

Usage

```
sim_portfolio_execution(
    timing = "next_eligible_open",
    fee_rt = 0,
    maker_fee_rt = NA_real_,
    slippage = 0,
    spread = 0,
    lev = 1,
    mmr = 0.02,
    max_gross_weight = 1
)
```

Arguments

timing	Execution timing. Phase 1 supports only "next_eligible_open": a decision made from a completed bar can first fill on a strictly later bar for the same asset.
fee_rt	Taker fee rate applied to filled notional.
maker_fee_rt	Optional maker fee rate for future limit-order support.
slippage	Absolute adverse price adjustment per unit.
spread	Absolute bid/ask spread; half is applied adversely on fills.
lev	Leverage used for initial-margin checks.
mmr	Maintenance-margin rate.
max_gross_weight	Maximum sum of absolute target weights.

Value

A named execution configuration list.

sim_portfolio_execution_quality

Project execution quality for durable portfolio target rebalances

Description

The projection evaluates each target using its decision-time target record, canonical order lifecycle, durable fills, and the position/account snapshot at the relevant settlement boundary. It never uses a later current mark to classify an earlier rebalance. Filled fee-aware target orders are assessed against their exact C++-recorded executable quantity.

Usage

```
sim_portfolio_execution_quality(exchange, agent_id = NULL, summary = FALSE)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Optional account identifier used to filter the projection.
summary	If FALSE (the default), return one row per rebalance and symbol. If TRUE, return one rebalance-level row whose quality is the worst component quality.

Value

A public-safe data.table. Symbol rows contain rebalance_id, agent_id, symbol, asset_id, decision_timestamp, eligible_after, settlement_timestamp, target_weight, decision_equity, decision_price, qty_step, contract_size, current_signed_quantity, expected_signed_quantity, expected_notional, realized_signed_quantity, realized_notional, quantity_deviation, notional_deviation, weight_deviation, execution_quality, and message.

sim_portfolio_export *Export a safe portfolio replay snapshot for an external consumer*

Description

Export a safe portfolio replay snapshot for an external consumer

Usage

```
sim_portfolio_export(exchange, agent_id, path, format = "json")
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Agent identifier.
path	Output directory.
format	Export format. Phase 1 supports JSON.

Value

A named vector of written paths. `fills.json` is sourced from the durable portfolio fill ledger and links every filled portfolio order to its `order_id` and `rebalance_id`; `rebalances.json` contains the linked accepted/rejected rebalance records.

sim_portfolio_market_step

Advance an Arena exchange at one completed market boundary

Description

This API advances a timestamped batch of completed bars exactly once. It executes only orders submitted at earlier boundaries; it never creates a target decision or rebalance. Each supplied asset bar must be genuinely new, so duplicate and stale batches fail rather than being silently reprocessed.

Usage

```
sim_portfolio_market_step(
  exchange,
  bars,
  execution = sim_portfolio_execution()
)
```

Arguments

exchange	A tradesimr_exchange.
bars	One timestamped batch of registered, completed OHLC bars.
execution	Execution assumptions from <code>sim_portfolio_execution()</code> .

Value

A public-safe list with bars, fills, events, positions, account, and outcomes.

sim_portfolio_step	<i>Step the C++ portfolio-margin kernel once</i>
--------------------	--

Description

Processes one timestamp batch of bars and explicit orders for one agent under one shared cash balance. This is the multi-asset primitive used by live exchanges when `portfolio_margin = TRUE`.

Usage

```
sim_portfolio_step(
  states,
  bars,
  orders = data.frame(),
  cov = diag(nrow(as_market_bars(bars))),
  shared_cash = 10000,
  ctr_size = 1,
  ctr_step = 1,
  lev = 10,
  fee_rt = 0,
  maker_fee_rt = NA_real_,
  taker_fee_rt = NA_real_,
  fund_rt = 0,
  funding_interval_hours = 8,
  mmr = 0.02,
  portfolio_margin_sigma = 3,
  portfolio_margin_floor = mmr,
  slippage = 0,
  spread = 0,
  record = TRUE
)
```

Arguments

states	Named list of prior <code>sim_state()</code> objects keyed by <code>asset_id</code> .
bars	One timestamp batch of market bars.
orders	Order data frame with <code>asset_id</code> , <code>action</code> , <code>dir</code> , <code>order_type</code> , <code>ctr_qty</code> , <code>price</code> , <code>strat_id</code> , and <code>action_id</code> .
cov	Return covariance matrix aligned to <code>bars\$asset_id</code> .
shared_cash	Shared account cash before this step.
ctr_size, ctr_step	Contract size and quantity step. Supply one value for all assets or one value per row of bars, aligned to <code>asset_id</code> .
lev	Leverage used for initial margin.

fee_rt	Trading fee rate on notional. Fees are reserved by target-derived opening/increase actions at their fill boundary; explicit contract orders continue to fail if their requested quantity is infeasible.
maker_fee_rt, taker_fee_rt	Optional maker/taker fee rates. Missing values fall back to fee_rt.
fund_rt	Funding rate per 8 hours on notional.
funding_interval_hours	Funding interval in hours.
mmr	Maintenance margin rate.
portfolio_margin_sigma	Sigma multiplier for covariance margin.
portfolio_margin_floor	Floor margin rate applied to gross exposure.
slippage	Absolute slippage added against trade direction.
spread	Absolute bid/ask spread; half spread is added against trade direction.
record	Whether to attach the execution recorder.

Value

A list with states, cash, equity, maintenance_margin, liquidated, and events. The state-list input is a compatibility projection; exchange and replay callers should use the typed heterogeneous account route.

sim_portfolio_target_replay

Replay a historical multi-asset target-weight panel

Description

This bulk API is intended for historical reconstruction. It preserves the public market-boundary and target-submission semantics while accumulating durable snapshots and events in batches rather than repeatedly growing their history tables. Live callers should continue to use [sim_portfolio_market_step\(\)](#) and [sim_portfolio_target_submit_batch\(\)](#).

Usage

```
sim_portfolio_target_replay(
    exchange,
    bars,
    target_weights,
    allowed_symbols = NULL,
    execution = sim_portfolio_execution(),
    decision_policy = sim_portfolio_decision_policy(),
    rebalance_policy = NULL,
    export_path = NULL,
```

```

    profile = FALSE,
    production_calendar = FALSE
)

```

Arguments

exchange	A tradesimr_exchange.
bars	Completed OHLC bars with timestamp, symbol, asset_id, open, high, low, and close. Each timestamp is one market boundary.
target_weights	A data frame with timestamp, agent_id, symbol, and target_weight. Each agent/timestamp group is submitted atomically.
allowed_symbols	Optional named list of allowed-symbol vectors by agent. When omitted, each agent's universe is inferred from all symbols in its panel rows. Multi-asset target groups are submitted only at boundaries containing one completed bar for every allowed symbol; incomplete groups are treated as absent decisions.
execution	Execution assumptions from sim_portfolio_execution() .
decision_policy	Market-observation policy from sim_portfolio_decision_policy() . Applied independently to each agent decision after the market boundary has been accepted.
rebalance_policy	Optional policy for sparse deterministic target panels. NULL (the default) preserves historical behavior and submits every agent/timestamp target group. When supplied, it must be a list with rebalance_due_column (default "rebalance_due") and a non-negative drift_tolerance. A group is submitted only when its due flag is TRUE and either its target vector differs from the last submitted vector or a supplied symbol's post-market realized-weight drift exceeds the tolerance. A skipped group is an absent decision: it creates no target, rebalance, or order record.
export_path	Optional directory for public-safe per-agent exports.
profile	Whether to return wall-time categories.
production_calendar	When TRUE, require an explicit exchange calendar_mode of "calendarize" or "strict"; production replay may not silently use legacy raw-bar admission.

Value

A list with the exchange, durable orders/fills/positions/accounts, targets/rebalances, execution quality, optional export paths, and timings.

sim_portfolio_target_step

Step one agent portfolio from target weights

Description

This is the stable Vox Arena integration point. It accepts a batch of genuinely new completed OHLC bars and, optionally, an explicit target-weight decision. Existing eligible orders are stepped first. A new target decision is then translated atomically into contract orders using current account equity and the decision-bar close (or the latest carried valuation for an asset absent from the batch). New orders are eligible only on a strictly later bar for their own asset. At that later fill boundary, target-derived opening/increase orders are capped to the largest step-rounded quantity whose fee and initial margin fit the account. Consequently a target weight of 1 at lev = 1 is near 100% notional after reserving the fee, rather than a failed all-cash order. Explicit contract orders retain reject-on-insufficient margin semantics.

Usage

```
sim_portfolio_target_step(
    exchange,
    agent_id,
    bars,
    target_weights = NULL,
    execution = sim_portfolio_execution(),
    decision_label = "target_weight",
    allowed_symbols = NULL,
    allowed_asset_ids = NULL,
    decision_policy = sim_portfolio_decision_policy()
)
```

Arguments

exchange	A <code>tradesimr_exchange</code> .
agent_id	Account identifier. Each agent has an isolated account.
bars	One timestamped batch of registered, completed OHLC bars.
target_weights	Optional named numeric target weights keyed by registered symbols.
execution	Execution assumptions from <code>sim_portfolio_execution()</code> .
decision_label	Optional durable label for the decision source.
allowed_symbols	Optional registered symbols this agent may target, hold, or trade.
allowed_asset_ids	Optional registered asset ids this agent may target, hold, or trade.
decision_policy	Market-observation policy from <code>sim_portfolio_decision_policy()</code> .

Details

A named target vector updates only its named symbols; supply an explicit zero target weight to flatten an asset. A NULL target is a no-decision: positions are retained and no new orders are submitted. Repeated/stale bars are ignored, so closed markets can retain their last valuation without creating strategy reactions or fills.

Value

A list containing orders, fills, positions, account, targets, realized_weights, and outcomes.

sim_portfolio_target_submit

Submit one Arena target-weight decision after a market boundary

Description

The supplied decision bars must already have been accepted by `sim_portfolio_market_step()`. Targets are converted atomically using the completed-bar close and the post-step account state. Submitted orders are eligible only on a strictly later bar of the same asset. NULL records a durable no-decision outcome and preserves current positions. Target-derived opening and increasing orders are clipped down to the largest executable contract-step quantity when fees or shared portfolio margin make the exact target infeasible. Explicit contract orders remain all-or-nothing. For inventory-profile target groups, fee-aware scaling is applied once to the atomic group, preserving target ratios subject to contract-step rounding. Such fills retain the durable `fee_scaled` reason code and are reported as partial execution-quality outcomes when the requested target is not fully reached. A later accepted target decision supersedes still-unfilled target-derived orders for the same agent and overlapping allowed assets; explicit contract orders are never superseded. For a multi-asset allowed universe, a non-NULL target decision is accepted only when the same market boundary contains exactly one completed bar for every allowed asset. This prevents target allocation from being planned from a partial cross-asset information set. Single-asset submissions are unaffected.

Usage

```
sim_portfolio_target_submit(
    exchange,
    agent_id,
    bars,
    target_weights = NULL,
    execution = sim_portfolio_execution(),
    decision_label = "target_weight",
    allowed_symbols = NULL,
    allowed_asset_ids = NULL,
    decision_policy = sim_portfolio_decision_policy()
)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Account identifier. Each agent has an isolated account.
bars	The already accepted timestamped completed-bar batch.
target_weights	Optional named numeric target weights keyed by registered symbols.
execution	Execution assumptions from <code>sim_portfolio_execution()</code> .
decision_label	Optional durable label for the decision source.
allowed_symbols	Optional registered symbols this agent may target, hold, or trade. Persisted on the agent after a successful submission.
allowed_asset_ids	Optional registered asset ids this agent may target, hold, or trade. When supplied with <code>allowed_symbols</code> , both must identify the same allowed universe.
decision_policy	Market-observation policy from <code>sim_portfolio_decision_policy()</code> .

Value

A list containing orders, fills, positions, account, targets, realized_weights, and outcomes.

sim_portfolio_target_submit_batch

Submit multiple Arena target-weight decisions after one market boundary

Description

This is the efficient replay interface for a common completed market boundary. It does not step market data. Instead, it validates the accepted boundary once and snapshots account, position, and carried-price state once before translating each agent's decision atomically. Each decisions element is a named list containing `target_weights` (or NULL for a no-decision), and optionally `decision_label`, `allowed_symbols`, and `allowed_asset_ids`. A multi-asset decision requires exactly one completed bar for every allowed asset at the common decision boundary.

Usage

```
sim_portfolio_target_submit_batch(
  exchange,
  bars,
  decisions,
  execution = sim_portfolio_execution(),
  decision_policy = sim_portfolio_decision_policy()
)
```

Arguments

exchange	A tradesimr_exchange.
bars	The already accepted timestamped completed-bar batch.
decisions	A named list keyed by agent_id.
execution	Execution assumptions from sim_portfolio_execution().
decision_policy	Market-observation policy applied to every decision.

Value

A list with the common boundary timestamp, public account and position snapshots, and a named submissions list containing the same result contract as sim_portfolio_target_submit() for each agent.

sim_positions	<i>Extract position snapshots from a simulation</i>
---------------	---

Description

Extract position snapshots from a simulation

Usage

```
sim_positions(sim)
```

Arguments

sim	A simulation result returned by sim_backtest().
-----	---

Value

A data.table of bar-level position snapshots.

sim_read_account	<i>Read exported account snapshots</i>
------------------	--

Description

Read exported account snapshots

Usage

```
sim_read_account(path)
```

Arguments

path	Export directory created by sim_export().
------	---

Value

A data.table of account snapshots.

sim_read_events	<i>Read exported simulation events</i>
-----------------	--

Description

Read exported simulation events

Usage

```
sim_read_events(path)
```

Arguments

path	Export directory created by sim_export().
------	---

Value

A data.table of events.

sim_read_manifest	<i>Read an export manifest</i>
-------------------	--------------------------------

Description

Read an export manifest

Usage

```
sim_read_manifest(path)
```

Arguments

path	Export directory or manifest file path.
------	---

Value

A data.table manifest.

sim_read_table	<i>Read an exported simulation table</i>
----------------	--

Description

Read an exported simulation table

Usage

```
sim_read_table(path, table, format = NULL)
```

Arguments

path	Export directory or table file path.
table	Table name when path is a directory.
format	Optional format. Inferred from manifest or file extension when absent.

Value

A data.table.

`sim_replay`*Replay historical bars through the simulation engine*

Description

Alias for `sim_backtest()` reserved for replay-oriented workflows.

Usage

```
sim_replay(data, ...)
```

Arguments

<code>data</code>	A data frame/data.table with timestamp, open, high, low, close, and target-position columns.
<code>...</code>	Additional arguments passed to <code>sim_backtest()</code> .

`sim_replay_dashboard_export`*Export a static replay dashboard*

Description

Writes dashboard-ready CSV tables, a manifest, and static HTML/CSS/JS assets. The dashboard is a read-only consumer of durable market, strategy, event, account, risk, order, and fill tables; it does not call C++, R6, or live exchange internals.

Usage

```
sim_replay_dashboard_export(sim, path)
```

Arguments

<code>sim</code>	A simulation result returned by <code>sim_backtest()</code> or <code>sim_exchange_step()</code> .
<code>path</code>	Output directory.

Value

Invisibly returns a named character vector of written files.

sim_risk	<i>Extract risk snapshots from a simulation</i>
----------	---

Description

Extract risk snapshots from a simulation

Usage

```
sim_risk(sim)
```

Arguments

sim A simulation result returned by `sim_backtest()`.

Value

A `data.table` of bar-level risk snapshots.

sim_run_from_events	<i>Reconstruct simulation views from exported events</i>
---------------------	--

Description

Reconstruct simulation views from exported events

Usage

```
sim_run_from_events(path)
```

Arguments

path Export directory created by `sim_export()`.

Value

A `data.table` simulation result when a simulation table exists, otherwise event-level account state reconstructed from events.

sim_schemas	<i>Simulation table schemas</i>
-------------	---------------------------------

Description

Simulation table schemas

Usage

```
sim_schemas()
```

Value

A named list of empty `data.tables` representing durable simulation table schemas.

sim_schema_migrate	<i>Migrate durable tables to the current schema</i>
--------------------	---

Description

Missing columns are added with typed NA values and existing columns are retained unchanged. This makes older CSV/fst exports readable without silently discarding consumer-defined extension columns.

Usage

```
sim_schema_migrate(tables, from_version = NULL)
```

Arguments

tables	A named list of durable data tables.
from_version	Optional source schema version retained in the returned metadata for audit. Missing fields are migrated deterministically.

Value

A named list upgraded to `sim_schema_version()`.

sim_schema_version	<i>Get the tradesimr schema version</i>
--------------------	---

Description

Get the tradesimr schema version

Usage

```
sim_schema_version()
```

Value

A scalar character schema version.

sim_spot_step	<i>Step a fully paid spot-inventory state</i>
---------------	---

Description

This Rcpp-backed state machine is separate from the derivatives margin kernel. It models long-only inventory, quote-currency cash, fees, marked inventory value, dividends, and stock splits.

Usage

```
sim_spot_step(
  state = NULL,
  close,
  signed_qty = 0,
  execution_price = NA_real_,
  contract_size = 1,
  fee_rt = 0,
  dividend_per_unit = 0,
  split_ratio = 1
)
```

Arguments

state	A prior spot state, or NULL for a new account.
close	Current mark price.
signed_qty	Positive to buy units and negative to sell units.
execution_price	Optional execution price; defaults to close.
contract_size	Units represented by one inventory unit.

fee_rt	Fee rate applied to execution notional.
dividend_per_unit	Cash dividend per unit.
split_ratio	Inventory split ratio.

Value

A list containing cash, units, average cost, market value, equity, P&L, fees, corporate-action cash, and execution status.

sim_spot_target_submit

Submit spot target weights for next-bar execution

Description

This is the inventory-accounting counterpart to the derivatives portfolio target API. It plans all supplied spot legs from one completed boundary and emits only next-eligible explicit inventory orders.

Usage

```
sim_spot_target_submit(exchange, agent_id, bars, target_weights, fee_rt = 0)
```

Arguments

exchange	A tradesimr_exchange.
agent_id	Account identifier.
bars	Completed registered spot bars at one timestamp.
target_weights	Named target weights keyed by symbols.
fee_rt	Non-negative execution fee rate.

Value

A list of accepted orders and planned target quantities.

`sim_state`*Create a simulation state object*

Description

Create a simulation state object

Usage

```
sim_state(  
  cash = 10000,  
  pos_dir = 0L,  
  ctr_unit = 0,  
  avg_price = NA_real_,  
  last_px = 0,  
  strat = 0L,  
  asset = 0L,  
  action_id_now = 1L,  
  old_timestamp = NA_real_,  
  liquidated = FALSE  
)
```

Arguments

<code>cash</code>	Account cash.
<code>pos_dir</code>	Position direction: -1, 0, or 1.
<code>ctr_unit</code>	Contract units held.
<code>avg_price</code>	Average entry price.
<code>last_px</code>	Last mark price.
<code>strat, asset</code>	Integer identifiers.
<code>action_id_now</code>	Next action id.
<code>old_timestamp</code>	Previous bar timestamp, used for funding accrual.
<code>liquidated</code>	Whether the account is liquidated.

Value

A list suitable for `sim_step()`.

sim_state_dashboard_export	<i>Export a live-state dashboard</i>
----------------------------	--------------------------------------

Description

Writes the current exchange state using the god-facing live-state dashboard shell. This dashboard can configure and step the feed but cannot place orders.

Usage

```
sim_state_dashboard_export(exchange, path)
```

Arguments

exchange	A tradesimr_exchange.
path	Output directory.

Value

Invisibly returns a named character vector of written files.

sim_step	<i>Step the C++ exchange kernel once</i>
----------	--

Description

Processes one bar and a batch of explicit order actions against prior account state. This is the incremental primitive underneath future live exchange workflows; it does not replay earlier bars.

Usage

```
sim_step(
  state,
  bar,
  orders = data.frame(),
  asset = state$asset %||% 0L,
  ctr_size = 1,
  ctr_step = 1,
  lev = 10,
  fee_rt = 0,
  maker_fee_rt = NA_real_,
  taker_fee_rt = NA_real_,
  fund_rt = 0,
  funding_interval_hours = 8,
```

```

    mmr = 0.02,
    slippage = 0,
    spread = 0,
    record = TRUE
)

```

Arguments

state	Prior state created by <code>sim_state()</code> or returned by <code>sim_step()</code> .
bar	One-row market bar coercible by <code>as_market_bars()</code> .
orders	Data frame with order columns: <code>action</code> , <code>dir</code> , <code>order_type</code> , <code>ctr_qty</code> , <code>price</code> , and optional <code>strat_id</code> , <code>action_id</code> , and <code>fee_aware_target</code> . The last flag is used internally for target-position and target-weight orders, whose opening quantity is capped at the fill price to reserve fees and initial margin.
asset	Integer asset identifier.
ctr_size	Contract size.
ctr_step	Minimum contract increment.
lev	Leverage used for initial margin.
fee_rt	Trading fee rate on notional. Fees are reserved by target-derived opening/increase actions at their fill boundary; explicit contract orders continue to fail if their requested quantity is infeasible.
maker_fee_rt, taker_fee_rt	Optional maker/taker fee rates. Missing values fall back to <code>fee_rt</code> .
fund_rt	Funding rate per 8 hours on notional.
funding_interval_hours	Funding interval in hours.
mmr	Maintenance margin rate.
slippage	Absolute slippage added against trade direction.
spread	Absolute bid/ask spread; half spread is added against trade direction.
record	Whether to attach the execution recorder.

Value

A list with state and events.

sim_strategy_list	<i>List registered strategy ids</i>
-------------------	-------------------------------------

Description

List registered strategy ids

Usage

sim_strategy_list(exchange)

Arguments

exchange A tradesimr_exchange.

Value

Character vector of registered strategy ids.

sim_strategy_register	<i>Register a strategy function for strategy-backed AI agents</i>
-----------------------	---

Description

Registered functions are runtime-only and are not serialized into exported CSV files. Store the durable strategy name in the agent config via strategy_id; register the function again after loading a saved exchange.

Usage

sim_strategy_register(exchange, strategy_id, strategy_fn)

Arguments

exchange A tradesimr_exchange.
strategy_id Strategy identifier used by agent config.
strategy_fn Function returning target positions, strategyr action plans, or order-intent rows.

Value

The strategy id, invisibly.

```
sim_strategy_unregister
```

Unregister a strategy function

Description

Unregister a strategy function

Usage

```
sim_strategy_unregister(exchange, strategy_id)
```

Arguments

exchange	A tradesimr_exchange.
strategy_id	Strategy identifier used by agent config.

Value

Invisibly returns TRUE when a strategy was removed.

```
sim_strategy_validate_config
```

Validate strategy-backed agent config

Description

Checks that a strategy agent config points to a registered function or a resolvable function name, and that supplied param_ keys are compatible with the strategy function signature when the function does not accept

Usage

```
sim_strategy_validate_config(exchange, config)
```

Arguments

exchange	A tradesimr_exchange.
config	Agent config list.

Value

A list with valid, message, strategy_id, strategy_fun, and params.

sim_submit_order	<i>Submit an agent order command</i>
------------------	--------------------------------------

Description

Appends an order request to the exchange command log. By default the command is processed immediately into the same exchange order model used by `sim_exchange_step()`.

Usage

```
sim_submit_order(
  exchange,
  agent_id,
  timestamp = Sys.time(),
  symbol = NULL,
  asset_id = NULL,
  tgt_pos = NULL,
  tol_pos = 0,
  order_type = c("market", "limit"),
  side = c("target", "buy", "sell", "flat"),
  qty_type = NULL,
  qty = NULL,
  limit_price = NA_real_,
  time_in_force = "gtc",
  client_order_id = NA_character_,
  process = TRUE
)
```

Arguments

exchange	A <code>tradesimr_exchange</code> .
agent_id	Agent identifier.
timestamp	Order timestamp.
symbol	Registered asset symbol.
asset_id	Registered asset identifier. Provide <code>symbol</code> , <code>asset_id</code> , or both when they identify the same asset.
tgt_pos	Target exposure. Kept for compatibility with earlier intent-level calls.
tol_pos	Target-position tolerance.
order_type	Order type: <code>market</code> or <code>limit</code> .
side	Order side: <code>target</code> , <code>buy</code> , <code>sell</code> , or <code>flat</code> .
qty_type	Quantity semantics: <code>contracts</code> or <code>target_pos</code> .
qty	Order quantity. Meaning is controlled by <code>qty_type</code> .
limit_price	Optional limit price for limit orders.

time_in_force Time-in-force label.
client_order_id Optional client order id.
process Whether to process pending commands immediately.

Value

The generated command id.

sim_trading_calendars	<i>List built-in trading calendars</i>
-----------------------	--

Description

List built-in trading calendars

Usage

sim_trading_calendars()

Value

A data.table describing the deterministic built-in session rules.

TRADESIMR_ACCOUNT_SCHEMA_VERSION	<i>Heterogeneous account schema version</i>
----------------------------------	---

Description

Heterogeneous account schema version

Usage

TRADESIMR_ACCOUNT_SCHEMA_VERSION

Format

An object of class character of length 1.

TRADESIMR_SCHEMA_VERSION
<i>tradesimr durable schema version</i>

Description

tradesimr durable schema version

Usage

TRADESIMR_SCHEMA_VERSION

Format

An object of class character of length 1.

validate_intents	<i>Validate target-position intent columns</i>
------------------	--

Description

Validate target-position intent columns

Usage

validate_intents(data, tgt_pos_col = "tgt_pos", tol_pos_col = NULL)

Arguments

- | | |
|-------------|---------------------------------|
| data | A table-like object. |
| tgt_pos_col | Target-position column name. |
| tol_pos_col | Optional tolerance column name. |

Value

Invisibly returns TRUE on success.

validate_market_data	<i>Validate core market-bar columns</i>
----------------------	---

Description

Validate core market-bar columns

Usage

```
validate_market_data(
  data,
  timestamp_col = "timestamp",
  open_col = "open",
  high_col = "high",
  low_col = "low",
  close_col = "close"
)
```

Arguments

data	A table-like object.
timestamp_col, open_col, high_col, low_col, close_col	Column names.

Value

Invisibly returns TRUE on success.

vec_batch_run_simulations	<i>Generate vectorized simulation inputs for multiple instruments</i>
---------------------------	---

Description

Legacy helper that loads candle data through strategyr and applies signal strategies before forming long, short, and both-side position series.

Usage

```
vec_batch_run_simulations(
  inst_ids,
  bar,
  signal_strategies,
  root_path = Sys.getenv("OKX_Candle_Data_Path"),
  bg_time = NULL,
  ed_time = NULL
)
```

Arguments

inst_ids	Instrument identifiers.
bar	Bar size passed to strategyr loaders.
signal_strategies	Named list of functions that add a signal column.
root_path	Candle data root path.
bg_time	Optional begin time.
ed_time	Optional end time.

Value

A named list of data.tables.

vec_sim_gen_plot	<i>Plot vectorized simulation results</i>
------------------	---

Description

Plot vectorized simulation results

Usage

```
vec_sim_gen_plot(vec_sim_res, report_mode = c("simple", "full"))
```

Arguments

vec_sim_res	Result from vec_sim_run_backtest().
report_mode	Plot report mode.

vec_sim_gen_summary	<i>Summarize vectorized simulation results</i>
---------------------	--

Description

Summarize vectorized simulation results

Usage

```
vec_sim_gen_summary(
  vec_sim_res,
  report_mode = c("short_txt", "long_txt", "dt")
)
```

Arguments

vec_sim_res Result from vec_sim_run_backtest().
report_mode Summary mode.

Value

Text or a one-row data.table depending on report_mode.

vec_sim_gen_summary_table	<i>Summarize a batch of vectorized simulations</i>
---------------------------	--

Description

Summarize a batch of vectorized simulations

Usage

vec_sim_gen_summary_table(DTs)

Arguments

DTs List of vectorized simulation data.tables.

Value

A data.table of summaries.

vec_sim_run_backtest	<i>Run a vectorized approximate backtest</i>
----------------------	--

Description

Lightweight legacy helper that converts a precomputed position column into log returns and an equity curve. It does not model the stateful exchange, order, margin, funding, or liquidation mechanics used by sim_backtest().

Usage

vec_sim_run_backtest(DT)

Arguments

DT A data.table containing at least datetime, close, and pos.

Value

A list containing equity series and summary statistics.

Index

* datasets

TRADESIMR_ACCOUNT_SCHEMA_VERSION,
84
TRADESIMR_SCHEMA_VERSION, 85

as_market_bars, 5
as_target_positions, 6

PaperTrader, 6
PaperTradingPlatform, 8

sim_account, 12
sim_agent_add, 13
sim_agent_command_schema, 14
sim_agent_dashboard_export, 14
sim_agent_dashboard_open, 15
sim_agent_rankings, 15
sim_agent_remove, 16
sim_agent_set_status, 16
sim_agents_step, 13
sim_asset_add, 17
sim_asset_remove, 18
sim_assets, 17
sim_backtest, 19
sim_bond_schedule_add, 21
sim_bond_schedule_add(), 51
sim_bond_schedules, 21
sim_calendar_expected_bars, 22
sim_calendar_holidays, 23
sim_calendar_is_open, 23
sim_calendar_settlement_timestamp, 24
sim_calendar_spec, 24
sim_cancel_order, 25
sim_cash_ledger, 26
sim_cross_asset_risk, 26
sim_dashboard_export, 27
sim_dashboard_open, 27
sim_events, 28
sim_exchange_account, 28
sim_exchange_account_state, 29

sim_exchange_accrue_carry, 29
sim_exchange_add_bars, 30
sim_exchange_calendar_exception, 31
sim_exchange_calendar_exception(), 23
sim_exchange_calendarize_bars, 30
sim_exchange_cancel_order, 32
sim_exchange_cash_adjust, 32
sim_exchange_cash_balances, 33
sim_exchange_convert_cash, 34
sim_exchange_corporate_action, 34
sim_exchange_dashboard, 35
sim_exchange_export_events, 36
sim_exchange_future_roll, 36
sim_exchange_fx_rate, 37
sim_exchange_load, 38
sim_exchange_new, 38
sim_exchange_new_events, 39
sim_exchange_orders, 39
sim_exchange_place_order, 40
sim_exchange_positions, 41
sim_exchange_process_commands, 41
sim_exchange_run, 42
sim_exchange_save, 42
sim_exchange_set_carry_rates, 43
sim_exchange_settle, 43
sim_exchange_step, 44
sim_exchange_validate_cadence, 44
sim_export, 45
sim_feed_config, 45
sim_feed_configure, 46
sim_feed_start, 47
sim_feed_status, 47
sim_feed_step, 48
sim_feed_stop, 48
sim_feed_warmup, 49
sim_fills, 50
sim_heterogeneous_account_step, 50
sim_heterogeneous_account_step(), 51
sim_heterogeneous_order_batch_schema,

51
sim_import, 52
sim_instrument_profile, 52
sim_instrument_profiles, 53
sim_live_service, 53
sim_live_service_run, 54
sim_live_state_dashboard_open, 54
sim_manifest, 55
sim_market_events, 55
sim_market_model_calibrate, 56
sim_market_model_calibrate_exchange,
56
sim_market_model_config, 57
sim_market_model_configure, 58
sim_market_model_status, 59
sim_metrics, 59
sim_orders, 60
sim_portfolio_decision_policy, 60
sim_portfolio_decision_policy(), 66, 67,
69
sim_portfolio_execution, 61
sim_portfolio_execution(), 66
sim_portfolio_execution_quality, 62
sim_portfolio_export, 62
sim_portfolio_market_step, 63
sim_portfolio_market_step(), 65
sim_portfolio_step, 64
sim_portfolio_target_replay, 65
sim_portfolio_target_step, 67
sim_portfolio_target_submit, 68
sim_portfolio_target_submit_batch, 69
sim_portfolio_target_submit_batch(),
65
sim_positions, 70
sim_read_account, 71
sim_read_events, 71
sim_read_manifest, 72
sim_read_table, 72
sim_replay, 73
sim_replay_dashboard_export, 73
sim_risk, 74
sim_run_from_events, 74
sim_schema_migrate, 75
sim_schema_version, 76
sim_schema_version(), 75
sim_schemas, 75
sim_spot_step, 76
sim_spot_target_submit, 77
sim_state, 78
sim_state_dashboard_export, 79
sim_step, 79
sim_strategy_list, 80
sim_strategy_register, 81
sim_strategy_unregister, 82
sim_strategy_validate_config, 82
sim_submit_order, 83
sim_trading_calendars, 84
TRADESIMR_ACCOUNT_SCHEMA_VERSION, 84
TRADESIMR_SCHEMA_VERSION, 85
validate_intents, 85
validate_market_data, 86
vec_batch_run_simulations, 86
vec_sim_gen_plot, 87
vec_sim_gen_summary, 87
vec_sim_gen_summary_table, 88
vec_sim_run_backtest, 88