QuantReplay REST API

QuantReplay REST API

Table of Contents

Introduction 4
Scope 4
General 5
REST Logic 5
REST Requests. 5
REST Usage 6
Base URL 6
Request Methods 6
HTTP GET 7
HTTP POST 7
HTTP PUT 8
HTTP DELETE 9
Data Format 10
Request Headers 10
Request Body 11
Response Body. 11
Strings 12
Numbers 12
Date/Time 12
Random Order Generation 13
Distribution of Actions 13
Distribution of Price. 16
Distribution of Quantity 18
Special Market Events 19
Data Source Playback 19
Format Specific Configuration. 20
CSV Files. 20
Databases. 22

Basic Single Level Order Book 22
Multiple Level Order Book 25
Market Phases 27
Open Phase 28
Closed Phase 28
Phase Halt 29
Persisted State 29
File Format 30
Data verification 34
Administrative Settings 47
General Settings 47
Get Settings 47
Update Settings 48
Venues 49
Market Phases Sub-List 52
Get Single Venue 54
Get Multiple Venues 56
Add Venue 57
Update Venue 59
Delete Single Venue 60
Listings 60
Get Single Listing 65
Get Multiple Listings 67
Add Listing 69
Update Listing 70
Data Sources 72
Column Mapping Sub-List 74
Get Single Data Source 77
Get Multiple Data Sources 78
Add Data Sources 79
Update Data Sources 81
Price Seeds 82
Get Single Price Seed 84

Get Multiple Price Seeds . . . 85
Add Price Seeds . . . 86
Update Price Seeds . . . 87
Delete Price Seeds . . . 88
Sync Price Seeds . . . 89
Administrative Commands . . . 90
System Status . . . 90
Get System Status . . . 90
Venue Status . . . 91
Get Single Venue Status . . . 91
Get Multiple Venue Status . . . 92
Order Generation . . . 93
Start/Stop Order Generation . . . 93
Status of Order Generation . . . 94
Market Phase Halt . . . 95
Halt Market Phase . . . 95
Resume Market Phase . . . 97
Persisted State . . . 98
Store Market State for Single Venue . . . 98
Recover Market State for Single Venue . . . 100

Introduction

This reference document provides details of the QuantReplay Market Simulator Representational State Transfer (REST) API.

Scope

This document provides technical-level details of using the administrative functions through the REST API of the QuantReplay Market Simulator, for modifying administrative settings and sending administrative requests to market simulator instances.

General

REST Logic

Market Simulator instances interact. So, if a person sends a request about Venue status from one Market Simulator, the system can receive it in another Market Simulator, process it and return a response. Final response to this person will be provided from the initial Market Simulator from which the person sent a request.

It is possible to check the status of the Market Simulator instance. All Market Simulators have REST patterns - types of requests that they can accept and process. If Market Simulator receives REST request that it cannot process, it returns an error - e.g.: *method is unavailable*.

The commands to Check Status, Stop/Start order Generation, check its status, etc. do not influence the Database, they influence only the work of the current simulator instance.

REST Requests

You can go straight to the Database and make the necessary changes for the Market Simulator instance, but it requires access to the Database. As an option, it is possible to send REST request from outside.

A client (e.g.: a Web interface) can make a request to the REST engine of one of the simulator instances - REST call to the REST port of one of the simulators - with the request to make some changes on the LSE instance, for example. It is possible to send status request, change the status,stop/start orders generation, etc. Market Simulator makes the requested changes in the Database. There is *NO* UM Messaging, Market Simulator instances communicate over REST.

REST Usage

All that we have in the Database should be accessible through REST. It is possible to compare the Database and GET REST request.

Through REST it is possible to:

- see some specific listing / venue / data sources
- see all listings / venues / price seed
- see venue / multiple venues status
- see order generation status
- start / stop order generation
- update some listing / venue / price seed / data sources /settings
- add listing / venue / price seed / data sources / settings
- send order

Market Simulator gets data from the Database at its start, as a result, Market Simulator provides only ***current*** data to REST.

> ⚠ Changes made directly to the database may require a restart of the simulator process in order to make these changes visible through the REST API.

Base URL

Requests to the Market Simulator REST API are unsecured, since the Market Simulator deals only with non-production simulated market data and trades in testing environments.

Request Methods

HTTP GET

Market Simulators use the GET request to retrieve information about the specified system resource, and can return the following response codes:

Status Code	Meaning
200 OK	Request was successful and the response body contains the requested resource.
404 NOT FOUND	The requested URI or resource identifier is unknown.
412 PRECONDITION FAILED	The value of the `X-API-Version` optional header mismatches with the simulator major version.
500 SERVER ERROR	Request could not be processed due to an internal server error.
503 SERVICE UNAVAILABLE	System is temporarily unable to process the request. Please wait and then try again.

HTTP POST

Market Simulators use the POST request to add one or more new resources to the system or make complex requests, and can return the following response codes:

Status Code	Meaning
201 CREATED	Request was successful, we added the new resource(s) and the response body contains the validated form of the resource(s) or the response to a complex request.

Status Code	**Meaning**
400 BAD REQUEST	The request failed validation, see response body for details.
403 FORBIDDEN	User is not allowed to execute requested action, see response body for details.
404 NOT FOUND	The requested URI is unknown.
405 METHOD NOT ALLOWED	POST is not valid for the specified resource.
409 CONFLICT	Request could not be processed due to an inconsistent server state, see response body for details.
412 PRECONDITION FAILED	The value of the `X-API-Version` optional header mismatches with the simulator major version.
500 SERVER ERROR	Request could not be processed due to an internal server error.
502 BAD GATEWAY	Request could not be forwarded to destination simulator instance, see response body for details.
503 SERVICE UNAVAILABLE	System is temporarily unable to process the request. Please wait and then try again.

HTTP PUT

Market Simulators use the PUT request to update one or more existing resources in the system, and can return the following response codes:

Status Code	Meaning
200 OK	Request was successful, we updated the resource(s) and the response body contains the validated form of the resource(s).
204 NO CONTENT	Request was successful, we updated the resource(s) and the response body is empty.
400 BAD REQUEST	The request failed validation, see response body for details.
404 NOT FOUND	The requested URI or resource identifier is unknown.
405 METHOD NOT ALLOWED	PUT is not valid for the specified resource.
412 PRECONDITION FAILED	The value of the `X-API-Version` optional header mismatches with the simulator major version.
500 SERVER ERROR	Request could not be processed due to an internal server error.
503 SERVICE UNAVAILABLE	System is temporarily unable to process the request. Please wait and then try again.

HTTP DELETE

Market Simulators use the DELETE request to remove one or more existing resources from the system, and can return the following response codes:

Status Code	Meaning
204 NO CONTENT	Request was successful, we deleted the resource(s) and the response body is empty.
404 NOT FOUND	The requested URI or resource identifier is unknown.
405 METHOD NOT ALLOWED	DELETE is not valid for the specified resource.
412 PRECONDITION FAILED	The value of the `X-API-Version` optional header mismatches with the simulator major version.
500 SERVER ERROR	Request could not be processed due to an internal server error.
503 SERVICE UNAVAILABLE	System is temporarily unable to process the request. Please wait and then try again.

Data Format

Request Headers

HTTP Header	Value type	Required	Meaning
X-API-Version	string	No	Defines the REST API version. If it mismatches with the simulator major version, the response is sent with status code 412 PRECONDITION FAILED. The equality of versions is only checked if the request contains the header.

Request Body

In the case of requests that contain a body (generally POST and PUT requests) the body must be sent in JSON format, using the following request header:

```
Content-Type: application/json
```

All property names in the request body must be double quoted. See below for further details on the format of specific types of request body values.

Response Body

In the case of responses that contain a body (generally GET, POST, and PUT responses) the body will be sent in JSON format. All property names in the response body will be double quoted. See below for further details on the format of specific types of response body values.

Strings

Any string values sent as part of a JSON request or response must be wrapped in double quotes. In the case where an indication of "no value" is required, a null property value may be specified.

Example

```
{
    "myStringProperty":"myStringValue",
    "myEmptyStringProperty":null
}
```

Numbers

Any numerical fields (Integer or Float) sent as part of a JSON request or response must not be wrapped in double quotes. In the case of Float values, a period (".") must be used as the decimal separator, regardless of a user’s regional settings. In the case where an indication of "no value" is required, a null property value may be specified.

Example

```
{
    "myIntegerProperty":123,
    "myFloatProperty":123.456,
    "myEmptyNumberProperty":null
}
```

Date/Time

All time stamps sent to or returned from Market Simulators must be expressed in UTC.

Time stamps with second granularity are expressed using the format: yyyy-MM-ddTHH:mm:ss

Example

```
2015-09-08T15:32:09
```

Time stamps with milli-second granularity are expressed using the format: yyyy-MM-ddTHH:mm:ss.SSS

Example

```
2015-09-08T15:32:09.119
```

Random Order Generation

Some notes on understanding how random order generation works in the market simulator.

Distribution of Actions

The simulator uses a separate pseudo random number generator per listing so that as each random action is determined based on probability distributions, as described below in greater detail, each subsequent action determined from the output of each pseudo random number generator for each listing is independent of any underlying threading model for managing the load across all active listings. Each of these separate pseudo random number generators is seeded by default with its own genuine random number provided by API calls specific to the platform where the simulator is running, so that the resulting sequence of actions will be different each time the simulator is started.

The frequency of random generation of orders is almost entirely dependent on Listings → randomOrdersRate. The primary setting controls the number of actions taken by the random order generator per second, including the resulting actions taken on resting orders (placing them, and amending or canceling them), as well as the frequency of matching orders resulting in trades. Some other settings, such as Venues → randomPartyCount and Listings

→ randomDepthLevels would also have some effect, but not as significant as randomOrdersRate.

The value entered for randomOrdersRate determines the frequency with which a timer will trigger in the market simulator to do an action. The specific action taken is randomly chosen each time by a statistically correct, evenly distributed random number generator.

There is a hard-coded 1/3 probability of the simulator choosing to "do nothing" as an action, therefore the effective applied frequency of choosing a random action (including "do nothing") is:

Actual Effective Random Action Frequency = (Configured randomOrdersRate value * 3) / 2

Of actual actions taken, there is a hard-coded probability distribution of:

40% probability of resting bid action

40% probability of resting ask action

10% probability of aggressive buy

10% probability of aggressive sell

That means that 80% of the time we should have an action on a resting order in the book, and 20% of the time we should have an action of an aggressive order that may match against a resting order in the book. Note however, aggressive orders will not be sent against an empty side of a book with no corresponding resting orders.

Venues → randomPartyCount value determines the number of counter parties that will be used to determine which party each order action is generated from, and each party is chosen using a simple round-robin choice of parties for each subsequent random action.

For example, if a value of randomPartyCount = 10 is configured, party #1 will be used for the first random action, party #2 will be used for the next action, and so on until we get to party #10 and then return back to party #1 and keep repeating.

Coming back to the probability of order actions, if the random action chosen is a resting bid (40% probability) or ask (40% probability) action, we then have an additional random choice of actions determined by the following logic and hard-coded probability distribution:

If a bid does not exist for a resting bid action (or an ask does not exist for a resting ask action) for the current counter party (as determined by the Venues → randomPartyCount setting described above), then simply place the new bid (or place the new ask). Otherwise, if there is already an existing resting bid (or ask) for the current counter party, use the following probability distribution:

45% probability of amend quantity of existing order

45% probability of amend price of existing order

10% probability of cancel existing order

So in summary, there is an 80% probability each second of performing an action on a resting order (either placing a new order for the party, otherwise 90% probability of amending, or 10% probability of canceling), and a 20% probability each second of performing an aggressive action (matching orders in the book, resulting in trades).

Coming back to our example of randomOrdersRate = 10, this should result in an average of about 8 resting order actions per second (new if no existing order for the party, otherwise about 7 amends, and about 1 cancel) and an average of about 2 matching order trades per second.

While randomOrdersRate is the primary setting determining frequency of actions taken by the simulator, another setting Listings → randomDepthLevels is worth noting. This determines the number of allowed depth levels in the book, such that if the random chosen action is to place a new resting bid or ask for the current counter party, it will not be done if it would mean exceeding this configured maximum number of levels in the book. Combined with the probability distribution of quantity and price as described in the following sections, this maximum number of depth levels could also limit the number of times there is an existing order in the book for a given counter party to perform amend or cancel actions on, and could also limit the range of quantity and price levels available for aggressive orders to match against. The deeper the book is allowed to grow, the closer the simulator will be able to apply the probabilities described above, without synthetically blocking an action due to a limit in the number of depth levels in the book.

Distribution of Price

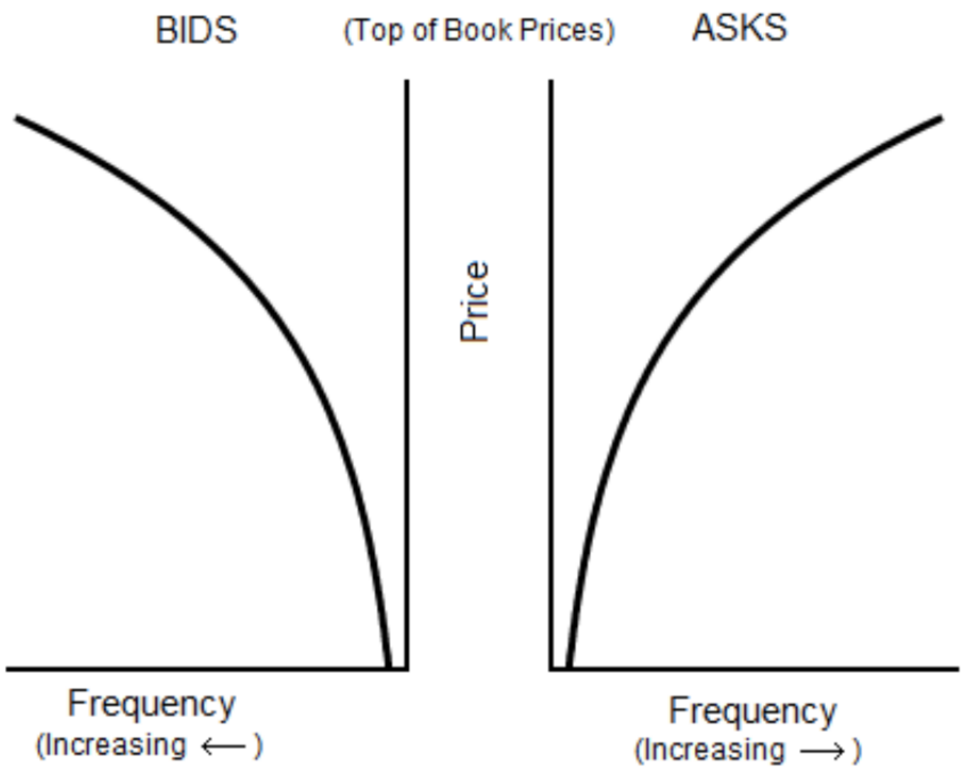
The price used on randomly generated orders is largely determined by the configuration of Listings → randomOrdersSpread to determine the minimum bid/ask spread to maintain, and Listings → randomTickRange to determine the range which prices are distributed across multiple depth levels of the order book.

The setting Listings → randomOrdersSpread is expressed as an absolute decimal price value, with an enforced minimum of the value of Listings → tickSize configured for the same listing. This effectively becomes the smallest spread allowed in the order book determined by generation of random orders. If the spread becomes less than this value (or crossed) it is likely due to other actions, such as historical data playback or manually entered orders.

Starting with an empty book, the initial value to consider when generating a new random price will be taken from a starting "seed" price as stored in a table of starting prices per instrument (across all listings of that instrument). After the book has resting orders, the initial value is taken from the best prices in the book. In both cases, the initial value is adjusted to incorporate the configured spread as described above. Specifically, prices for new resting Bid orders and aggressive Sell orders use an initial starting value of the best Ask price in the order book subtracting the configured spread, or the best Bid price directly if there are no resting Ask orders, or failing that the configured Bid price directly from the table of starting prices (or Mid price if there is no Bid). Respectively, prices for new resting Ask orders and aggressive Buy orders use an initial starting value of the best Bid price in the order book adding the configured spread, or the best Ask price directly if there are no resting Bid orders, or failing that the configured Bid price directly from the table of starting prices (or Mid price if there is no Ask).

Once this initial price value is determined, the setting Listings → randomTickRange is used to determine the number of ticks by which to vary the price of random orders relative to the starting value. New resting Bid order prices and aggressive Sell order prices are randomly chosen as a number of ticks below the initial value, and new resting Ask order prices and new aggressive Buy order prices are randomly chosen as a number of ticks above the initial value. The frequency distribution of random prices across this range of prices is exponentially weighted by a factor of 1.05 ^ (randomTickRange - 1) so that more orders are placed near the top of the book than lower down.

The following diagram represents this distribution:



The final price of a new random order is chosen as a number of ticks away from the initial price value, weighted by the frequency distribution as described above. In this way, the final price is a multiple of the configured ticks on the listing (Listings → PriceTickSize), rounded to the configured precision of the listing (Listings → PricePrecision).

Distribution of Quantity

The quantity used on randomly generated orders is determined by the configuration of minimum and maximum values of either quantity or amount per each listing.

If the setting Listings → randomAmtMinimum is not set, then the value Listings → randomQtyMinimum is used as the lowest possible quantity value (or instead qtyMinimum if that is larger than randomQtyMinimum or randomQtyMinimum is not set). If the setting Listings → randomAmtMinimum is set, then this value divided by the determined random price (per the section above) is used as the lowest possible quantity value (or instead Listings → randomQtyMinimum if that is set and is larger than the minimum quantity determined from randomAmtMinimum, or again instead qtyMinimum if that is larger than the minimum quantity determined from randomAmtMinimum and randomQtyMinimum if it is set). This lowest possible quantity value is then rounded up to the nearest multiple of the setting Listings → qtyMultiple.

Likewise, if the setting Listings → randomAmtMaximum is not set, then the value Listings → randomQtyMaximum is used as the highest possible quantity value (or instead qtyMaximum if that is smaller than randomQtyMaximum or randomQtyMaximum is not set). If the setting Listings → randomAmtMaximum is set, then this value divided by the determined random price (per the section above) is used as the highest possible quantity value (or instead Listings → randomQtyMaximum if that is set and is smaller than the maximum quantity determined from randomAmtMaximum, or again instead qtyMaximum if that is smaller than the maximum quantity determined from randomAmtMaximum and randomQtyMaximum if it is set). This highest possible quantity value is then rounded down to the nearest multiple of the setting Listings → qtyMultiple.

If the settings Listings → randomAggQtyMinimum / randomAggQtyMaximum / randomAggAmtMinimum / randomAggAmtMaximum are set, then these values are used instead for aggressive orders, and the other corresponding settings described above are used only for passive orders.

Once the lowest and highest possible quantity values are determined, the quantity of random order actions is generated as an integer between these values, rounded to the nearest multiple of the setting Listings → qtyMultiple.

Special Market Events

Documentation TBC (controlling high volatility, spikes, market impact, etc.).

Data Source Playback

The market simulator can read data from an external data source and play it back through the main order book of the simulator to allow clients to consume this as tradable market data. For instance, a data source with bid/ask market data is turned into orders that are sent into the simulator's matching engine, which then results in published market data and the ability for other clients to send orders into the same matching engine's order book.

The market simulator currently supports either CSV file format or databases such as PostgreSQL and TimescaleDB. The simulator can read market data in the form of multiple bid/ask order book levels.

See this section for specific details of configuring data sources.

Format Specific Configuration

CSV Files

The connection string for a CSV file is the full absolute local file path to the file containing data. For example: "/path/to/my/file.csv"

Following usual CSV standards:

- Numeric values are expected to be specified with a decimal point for fractional amounts. e.g. 3 or 3.45
- Strings can be specified without double quotes (") if they have no spaces, otherwise if spaces are present the string should be surrounded by double quotes ("). e.g. abc or "a b c"
- If a string is surrounded by double quotes and it contains a double quote, it must be escaped by using 2 double quotes. e.g. "a b "" c" for the string a b " c
- Either a Unix or Windows style end of line can be used for each row.

An additional point is that currently all Date/Time strings are expected to be specified in the format: "YYYY-MM-DD HH:MM:SS.MMM"

The following data source properties can be configured to indicate aspects specific to the format of a CSV file:

- textDelemeter - delimiter used to separate values ("," by default if not specified)
- textHeaderRow - 1-based row index of where header row is located (0 indicates no header row)
- textDataRow - 1-based row index of where the first row of data is located. Cannot be 0, and must be greater than `textHeaderRow`

Specifically for CSV format files, and only when textHeaderRow is 0 or not specified, the column mapping properties of the data source can be used to map columns by index number. The columnTo property of the column mapping can be used to specify a 1-based column number to map to in the data source. In this case, if no column mappings are indicated, the following default order of columns is expected:

```
ReceivedTimeStamp, MessageTimeStamp, Instrument, BidParty,
BidQuantity, BidPrice, AskPrice, AskQuantity, AskParty
```

Also in this case, if some columns are mapped but not others, any unmapped column is assumed to be at the position as indicated above by the default order of columns.

The following examples show sample generic CSV formats using these data source properties:

textDelimiter absent / textHeaderRow=1 / textDataRow=2

```
col1,col2,col3,col4,col5,col6,col7
"2019-03-07 15:00:00.243",value1,"value 2","value "" 3",1,2,3
"2019-03-07 15:01:05.876",value4,"value 5","value "" 6",4,5,6
```

textDelimiter=; / textHeaderRow=2 / textDataRow=4

```
col1,col2,col3,col4,col5,col6,col7

"2019-03-07 15:00:00.243";value1;"value 2";"value "" 3";1;2;3
"2019-03-07 15:01:05.876";value4;"value 5";"value "" 6";4;5;6
```

textDelimiter absent / textHeaderRow=0 / textDataRow=1

```
"2019-03-07 15:00:00.243",value1,"value 2","value "" 3",1,2,3
"2019-03-07 15:01:05.876",value4,"value 5","value "" 6",4,5,6
```

Databases

The connection string for a database is the URL containing the IP, port, authentication details, and database name required to connect to the database instance. For example:
“postgresql://myuser:mypassword@10.0.1.1:5432/mydatabasename”

The following data source properties can be configured to indicate aspects specific to the format of a database:

- tableName - name of table containing data in the database

Basic Single Level Order Book

For a simple data source where each row contains only a single level bid/ask order book, the simulator starts reading the single level of depth from the first row of the datasource, and then proceeds to read each subsequent row after that. The trigger for playing back the content of a data source follows the same logic as the configuration used to indicate whether or not to generate random orders. The orderOnStartup property of a venue determines whether to play back any configured data sources when the simulator starts, and the genstart REST API can be used to start play back on demand. If the randomOrdersEnabled property of a listing is true, then both random order generation and playback from a configured data source will begin and continue to run simultaneously.

In processing each data source row, the simulator compares the current state of the matching engine for the given listing of each row, and generates a series of order actions (new orders, amends, cancels) to modify the current state of the matching engine to change to a state that matches the values from the current data source row being processed. The following logic is applied to derive order actions from each row of a data source:

- Any existing orders from counter parties that do not match the party of the data source row being processed are canceled.
- Any existing orders for the same party as the data source row being processed without a corresponding bid or ask entry in the data source are canceled.
- Any existing orders for the same party as the data source row being processed with a corresponding bid or ask entry in the data source are amended to the same price and quantity as the corresponding bid or ask entry in the data source.
- Any bid or ask entries in the data source without a corresponding existing order for the same party results in a new order with the properties of that bid or ask entry in the data source.

This will result in all of the usual market data updates generated when any order actions are sent into the matching engine. Each row of the datasource is played back into the matching engine at a rate that corresponds with the ReceivedTimeStamp property of the row. The first row will be played back immediately, the second row will be played back with a delay of the difference between the ReceivedTimeStamp of the second and first rows, and so on. This continues until the last row of the data source, at which point if the repeat property of the venue is false, then play back will stop, otherwise if it is true the simulator will continue to play back the content of the data source restarting from the first row.

The order actions derived from each data source row processed by the simulator use the bid and ask party, quantity, and price values to update the state of the matching engine. While ReceivedTimeStamp is used only to determine when the order actions are played back into the matching engine, the MessageTimeStamp is used to set the timestamp property (generally set by the sender) of each order action. The MessageTimeStamp property value used to set the timestamp on each order action is adjusted with the difference between the ReceivedTimeStamp of the first row and the current time of day at that moment.

For example, if the following 2 first rows from a data source were being played back at a current time of day of 14:30:00.500 :

```
ReceivedTimeStamp,MessageTimeStamp,...
"2019-03-07 15:00:00.243","2019-03-07 15:00:00.115",...
"2019-03-07 15:01:05.876","2019-03-07 15:01:05.203",...
```

...then the resulting order actions would be played back at the following times, with the following timestamps:

@ 14:30:00.500 : timestamp = 14:30:00.372

@ 14:31:06.133 : timestamp = 14:31:05.460

Note that the simulator will play back order actions with at least the same delay between values of ReceivedTimeStamp for each row, though the actual delay may be longer if the simulator is unable to play back the order actions for each row for too small of a specified delay.

For database format and CSV file format with a non-zero textHeaderRow value, the column mapping properties of the data source can be used to map columns by name. The columnFrom property of the column mapping is used to select the internal simulator field that will be mapped from, and the columnTo property indicates the data source column that the simulator field will be mapped to. In this case, any unmapped columns are assumed to use the same values as available for the columnFrom property as the column name in the data source.

See the section Market Simulator REST API | Column Mapping Sub List for descriptions of all possible fields that can be mapped from a data source as configured using the columnFrom property of a data source column mapping.

If any bid or ask level is missing a price or quantity value, that bid or ask entry is considered to be empty and removed from the state of the matching engine. If the party value for a bid or ask level is missing, a default value of “CP1” is used for data sources with a single level bid/ask order book.

The following shows a sample data source list of single level bid/ask order books, which can be interpreted either as a CSV or database format.

```
ReceivedTimeStamp,MessageTimeStamp,Instrument,BidParty,BidQuantity,
BidPrice,AskPrice,AskQuantity,AskParty
"2019-03-07 15:00:00.243","2019-03-07
15:00:00.115","VOD.L",CP1,10,133.50,134.85,15,CP2
"2019-03-07 15:01:05.876","2019-03-07
15:01:05.203","VOD.L",CP1,10,133.50,135.83,15,CP2
"2019-03-07 15:01:14.667","2019-03-07
15:01:13.998","VOD.L",CP1,8,133.50,135.70,18,CP2
```

Multiple Level Order Book

For a more complex data source where each row contains a bid/ask order book with more than one level, the simulator processes each row much like as described above for a single level order book, but changing the state of the matching engine to have the multiple bid/ask levels indicated by each row of the data source.

The column mapping properties of the data source can be used to map to multiple simulator fields corresponding to the columnFrom values of BidParty, BidQuantity, BidPrice, AskPrice, AskQuantity, AskParty. For each of these columnFrom values, a 1-based index can be appended to indicate which bid or ask level that value should be applied to, for example BidParty1 for the first bid level, BidParty2 for the second bid level, and so on. In this case, the columnTo property still works the same way, indicating either the name or (for a CSV file format when textHeaderRow is 0 or not specified) the index position of the data source column to map to. If no column mapping is indicated for one of the columnFrom values listed above, then either for a database format or CSV file format with a non-zero value for textHeaderRow, it is expected that column names are specified in the data source, and if that unmapped column name (from the list of BidParty, BidQuantity, BidPrice, AskPrice, AskQuantity, AskParty) is not found directly as one of the columns of the data source, then that same column name is expected to be present but with a 1-based index appended to indicate which bid or ask level that column of values should be applied to. For instance, if BidPrice is not specified as a mapped column of the data source, and BidPrice as a column name is not present in the data source, then it is expected that BidPrice1, BidPrice2, etc will be specified as columns in the data source for each level.

If the party value for a bid or ask level is missing for data sources with multiple bid/ask levels, a default value of “CP#” is used, where # is the index for each level (e.g. level 1 would use CP1, etc).

Note that the # character can also be appended to either the columnFrom or columnTo data source column mappings to indicate that all available depth levels can be mapped. For instance a column mapping of columnFrom=BidPrice# and columnTo=bidpx# indicates that a simulator field of BidPrice for level 1 of bids would be mapped to a column name in the data source of bidpx1, and a simulator field of BidPrice for level 2 of bids would be mapped to a column name in the data source of bidpx2, and so on.

It is also possible to limit the number of depth levels processed from a data source using the maxDepthLevels property of a venue. This value can optionally be used to indicate the maximum number of bid/ask levels that will be processed by the simulator in the data source, after which the remaining levels will be ignored and treated as empty.

The following shows a sample data source list of bid/ask order books with multiple (5) levels, some with empty entries, which can be interpreted either as a CSV or database format.

```
ReceivedTimeStamp,MessageTimeStamp,Instrument,BidParty1,BidQuantity
1,BidPrice1,BidParty2,BidQuantity2,BidPrice2,BidParty3,BidQuantity3
,BidPrice3,BidParty4,BidQuantity4,BidPrice4,BidParty5,BidQuantity5,
BidPrice5,AskPrice1,AskQuantity1,AskParty1,AskPrice2,AskQuantity2,A
```

```
skParty2,AskPrice3,AskQuantity3,AskParty3,AskPrice4,AskQuantity4,As
kParty4,AskPrice5,AskQuantity5,AskParty5
2021-04-08 12:34:00.003,2021-04-08
12:34:00.003,VOW,CP1,500000.0,1.18811,CP2,1000000.0,1.1881,CP3,3000
000.0,1.18807,CP4,5000000.0,1.18805,,,,1.18818,1500000.0,CP6,1.1882
3,3000000.0,CP7,1.18826,5000000.0,CP8,,,,,,
2021-04-08 12:34:10.044,2021-04-08
12:34:10.044,VOW,CP1,1000000.0,1.18812,,,,,,,,,,,,1.18819,1000000.
0,CP6,,,,,,,,,,,,
2021-04-08 12:34:20.048,2021-04-08
12:34:20.048,VOW,CP1,3000000.0,1.18812,CP2,1000000.0,1.18811,CP3,50
0000.0,1.1881,,,,,,,1.18815,2000000.0,CP6,1.18816,1000000.0,CP7,1.1
8817,1000000.0,CP8,,,,,,
```

Market Phases

The market simulator can be configured to operate in the context of specific market phases following a configured schedule. Each phase has distinct rules of operation regarding what type of trading is allowed and how orders are matched.

If any market phases are scheduled according to the startTime and endTime properties of phases configured on a venue, each of the scheduled market phases will become the active market phase between these start and end times during that day, interepreting these times according to the timeZone property configured on the venue. Any phase with the same end time as the start time of the next phase (including 00:00 and 24:00 which are considered as overlapping) will result in the immediate transition from the first phase to the next phase at this common time. If there is a gap in scheduled market phases, and no market phase is scheduled to be currently active, then the default active market phase is the Open phase. Any phase scheduled with a start time after its own end time is considered invalid and ignored.

For any scheduled market phases with overlapping start and end times, the phase scheduled to start within the start and end times of another phase will take priority as the active market phase, and then when it ends, revert to the initial market phase, unless that phase is also now ended. In general, if multiple phases have overlapping start and end times, each new phase with the most recent start time will be considered active at its start time, then at its end time the active phase should revert to the phase with the most recent start time that has not yet reached its end time.

Some examples of this include:

- If phase1 has a start time and end time between the start time and end time of phase2, then phase1 is active between phase2 being active (phase2 follows phase1 which then reverts back to phase2).
- If phase1 has a start time between the start time and end time of phase2, and phase1 has an end time greater than the end time of phase2, then phase1 would be active for the last part of phase2 and continue past it (phase2 follows phase1 which then becomes the Open phase if no other phase is scheduled).

See this section for specific details of configuring market phases on venues.

The following indicates details on the operation of each specific phase while it is active.

Open Phase

While the Open market phase is active, all normal trading activity is allowed.

Closed Phase

While the Closed market phase is active, all trading activity is blocked. It is still possible for market sessions to be connected and remain connected, and it is still possible to subscribe to market data updates and remain subscribed to market data updates, but all order action requests will be rejected, such as requests for new orders, amending orders, or canceling orders. In addition, at the moment a Closed phase starts, any existing orders in the matching engine for any listing will be terminated and rejected if they have a Time In Force of Day or Time In Force of GoodTillDate with an expiration of the current day.

Phase Halt

The halting of a phase is not technically a market phase itself, but rather the temporary halt of the currently active market phase. In this state, any order action request sent to the simulator will be rejected.

If a phase halt is explicitly configured as part of the phase schedule, it would only halt phases other than Closed. A Closed phase is always Closed and cannot be halted. A scheduled halt starting before a scheduled Closed phase would become a Closed phase, though a scheduled halt starting before or during a scheduled Closed phase that ends after the end of the Closed phase would then become a halt of the active phase following the Closed phase.

If a phase halt is triggered on demand through a halt request, it will halt the currently active market phase. However, if the request to halt a phase is made before a scheduled halt, the start of the scheduled halt would override the requested halt, such that the end of the scheduled halt would once again resume the currently active market phase.

For either a scheduled or requested halt of the currently active market phase, the property allowCancels in the scheduled halt or in the halt request indicates whether the halted phase would reject only new order and amend order requests, or also reject order cancel requests. In this way, if allowCancels is true, a halted phase may still process order cancel requests.

A currently halted phase can also be removed on demand through a resume request to reactive the otherwise currently active scheduled market phase.

Persisted State

The market simulator may store and recover:

- The order book;
- The last trade;

- Low and high trade prices.

The settings persistenceEnabled and persistenceFilePath are used to configure the feature - if persistenceEnabled is true:

- The market simulator recovers its state from the file placed at persistenceFilePath on startup or by request Market Simulator REST API | Recover Market State For Single Venue ;
- The market simulator stores its state on clean shutdown or by request Market Simulator REST API | Store Market State For Single Venue.

If the file's content is correct JSON and the instrument is found, the order book, last trade, and low and high trade prices will be removed during the recovery.

File Format

The data is stored in the JSON file.

For instance, the venue "LSE" contains two listings "AAPL" and "VOW":

- "AAPL" has no orders in the order book, no last trade, and low and high prices.
- "VOW" has the last trade, low and high prices. Its order book contains one buy order and two sell orders.

```
{
    "venue_id": "LSE",
    "instruments": [
        {
            "instrument": {
                "symbol": "AAPL",
                "price_currency": "USD",
                "base_currency": null,
                "security_exchange": "XLOM",
                "party_id": null,
                "cusip": null,
                "sedol": "B0YQ5W0",
                "isin": "US0378331005",
                "ric": null,
                "exchange_id": null,
                "bloomberg_id": null,
                "price_tick": 0.1,
                "quantity_tick": 1.0,
                "min_quantity": 1.0,
                "max_quantity": 10000.0,
                "party_role": null,
                "security_type": "CommonStock"
            },
            "last_trade": null,
            "info": null,
            "order_book": {
                "buy_orders": [],
                "sell_orders": []
            }
        },
        {
            "instrument": {
                "symbol": "VOW",
                "price_currency": "EUR",
                "base_currency": null,
                "security_exchange": "XLOM",
                "party_id": null,
                "cusip": null,
                "sedol": "0308908",
                "isin": "DE0007664005",
                "ric": null,
                "exchange_id": null,
                "bloomberg_id": null,
                "price_tick": 0.00001,
                "quantity_tick": 1.0,
                "min_quantity": 1.0,
                "max_quantity": 1000000000.0,
                "party_role": null,
                "security_type": "CommonStock"
            },
            "last_trade": {
                "buyer": "CP3",
                "seller": "CP6",
                "trade_price": 1.18815,
                "traded_quantity": 290.0,
                "aggressor_side": "Buy",
                "trade_time": "2025-05-16 14:42:49.958327",
                "market_phase": {
                    "trading_phase": "Open",
                    "trading_status": "Resume"
                }
            },
            "info": {
                "low_price": 1.18812,
                "high_price": 1.18818
            },
            "order_book": {
                "buy_orders": [
                    {
                        "client_instrument_descriptor": {
                            "security_id": "DE0007664005",
                            "symbol": "VOW",
                            "currency": "EUR",
                            "security_exchange": "XLOM",
                            "parties": [],
                            "requester_instrument_id": 1,
                            "security_type": "CommonStock",
                            "security_id_source": "ISIN"
                        },
                        "client_session": {
                            "type": "Generator",
                            "fix_session": null
                        },
                        "client_order_id": "SIM-
1747406520164373224",
                        "order_parties": [
                            {
                                "identifier": {
                                    "party_id": "CP1",
                                    "source": "Proprietary"
                                },
                                "role": "ExecutingFirm"
                            }
                        ],
                        "expire_time": null,
                        "expire_date": null,
                        "short_sale_exemption_reason": null,
                        "time_in_force": "Day",
                        "order_id": 250516144244000013,
                        "order_time": "2025-05-19 14:42:48.614534",
                        "side": "Buy",
                        "order_status": "Modified",
                        "order_price": 0.38799,
                        "total_quantity": 3000000.0,
                        "cum_executed_quantity": 1000535.0
                    }
                ],
                "sell_orders": [
                    {
                        "client_instrument_descriptor": {
                            "security_id": "DE0007664005",
                            "symbol": "VOW",
                            "currency": "EUR",
                            "security_exchange": "XLOM",
                            "parties": [],
                            "requester_instrument_id": 1,
                            "security_type": "CommonStock",
                            "security_id_source": "ISIN"
                        },
                        "client_session": {
                            "type": "Fix",
                            "fix_session": {
                                "begin_string": "FIXT.1.1",
                                "sender_comp_id": "SENDER",
                                "target_comp_id": "TARGET",
                                "client_sub_id": null
                            }
                        },
                        "client_order_id": "SIM-
    1747406520164373237",
                        "order_parties": [
                            {
                                "identifier": {
                                    "party_id": "CP5",
                                    "source": "Proprietary"
                                },
                                "role": "ExecutingFirm"
                            }
                        ],
                        "expire_time": null,
                        "expire_date": null,
                        "short_sale_exemption_reason": null,
                        "time_in_force": "Day",
                        "order_id": 250516144250000026,
                        "order_time": "2025-05-16 14:42:51.522018",
                        "side": "Sell",
                        "order_status": "Modified",
                        "order_price": 1.188,
                        "total_quantity": 408.0,
                        "cum_executed_quantity": 0.0
                    },
                    {
                        "client_instrument_descriptor": {
                            "security_id": "DE0007664005",
                            "symbol": "VOW",
                            "currency": "EUR",
                            "security_exchange": "XLOM",
                            "parties": [],
                            "requester_instrument_id": 1,
                            "security_type": "CommonStock",
                            "security_id_source": "ISIN"
                        },
                        "client_session": {
                            "type": "Generator",
                            "fix_session": null
                        },
                        "client_order_id": "SIM-
    1747406520164373236",
                        "order_parties": [
                            {
                                "identifier": {
                                    "party_id": "CP8",
                                    "source": "Proprietary"
                                },
                                "role": "ExecutingFirm"
                            }
                        ],
                        "expire_time": null,
                        "expire_date": null,
                        "short_sale_exemption_reason": null,
                        "time_in_force": "Day",
                        "order_id": 250516144250000025,
                        "order_time": "2025-05-16 14:42:50.627388",
                        "side": "Sell",
                        "order_status": "Modified",
                        "order_price": 1.18812,
                        "total_quantity": 606.0,
                        "cum_executed_quantity": 0.0
                    }
                ]
            }
        }
    ]
}
```

Data verification

On storing, the market simulator does the following verifications:

	Verification	Actions if fail
1	persistenceEnabled is true	Output log info message: "The market state was not stored: the persistence is disabled." Stop storing.

	Verification	Actions if fail
2	persistenceFilePath is not empty	Output log error message: "The market state was not stored: the persistence file path is empty." Stop storing.
3	The directory of the persistence file exists	Output log error message: "The market state was not stored: the persistence file path directory does not exist." Stop storing.
4	The file can be saved	Output log error message: "The market state was not stored: an error when unable to open file." Stop storing.

On recovery, the market simulator does the following verifications:

	Verification	Actions if fail
1	persistenceEnabled is true	Output log info message: "The market state was not recovered: the persistence is disabled." Stop the recovery

	Verification	Actions if fail
2	persistenceFilePath is not empty	Output log info message: "The market state was not recovered: the persistence file path is empty." Stop the recovery.
3	persistenceFilePath contains the path to the existing file	Output log info message: "The market state was not recovered: the persistence file path is unreachable." Stop the recovery.
4	The file was successfully opened	Output log error message: "The market state was not recovered: an error when unable to open file." Stop the recovery.
5	The file content was successfully parsed	Output log error message: “The market state was not recovered: the persistence file is malformed: {}", where “{}” is a placeholder for details. Stop the recovery.

	Verification	Actions if fail
6	The instrument is found and enabled. The equality of the following fields is used to check that the parsed instrument is present and enabled in the venue: • `symbol` • `price_currency` • `base_currency` • `security_exchange` • `party_id` • `cusip` • `sedol` • `isin` • `ric` • `exhange_id` • `bloomberg_id` • `party_role` • `security_type` If the instrument is found, all orders from its order book will be cancelled.	Output log warning message: "The instrument was not found, its recovery was ignored: {}", where “{}” is a placeholder for the instrument. Move on to the next instrument.

	Verification	Actions if fail
7	If the market phase is Closed, the order’s `time_in_force` is not “Day”.	Output log error message: “validation failed with 'the order is already expired because its time_in_force is Day and the market phase is Closed' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.
8	The order’s `client_instrument_descriptor` corresponds to the Instrument using the Instrument Resolution algorithm.	Output one of the log error messages: • "validation failed with 'client_instrument_descriptor is malformed' error, order was not recovered: {}" • "validation failed with 'client_instrument_descriptor does not match the instrument' error, order was not recovered: {}" where “{}” is a placeholder for the order. Move on to the following order.

	Verification	Actions if fail
9	If the order is in the “buy_orders” list, the order’s `side` is “Buy”.	Output log error message: “validation failed with 'invalid side value' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.
10	If the order is in the “sell_orders” list, the order’s `side` is “Sell”, “SellShort”, or “SellShortExempt”.	Output log error message: “validation failed with 'invalid side value' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.
11	The order’s `total_quantity` is greater than or equal to qtyMinimum.	Output log error message: “validation failed with 'total quantity minimal constraint violated' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.

	Verification	Actions if fail
12	The order’s `total_quantity` is less than or equal to qtyMaximum.	Output log error message: “validation failed with 'total quantity maximal constraint violated' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.
13	The order’s `total_quantity` is a multiple of qtyMultiple.	Output log error message: “validation failed with 'total quantity multiple constraint violated' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.
14	The order’s `cum_executed_quantity` is greater than or equal to 0.	Output log error message: “validation failed with 'cumulative executed quantity is less than zero' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.

	Verification	Actions if fail
15	The order's `cum_executed_quantity` is a multiple of qtyMultiple.	Output log error message: "validation failed with 'cumulative executed quantity multiple constraint violated' error, order was not recovered: {}", where "{}" is a placeholder for the order. Move on to the following order.
16	The order's `cum_executed_quantity` is less than total_quantity.	Output log error message: "validation failed with 'cumulative executed quantity is not less than total quantity' error, order was not recovered: {}", where "{}" is a placeholder for the order. Move on to the following order.
17	The order's `order_price` is a multiple of priceTickSize.	Output log error message: "validation failed with 'order price tick constraint violated' error, order was not recovered: {}", where "{}" is a placeholder for the order. Move on to the following order.

	Verification	**Actions if fail**
18	The order’s `order_status` is “New”, “PartiallyFilled”, or “Modified”.	Output log error message: “validation failed with 'unsupported order status value' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.
19	The order’s `time_in_force` is “Day”, “GoodTillDate”, or “GoodTillCancel”.	Output log error message: “validation failed with 'time in force value is invalid' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.
20	If the order’s `time_in_force` is “Day”, “order_time” is not the prior date.	Output log error message: “validation failed with 'order already expired' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.

	Verification	Actions if fail
21	If the order’s `time_in_force` is “GoodTillDate”, `expire_time` or `expire_date` is not null.	Output log error message: “validation failed with 'neither expire date nor expire time specified' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.
22	If the order’s `time_in_force` is “GoodTillDate”, the order is not expired.	Output log error message: “validation failed with 'order already expired' error, order was not recovered: {}”, where “{}” is a placeholder for the order. Move on to the following order.
23	The `last_trade` is not `null.`	Output log warning message: ”last trade is empty, nothing to validate” Remove the information about the last trade from the instrument. Then move on to low and high trade prices.

	Verification	Actions if fail
24	The `last_trade` 's `trade_price` is a multiple of priceTickSize.	Output log error message: “validation failed with 'trade price tick constraint violated' error, last trade was not recovered: {}”, where “{}” is a placeholder for the `last_trade` . Move on to low and high trade prices.
25	The `last_trade` 's `traded_quantity` is a multiple of qtyMultiple.	Output log error message: “validation failed with 'traded quantity multiple constraint violated' error, last trade was not recovered: {}”, where “{}” is a placeholder for the `last_trade` . Move on to low and high trade prices.
26	The `last_trade` 's `traded_quantity` is greater than or equal to qtyMinimum.	Output log error message: “validation failed with 'minimal traded quantity constraint violated' error, last trade was not recovered: {}”, where “{}” is a placeholder for the `last_trade` . Move on to low and high trade prices.

	Verification	Actions if fail
27	The `last_trade` ’s `traded_quantity` is less than or equal to qtyMaximum.	Output log error message: “validation failed with 'maximal traded quantity constraint violated' error, last trade was not recovered: {}”, where “{}” is a placeholder for the `last_trade` . Move on to low and high trade prices.
28	The `info` is not `null` .	Output log warning message: ”info is empty, nothing to validate” Remove the information about the low and high prices from the instrument. Move on to the following instrument.
29	The `info` ’s `low_price` is a multiple of priceTickSize.	Output log error message: “validation failed with 'low price tick constraint violated' error, instrument info was not recovered: {}”, where “{}” is a placeholder for the `info` . Move on to the following instrument.

	Verification	Actions if fail
30	The `info` 's `high_price` is a multiple of priceTickSize.	Output log error message: “validation failed with 'high price tick constraint violated' error, instrument info was not recovered: {}”, where “{}” is a placeholder for the `info` . Move on to the following instrument.
31	The `info` 's `low_price` is less than or equal to `info` 's `high_price` .	Output log error message: “validation failed with 'low price is less than or equal to high price constraint violated' error, instrument info was not recovered: {}”, where “{}” is a placeholder for the `info` . Move on to the following instrument.

Administrative Settings

General Settings

Manage general settings across all market simulator instances.

Path	Type	Description
key	Text	Unique primary key
value	Text	Value of setting

A list of settings, currently defined in the Market Simulator backend:

Setting	Key	Meaning
System Display Name	DisplayName	The name of the site for display purposes
Price Seed Values Database Connection	SeedPriceDatabaseConnection	Connection string to a PostgreSQL database to synchronize price seeds
Price Seed Values Last Update	SeedPricesLastUpdated	Last time when price seeds were synchronised

Market Simulator REST backend allows a user to retrieve general settings and update them. There is no possibility to create a new setting or drop an existing one via the REST API.

Get Settings

Request all market simulator general settings.

Resource URI

```
GET /api/settings
```

Example: Request all settings

```
GET /api/settings HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible response

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: XXX

{
    "settings" : [
        {
            "key":"DisplayName",
            "value":"SITE NAME"
        },
        {
            "key" : "SeedPriceDatabaseConnection",
            "value" :
"postgresql://user:password@1.2.3.4:1234/simdb"
        },
        {
            "key" : "SeedPricesLastUpdated",
            "value" : "2023-09-07 12:34:55"
        }
    ]
}
```

Update Settings

Update Market Simulator general setting values.

Resource URI

```
PUT /api/settings
```

Example: Update settings

```
PUT /api/settings/DisplayName HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost

{
    "settings" : [
      {
        "key":"DisplayName",
        "value":"MKTSIMULATOR-DEV"
      },
      {
        "key" : "SeedPriceDatabaseConnection",
        "value" : "postgresql://user:pass@1.2.3.4:1234/simdb"
      }
    ]
}
```

Example: Positive reply

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: XXX

{
    "result" : "General settings updated successfully"
}
```

Example: Negative reply

```
HTTP/1.1 400 Bad Request
Connection: Close
Content-Length: XXX

{
    "result" : "Unknown BadKey setting update requested"
}
```

SeedPricesLastUpdated is considered as read-only to a user. It can be updated by the MarketSimulator BE itself only, and will be ignored if included in the settings update request.

Venues

Manage settings for each market simulator venue entity.

Resource Properties

Path	Type	Description
id	Text	Unique primary key
name	Text	Display name
engineType	Enum	Indicates which type of engine we want to use for this venue (Currently only matching engine is available) • **Matching** - Matching engine • **Quoting** - Quote engine
supportTifIoc	Boolean	Whether IOC limit orders are supported
supportTifFok	Boolean	Whether FOK limit orders are supported
supportTifDay	Boolean	Whether Day limit orders are supported
includeOwnOrders	Boolean	Whether to include a party’s own orders in published depth
restPort	Integer	The port that will be used to send REST API calls
orderOnStartup	Boolean	Whether to start generating orders when MktSimulator launched
randomPartyCount	Integer	The amount of different counter party id’s used by generator

Path	Type	Description
timeAndSalesEnabled	Boolean	Whether time and sales trades should be included at all in market data updates
timeAndSalesQuantityEnabled	Boolean	Whether time and sales trades should include quantity of the trade
timeAndSalesSideEnabled	Boolean	Whether time and sales trades should include side of the trade
timeAndSalesPartiesEnabled	Boolean	Whether time and sales trades should include counter parties of the trade
timeZone	Text	Local time zone for a venue (f.e. Europe/Kyiv or America/Los_Angeles) For reference: https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

Path	Type	Description
cancelOnDisconnect	Boolean	If a client's FIX connection disconnects: • False (default) - any of the live resting orders placed through that COMPID should be left in the Order Book. • True - any of the live resting orders that were placed through that COMPID should be cancelled, the rejection messages the next time that COMPID reconnects should be sent.
persistenceEnabled	Boolean	Whether a matching engine persisted state functionality should be enabled
persistenceFilePath	Text	A file path to the persistence file where matching engine state should be stored/recovered

Market Phases Sub-List

Path	**Type**	**Description**
phase	Text	Enum value from a pre-defined list of supported values: • Open • Closed • PreOpen • PreClose • Auction • TradeAtLast
startTime	Text	The time a phase should begin set in corresponding venue timezone, specified to the granularity of minutes
endTime	Text	The time a phase should end set in corresponding venue timezone, specified to the granularity of minutes

Path	Type	Description
endTimeRange	Integer	Time range to choose a random actual end time from the specified endTime for the phase (this setting is only used during an Auction phase). The value indicates the range after and before that a random end time will be chosen. For example, a value of 5 indicates that a random end time will be chosen between 5 minutes before and 5 minutes after the specified endTime.
allowCancels	Boolean	Whether or not to allow cancel of orders (this setting is only used during a Halt phase).

Get Single Venue

Request a single market simulator venue entity.

Resource URI

```
GET /api/venues/{venueId}
```

Example: Request single venue

```
GET /api/venues/LSE HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Positive reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "id" : "LSE",
    "name" : "London Stock Exchange",
    "engineType" : "Matching",
    "supportTifIoc" : true,
    "supportTifFok" : true,
    "supportTifDay" : true,
    "includeOwnOrders" : true,
    "restPort":9184,
    "orderOnStartup" : false,
    "randomPartyCount" : 10,
    "timeAndSalesEnabled" : true,
    "timeAndSalesQuantityEnabled" : true,
    "timeAndSalesSideEnabled" : true,
    "timeAndSalesPartiesEnabled" : true,
    "timezone" : "America/Los_Angeles",
    "cancelOnDisconnect" : false,
    "persistenceEnabled" : true,
    "persistenceFilePath" : "/path/to/LSE-state.json"
    "phases" : [
        {
             "phase" : "Open",
             "startTime" : "07:39:00",
             "endTime" : "07:39:00",
             "endTimeRange" : 0
        },
        {
             "phase" : "Closed",
             "startTime" : "09:27:00",
             "endTime" : "09:28:00",
             "endTimeRange" : 0
        }
    ]
}
```

Example: Negative reply

```
HTTP/1.1 404 Not Found
Connection: Close
Content-Length: XXX

{
    "result" : "No such venue"
}
```

Get Multiple Venues

Request all market simulator venue entities.

Resource URI

```
GET /api/venues
```

Example: Request all venues

```
GET /api/venues HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible response

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: XXX

{
    "venues": [
        {
            "id": "FASTMATCH",
            "name": "FASTMATCH",
            "engineType": "Quoting",
            "supportTifIoc": true,
            "supportTifFok": true,
            "supportTifDay": true,
            "includeOwnOrders": true,
            "restPort": 9182,
            "orderOnStartup": false,
            "randomPartyCount": 10,
            "timeAndSalesEnabled": true,
            "timeAndSalesQuantityEnabled": true,
            "timeAndSalesSideEnabled": true,
            "timeAndSalesPartiesEnabled": true,
            "timezone" : "America/Los_Angeles",
            "cancelOnDisconnect": false,
            "persistenceEnabled" : false,
            "phases": []
        },
        {
            "id" : "LSE",
            "name" : "London Stock Exchange",
            "engineType" : 1,
            "supportTifIoc" : true,
            "supportTifFok" : true,
            "supportTifDay" : true,
            "includeOwnOrders" : true,
            "restPort":9184,
            "orderOnStartup" : false,
            "randomPartyCount" : 10,
            "timeAndSalesEnabled" : true,
            "timeAndSalesQuantityEnabled" : true,
            "timeAndSalesSideEnabled" : true,
            "timeAndSalesPartiesEnabled" : true,
            "timezone" : "America/Los_Angeles",
            "cancelOnDisconnect" : false,
            "persistenceEnabled" : true,
            "persistenceFilePath" : "/path/to/LSE-state.json"
            "phases" : [
                {
                   "phase" : "Open",
                   "startTime" : "07:39:00",
                   "endTime" : "07:39:00",
                   "endTimeRange" : 0
                },
                {
                   "phase" : "Closed",
                   "startTime" : "09:27:00",
                   "endTime" : "09:28:00",
                   "endTimeRange" : 0
                }
            ]
        }
    ]
}
```

Add Venue

Add new market simulator venue entity.

Resource URI

```
POST /api/venues
```

Example: Add venue

```
POST /api/venues HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
Content-Length: XXX

{
    "id" : "NewExchange",
    "name" : "NewExchange Description",
    "engineType" : "Matching",
    "supportTimeInSales" : false,
    "supportTifIoc" : false,
    "supportTifFok" : false,
    "supportTifDay" : false,
    "includeOwnOrders" : false,
    "restPort" : 9087,
    "orderOnStartup" : true,
    "randomPartyCount" : 1
}
```

Example: Positive reply

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Length: XXX

{
    "result" : "Requested insert of the venue - NewExchange"
}
```

Example: Negative reply

```
HTTP/1.1 400 Bad Request
Connection: Close
Content-Length: XXX

{
    "result" : "There is already such venue"
}
```

Update Venue

Update existing market simulator venue entities.

Resource URI

```
PUT /api/venues/{venueId}
```

Example: Update Venue

```
POST /api/venues/NewExchange HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
Content-Length: XXX

{
    "randomOrdersRate" : 12,
    "orderOnStartup" : true
}
```

Example: Positive reply

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: XXX

{
    "result" : "Requested update of the venue - NewExchange"
}
```

Example: Negative reply

```
HTTP/1.1 404 Not Found
Connection: Close
Content-Length: XXX

{
    "result":"No such venue"
}
```

Delete Single Venue

It is not allowed to delete venues using the REST API.

Listings

Manage settings for specific listings on each market simulator instance.

Resource Properties

Path	Type	Description
id	Numeric	Unique primary key
symbol	Text	The symbol for this listing, unique to the venue
venueId	Text	Unique venue key for this listing

Path	Type	Description
securityType	Text	Security type for this listing, possible values are: • CS - Common Stock • FUT - Future • OPT - Option • MLEG - Multi-Leg Instrument • SML - Synthetic Multi-Leg Instrument • WAR - Warrant • MF - Mutual Fund • CORP - Corporate Bond • CB - Convertible Bond • REPO - Repurchase Agreement • INDEX - Index • CFD - Contract for Difference • CD - Certificate • FXSPOT - Forex Spot • FORWARD - Forward • FXFWD - Forex Forward • FXNDF - Forex Non-Deliverable Forward • FXSWAP - Forex Swap • FXNDS - Forex Non-Deliverable Swap
priceCurrency	Text	Currency of price (for FX, second currency in the pair)
fxBaseCurrency	Text	For FX only, first currency in the pair, representing current of quantity

Path	Type	Description
instrSymbol	Text	A common symbol used for this listing across different venues
securityExchange	Text	Exchange name to indicate where a listing is traded
partyId	Text	Additional value to indicate where a listing is traded
partyRole	Text	Additional value to indicate where a listing is traded
cusipId	Text	CUSIP instrument/listing identifier
sedolId	Text	SEDOL instrument/listing identifier
isinId	Text	ISIN instrument/listing identifier
ricId	Text	RIC instrument/listing identifier
exchangeSymbolId	Text	Custom Exchange listing identifier
bloombergSymbolId	Text	Bloomberg instrument/listing identifier
qtyMinimum	Decimal	The minimum allowed order quantity (0 by default, though all orders require >0 quantity)
qtyMaximum	Decimal	The maximum allowed order quantity (no limit by default)
qtyMultiple	Decimal	Required even multiple of order quantity (any multiple by default)
priceTickSize	Decimal	Required even multiple of order price (any multiple by default)

Path	Type	Description
randomQtyMinimum	Decimal	The minimum quantity that can be created by the Random Order Generator (uses qtyMinimum by default or if qtyMinimum has a higher value) Used only for passive orders if randomAggQtyMinimum and/or randomAggAmtMinimum have a value.
randomQtyMaximum	Decimal	The maximum quantity that can be created by the Random Order Generator (uses qtyMaximum by default or if qtyMaximum has a lower value) Used only for passive orders if randomAggQtyMaximum and/or randomAggAmtMaximum have a value.
randomAmtMinimum	Decimal	The amount used to derive the minimum quantity that can be created by the Random Order Generator (uses randomQtyMinimum by default or if randomQtyMinimum has a higher value than the quantity derived from this value) Used only for passive orders if randomAggQtyMinimum and/or randomAggAmtMinimum have a value.

Path	Type	Description
randomAmtMaximum	Decimal	The amount used to derive the maximum quantity that can be created by the Random Order Generator (uses randomQtyMaximum by default or if randomQtyMaximum has a lower value than the quantity derived from this value) Used only for passive orders if randomAggQtyMaximum and/or randomAggAmtMaximum have a value.
randomAggQtyMinimum	Decimal	The minimum quantity that can be created by the Random Order Generator for aggressive orders (uses randomQtyMinimum by default)
randomAggQtyMaximum	Decimal	The maximum quantity that can be created by the Random Order Generator for aggressive orders (uses randomQtyMaximum by default)
randomAggAmtMinimum	Decimal	The amount used to derive the minimum quantity that can be created by the Random Order Generator for aggressive orders (uses randomAggQtyMinimum by default or if randomAggQtyMinimum has a higher value than the quantity derived from this value)
randomAggAmtMaximum	Decimal	The amount used to derive the maximum quantity that can be created by the Random Order Generator for aggressive orders (uses randomAggQtyMaximum by default or if randomAggQtyMaximum has a lower value than the quantity derived from this value)
randomDepthLevels	Integer	Maximum count of depth levels that can be created by Random Order Generator (no limit by default)
randomOrdersSpread	Decimal	The smallest top of book bid/ask price difference for prices created by the Random Order Generator (uses priceTickSize by default)

Path	Type	Description
randomOrdersRate	Integer	The number of order actions (new order/modification/cancel/wait) per second during Random Order Generation
randomTickRange	Integer	Range of price ticks used to calculate prices created by the Random Order Generator (10 by default)
randomOrdersEnabled	Boolean	Indicates if the Random Order Generator has to generate random orders on a listing
enabled	Boolean	Indicates if the listing is enabled and should be used by the Market Simulator

Get Single Listing

Request a single market simulator listing entity via `id`.

Resource URI

```
GET /api/listings/{id}
```

Request a single market simulator listing entity via `symbol`.

Resource URI

```
GET /api/listings/{symbol}
```

Example: Request by symbol

```
GET /api/listings/DE0007664005 HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Positive reply

```
HTTP/1.1 200 OK
```

```
Connection: Close
Content-Length: XXX

{
    "id" : 1,
    "symbol" : "DE0007664005",
    "venueId" : "XETRA",
    "securityType" : "CS",
    "priceCurrency" : "EUR",
    "fxBaseCurrency" : "",
    "instrSymbol" : "DE0007664005",
    "securityExchange" : "XETR",
    "partyId" : "",
    "partyRole" : "",
    "cusipId" : "",
    "sedolId" : "",
    "isinId" : "DE0007664005",
    "ricId" : "",
    "exchangeSymbolId" : "",
    "bloombergSymbolId" : "",
    "qtyMinimum" : 1.0,
    "qtyMaximum" : 100000.0,
    "qtyMultiple" : 1.0,
    "priceTickSize" : 0.001,
    "enabled" : true,
    "randomQtyMinimum" : 1.0,
    "randomQtyMaximum" : 700.0,
    "randomAmtMinimum" : 0.0,
    "randomAmtMaximum" : 0.0,
    "randomDepthLevels" : 7,
    "randomOrdersSpread" : 0.01,
    "randomOrdersRate" : 7,
    "randomTickRange" : 100,
    "randomOrdersEnabled" : true,
    "randomAggQtyMinimum" : 2.0,
    "randomAggQtyMaximum" : 200.0,
    "randomAggAmtMinimum" : 1000.0,
    "randomAggAmtMaximum" : 3000.0
}
```

Example: Negative reply

```
HTTP/1.1 404 Not Found
Connection: Close
Content-Length: XXX

{
    "result" : "No such listing"
}
```

Market Simulator selects a listing by both symbol and VenueID implicitly when a GET request is received with a symbol. The VenueID value is taken from the simulator instance configuration.

Get Multiple Listings

Request all market simulator listings entities.

Resource URI

```
GET /api/listings/
```

Example: Request multiple listings

```
GET /api/listings HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "listings": [
        {
            "id": 1,
            "symbol": "DE0007664005",
            "venueId": "XETRA",
            "securityType": "CS",
            "priceCurrency": "EUR",
            "fxBaseCurrency": "",
            "instrSymbol": "DE0007664005",
            "securityExchange": "XETR",
            "partyId": "",
            "partyRole": "",
            "cusipId": "",
            "sedolId": "",
            "isinId": "DE0007664005",
            "ricId": "",
            "exchangeSymbolId": "",
            "bloombergSymbolId": "",
            "qtyMinimum": 1,
            "qtyMaximum": 1000000,
            "qtyMultiple": 1,
            "priceTickSize": 0.001,
            "randomQtyMinimum": 1,
            "randomQtyMaximum": 1000,
            "randomAmtMinimum": 0,
            "randomAmtMaximum": 0,
            "randomDepthLevels": 10,
            "randomOrdersSpread": 1,
            "randomOrdersRate": 1,
            "randomTickRange": 10,
            "randomAggQtyMinimum" : 2.0,
            "randomAggQtyMaximum" : 200.0,
            "randomAggAmtMinimum" : 1000.0,
            "randomAggAmtMaximum" : 3000.0
        },
        {
            "id": 3,
            "symbol": "EUR/USD",
            "venueId": "FASTMATCH",
            "securityType": "FXSPOT",
            "priceCurrency": "USD",
            "fxBaseCurrency": "EUR",
            "instrSymbol": "EUR/USD",
            "securityExchange": "FASTMATCH",
            "partyId": "",
            "partyRole": "",
            "cusipId": "",
            "sedolId": "",
            "isinId": "",
            "ricId": "",
            "exchangeSymbolId": "",
            "bloombergSymbolId": "",
            "qtyMinimum": 1,
            "qtyMaximum": 1.0E8,
            "qtyMultiple": 1,
            "priceTickSize": 0.001,
            "randomQtyMinimum": 1,
            "randomQtyMaximum": 1.0E7,
            "randomAmtMinimum": 0,
            "randomAmtMaximum": 0,
            "randomDepthLevels": 3,
            "randomOrdersSpread": 1.0E-4,
            "randomOrdersRate": 1,
            "randomTickRange": 10,
            "randomAggQtyMinimum" : 2.0,
            "randomAggQtyMaximum" : 200.0,
            "randomAggAmtMinimum" : 1000.0,
            "randomAggAmtMaximum" : 3000.0
        }
    ]
}
```

Add Listing

Add a new market simulator listing entity.

Resource URI

```
POST /api/listings/
```

Example: Request to add a new listing

```
POST /api/listings HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost

{
    "symbol" : "AAPL",
    "venueId" : "XETRA",
    "instrSymbol" : "AAPL",
    "securityType" : "CS",
    "qtyMinimum" : 1,
    "qtyMaximum" : 100000,
    "randomQtyMultiple" : 1,
    "randomQtyMinimum" : 200,
    "randomQtyMaximum" : 1100,
    "randomAmtMinimum" : 100,
    "randomAmtMaximum" : 2000,
    "randomOrdersEnabled" : false,
    "enabled" : false
}
```

Example: Positive reply

```
HTTP/1.1 201 Created
Connection: Close
Content-Length: XXX

{
    "result" : "Requested insert of the listing - AAPL"
}
```

Example: Negative reply

```
HTTP/1.1 400 Bad Request
Connection: Close
Content-Length: XXX

{
    "result" : "There is already such listing"
}
```

Update Listing

Update existing market simulator listing entity via ID.

Resource URI

```
PUT /api/listings/{listingId}
```

Example: Request to update a listing

```
PUT /api/listings/AAPL HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost

{
    "enabled" : true,
    "randomOrdersEnabled" : true
}
```

Example: Positive reply

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: XXX

{
    "result" : "Requested update of the listing - AAPL"
}
```

Example: Negative reply

```
HTTP/1.1 404 Not Found
```

```
Connection: Close
Content-Length: XXX

{
    "result" : "No such listing"
}
```

Update existing market simulator listing entities via Symbol.

Resource URI

```
PUT /api/listings/{symbol}
```

Example: Request to update a new listing

```
PUT /api/listings/AAPL HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost

{
    "priceTickSize" : 0.001,
    "qtyMultiple" : 2
}
```

Example: Positive reply

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: XXX

{
    "result" : "Requested update of the listing - AAPL"
}
```

Example: Negative reply

```
HTTP/1.1 404 Not Found
Connection: Close
Content-Length: XXX

{
    "result" : "No such listing"
}
```

⚠ Market Simulator updates a listing by both symbol and VenueID implicitly when a PUT request is received with a symbol. The VenueID value is taken from the simulator instance configuration.

Data Sources

Manage settings for specific data sources on each market simulator instance.

See this section for more information about playback of data from data sources.

Resource Properties

Path	Type	Description
id	Numeric	Unique primary key
enabled	Boolean	Whether or not this data source is enabled or disabled
name	Text	Name for this data source, unique to the venue
venueId	Text	Unique venue key for a data source
connection	Text	Connection string for the datasource (eg a file path for CSV files, or a database connection string for database connections).

Path	Type	Description
format	Text	Indicates the type of historic data storage. Supported values: • CSV - comma-separated value file • PSQL - PostgreSQL (or TimescaleDB) database
type	Text	Indicates the data source format. Supported values: • OrderBook - L1/L2 market data format
repeat	Boolean	Whether to start reading from the beginning of the data source when the end is reached (supported only for files and databases)
textDelimeter	Char	Delimiter used to separate values (",") by default if not specified) (supported only for CSV format)
textHeaderRow	Integer	1-based row index of where header row is located (0 indicates no header row) (supported only for CSV format)

Path	Type	Description
textDataRow	Integer	1-based row index of where the first row of data is located. Cannot be 0, and must be greater than `textHeaderRow`, if present (supported only for CSV format)
tableName	Text	Name of table (supported only for databases)
maxDepthLevels	Integer	Maximum depth levels to read from the data source. 0 or unspecified indicates to read all levels.

Column Mapping Sub-List

Path	Type	Description
dataSourceId	Integer	An identifier of the data source record which has a column mapping config. Users may omit to send this value in the requests.

Path	Type	Description
columnFrom	Enum	Internal simulator field for mapping data. Supported values (part 1, continued on next page): • ReceivedTimeStamp - timestamp market data update was received (required, date/time precision to the ms, e.g. `2019-03-07 15:00:00.243`) • MessageTimeStamp - timestamp from original sent market data message (required, date/time precision to the ms, e.g. `2019-03-07 15:00:00.115`) • Instrument - instrument symbol matching one of the listing’s Symbol for this venue (required, string) • BidParty - bid level counter party (optional, string, e.g. `CP1`) • BidQuantity - bid level quantity (optional, float, e.g. `10`) • BidPrice - bid level price (optional, float, e.g. `133.5`) • AskPrice - ask level price (optional, float, e.g. `134.85`) • AskQuantity - ask level quantity (optional, float, e.g. `15`) • AskParty - ask level counter party (optional, string, e.g. `CP2`)

Path	Type	Description
columnFrom	Enum	Internal simulator field for mapping data. Supported values (part 2, continued from previous page): Any bid or ask level missing a price or quantity value is considered to be empty and removed from the state of the matching engine. If the party value for a bid or ask level is missing, a default value of "CP#" is used, where # is the index for each level (e.g. level 1 would use CP1, etc). If the datasource contains several levels, a 1-based level index must be appended to each of column names BidParty, BidQuantity, BidPrice, AskPrice, AskQuantity, AskParty. For example, BidParty1, BidParty2 etc. The # character can also be used with these fields to indicate reading from whatever index levels are specified in the file or database itself. For example, BidParty#, BidQuantity# etc.
columnTo	Text	External data source column name for mapping data. If the datasource contains several levels, the # character can be used as a placeholder for the level number. For instance, `bidparty#` instead of `bidparty1`, `bidparty2`, etc. In the case of CSV format, a 1-based column number can be indicated to specify the position of columns instead of the name of columns. To use this method, `textHeaderRow` for the datasource must be 0.

Get Single Data Source

Request a single market simulator data source entity.

Resource URI

```
GET /api/datasources/{id}
```

Example: Request single data source

```
GET /api/datasource/1 HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Positive reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "id" : 1,
    "enabled" : false,
    "name" : "LSE1",
    "venueId" : "LSE",
    "connection" : "lse1.csv",
    "format" : "CSV",
    "type" : "OrderBook",
    "repeat" : false,
    "textDelimeter" : "|",
    "textHeaderRow" : 0,
    "textDataRow" : 1,
    "maxDepthLevels": 3,
    "columnMapping": [
        {
            "dataSourceId" : 1,
            "columnFrom" : "ReceivedTimeStamp",
            "columnTo" : "ActionTime"
        },
        {
            "dataSourceId" : 1,
            "columnFrom" : "MessageTimeStamp",
            "columnTo" : "2"
        },
        {
            "dataSourceId" : 1,
            "columnFrom" : "BidPrice3",
            "columnTo" : "bid_price_three"
        }
    ]
}
```

Example: Negative reply

```
HTTP/1.1 404 Not Found
Connection: Close
Content-Length: XXX

{
    "result" : "No such data source"
}
```

Get Multiple Data Sources

Request all market simulator data sources entities.

Resource URI

```
GET /api/datasources
```

Example: Get all data sources

```
GET /api/datasources HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible reply

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: XXX

{
    "dataSources": [
        {
            "id" : 1,
            "enabled" : true,
            "name" : "XETRA-3",
            "venueId" : "XETRA",
            "connection" :
"postgresql://develop:develop@172.16.238.3:5432/simhistoricaldb",
            "format" : "PSQL",
            "type" : "OrderBook",
            "repeat" : true,
            "tableName" : "historical_data",
            "columnMapping": [
                {
                    "dataSourceId" : 1,
                    "columnFrom" : "ReceivedTimeStamp",
                    "columnTo" : "ActionTime"
                },
                {
                    "dataSourceId" : 1,
                    "columnFrom" : "MessageTimeStamp",
                    "columnTo" : "2"
                },
            ]
        },
        {
            "id" : 2,
            "enabled" : false,
            "name" : "XETRA-CONT-1",
            "venueId" : "XETRA",
            "connection" : "/rodata/XETRA-trimmed.csv",
            "format" : "CSV",
            "type" : "OrderBook",
            "repeat" : true,
            "textDelimeter" : ";",
            "textHeaderRow" : 1,
            "textDataRow" : 2,
            "columnMapping" : []
        }
    ]
}
```

Add Data Sources

Add a new market simulator data source entity.

Resource URI

```
POST /api/datasources
```

Example: Request to add a new data source entity

```
POST /api/datasources HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost

{
    "enabled" : false,
    "name" : "XETRA-NEW",
    "venueId" : "XETRA",
    "connection" : "/rodata/XETRA-new.csv",
    "format" : "CSV",
    "type" : "OrderBook",
    "repeat" : true,
    "textDelimiter" : ",",
    "textHeaderRow" : 0,
    "textDataRow" : 100,
    "columnMapping": [
        {
            "columnFrom" : "ReceivedTimeStamp",
            "columnTo" : "ActionTime"
        }
    ]
}
```

Example: Positive reply

```
HTTP/1.1 201 Created
Connection: Close
Content-Length: XXX

{
    "result" : "Requested insert of the data source - XETRA-NEW"
}
```

Example: Negative reply

```
HTTP/1.1 400 Bad Request
Connection: Close
Content-Length: XXX

{
    "result" : "There is already such data source"
}
```

Update Data Sources

Update existing market simulator data source entities.

Resource URI

```
PUT /api/datasources/{id}
```

Example: Request to update a data source entity

```
PUT /api/datasources/1 HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost

{
    "enabled" : true,
    "venueId" : "XETRA",
    "repeat" : false
}
```

Example: Positive reply

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: XXX

{
    "result" : "Requested update of the data source with identifier
- 1"
}
```

Example: Negative reply

```
HTTP/1.1 404 Not Found
Connection: Close
Content-Length: XXX

{
    "result" : "No such data source"
}
```

Price Seeds

Manage values for price seeds on which will be used to generate random orders on market simulator.

Resource Properties

Path	Type	Description
id	Integer	A unique price seed record identifier
symbol	Text	Symbol for this instrument

Path	Type	Description
securityType	Text	Security type for a target instrument: • CS - Common Stock • FUT - Future • OPT - Option • MLEG - Multi-Leg Instrument • SML - Synthetic Multi-Leg Instrument • WAR - Warrant • MF - Mutual Fund • CORP - Corporate Bond • CB - Convertible Bond • REPO - Repurchase Agreement • INDEX - Index • CFD - Contract for Difference • CD - Certificate • FXSPOT - Forex Spot • FORWARD - Forward • FXFWD - Forex Forward • FXNDF - Forex Non-Deliverable Forward • FXSWAP - Forex Swap • FXNDS - Forex Non-Deliverable Swap

Path	Type	Description
priceCurrency	Text	Currency of price (for FX, second currency in the pair), using ISO-4217 3-char currency codes
securityId	Text	Security ID for this listing
securityIdSource	Text	Type of security ID entered, can be null (i.e. for FX where we don't require a security ID)
instrumentSymbol	Text	Common symbol used for this listing across different venues
midPrice	Decimal	Last known mid price
bidPrice	Decimal	Last known bid price
offerPrice	Decimal	Last known offer price
lastUpdate	DateTime	Date and time in UTC of last time instrument price was last successfully refreshed

Get Single Price Seed

Request a single market simulator price seed entity.

Resource URI

```
GET /api/priceseeds/{id}
```

Example: Get a single price seed

```
GET /api/priceseeds/1 HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Positive reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "id" : 1,
    "symbol" : "PAR_ST",
    "securityType" : "CS",
    "priceCurrency" : "USD",
    "securityId" : "SEDOL",
    "securityIdSource" : "SEDOL",
    "instrumentSymbol" : "PAR",
    "midPrice" : 10,
    "bidPrice" : 9,
    "offerPrice" : 11,
    "lastUpdate" : "2015-09-08T15:32:09"
}
```

Example: Negative reply

```
HTTP/1.1 404 Not Found
Connection: Close
Content-Length: XXX

{
    "result" : "Can not resolve a single PriceSeed by a given key"
}
```

Get Multiple Price Seeds

Request all market simulator price seed entities.

Resource URI

```
GET /api/priceseeds
```

Example: Get all price seeds

```
GET /api/priceseeds HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible reply

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: XXX

{
    "priceSeeds" : [
        {
            "id" : 2,
            "symbol" : "DE0007664005",
            "securityType" : "CS",
            "priceCurrency" : "EUR",
            "securityId" : "DE0007664005",
            "securityIdSource" : "ISI",
            "instrumentSymbol" : "DE0007664005",
            "midPrice" : 140.0,
            "bidPrice" : 135.0,
            "offerPrice" : 145.0,
            "lastUpdate" : "2023-May-10 12:47:08.252739"
        },
        {
            "id" : 3,
            "symbol" : "EUR/USD",
            "securityType" : "FXSPOT",
            "priceCurrency" : "USD",
            "securityId" : "",
            "securityIdSource" : "",
            "instrumentSymbol" : "EUR/USD",
            "midPrice" : 31.345,
            "bidPrice" : 30.435,
            "offerPrice" : 33.435,
            "lastUpdate" : "2023-May-10 12:47:08.252739"
        }
    ]
}
```

Add Price Seeds

Add new market simulator price seed entity.

Resource URI

```
POST /api/priceseeds
```

Example: Add new price seed

```
POST /api/priceseeds HTTP/1.1
```

```
Accept: application/json;charset=UTF-8
Host: localhost

{
    "symbol" : "AAPL",
    "securityType" : "CS",
    "priceCurrency" : "USD",
    "instrumentSymbol" : "AAPL",
    "midPrice" : 120.5,
    "bidPrice" : 110.5,
    "offerPrice" : 130.5
}
```

Example: Positive reply

```
HTTP/1.1 201 Created
Connection: Close
Content-Length: XXX

{
    "result" : "Successfully added a new price seed"
}
```

Example: Negative reply

```
HTTP/1.1 400 Bad Request
Connection: Close
Content-Length: XXX

{
    "result" : "Requested operation violates data integrity
constraints"
}
```

Update Price Seeds

Update existing market simulator price seed entities.

Resource URI

```
PUT /api/priceseeds/{id}
```

Example: Update price seed

```
PUT /api/priceseeds/4 HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost

{
    "midPrice" : 120.5,
    "offerPrice" : 123.5
}
```

Example: Positive reply

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: XXX

{
    "result" : "Successfully updated the price seed with 4
identifier"
}
```

Example: Negative reply

```
HTTP/1.1 404 Not Found
Connection: Close
Content-Length: XXX

{
    "result" : "Can not resolve a single PriceSeed by a given key"
}
```

Delete Price Seeds

Delete market simulator price seeds entity.

Resource URI

```
DELETE /api/priceseeds/{id}
```

Example: Delete price seed

```
DELETE /api/priceseeds/4 HTTP/1.1
Accept: application/json;charset=UTF-8
```

```
Host: localhost
```

Example: Positive reply

```
HTTP/1.1 204 No Content
Connection: Close
Content-Length: 0
```

Example: Negative reply

```
HTTP/1.1 404 Not Found
Connection: Close
Content-Length: XXX

{
    "result" : "Can not resolve a single PriceSeed by a given key"
}
```

Sync Price Seeds

Retrieve and update price seeds from an external database

Resource URI

```
PUT /api/syncpriceseeds
```

Example: Sync price seeds

```
PUT /api/syncpriceseeds HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible reply

```
HTTP/1.1 200 OK
Connection: Close
Content-Length: 45

{
    "result" : "Price seeds successfully synchronized"
}
```

Administrative Commands

System Status

Query current live status of system.

Resource Properties

Path	Type	Description
id	Text	Venue short name
name	Text	Venue full name
startTime	Text	Time when the instance started
version	Text	Version of MktSimulator

Get System Status

Request system status.

Resource URI

```
GET /api/status
```

Example: Request system status

```
GET /api/status HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "id" : "LSE",
    "name": "London Stock Exchange",
    "startTime":"2022-Feb-03 12:16:50"
    "version" : "99.116.15829062-734713bc (28/02/2020 17:40:18)"
}
```

Venue Status

Manage current live status of each market simulator venue instance.

Resource Properties

Path	Type	Description
id	Text	Venue short name
name	Text	Venue full name
startTime	Text	Time when the instance started
version	Text	Version of MktSimulator
responseCode	Integer	HTTP code received in response to venue status request

Get Single Venue Status

Request a single market simulator venue entity status.

Resource URI

```
GET /api/venuestatus/{venueId}
```

Example: Request single venue status

```
GET /api/venuestatus/LSE HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "id" : "LSE",
    "name": "London Stock Exchange",
    "startTime":"2022-Feb-03 12:16:50"
    "version" : "99.116.15829062-734713bc (28/02/2020 17:40:18)"
    "statusCode" : 200
}
```

Get Multiple Venue Status

Request all market simulator venue entity status.

Resource URI

```
GET /api/venuestatus
```

Example: Request single venue status

```
GET /api/venuestatus HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "venuestatus": [
        {
            "id":"FASTMATCH",
            "name":"FASTMATCH",
            "startTime":"2023-Feb-09 09:34:42.216182",
            "version":"develop.3.1675934598-87a7b61c(2023-02-09
11:23:18 +0200)",
            "statusCode":200
        },
        {
            "id":"XETRA",
            "name":"XETRA",
            "startTime":"2023-Feb-09 09:34:42.216182",
            "version":"develop.3.1675934598-87a7b61c(2023-02-09
11:23:18 +0200)",
            "statusCode":200
        },
        {
            "id":"BSE",
            "name":"BSE",
            "startTime":"2023-Aug-07 11:28:19.623110",
            "version":"3.1691404393_9ee00fc9(2023-08-07 13:33:13
+0300)",
            "statusCode":503
        }
    ]
}
```

Order Generation

Start/Stop Order Generation

Start or stop generation of random orders and historical data for instance indicated by VenueID

Resource URI

```
PUT /api/[genstart|genstop]/{venueId}
```

Example: Request to start generation for XETRA instance

```
PUT /api/genstart/XETRA HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result" : "Random orders generator started successfully"
}
```

Example: Request to stop generation for XETRA instance

```
PUT /api/genstop/XETRA HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Possible reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result" : "Random orders generator stopped successfully"
}
```

Status of Order Generation

Get whether generation of random orders and historical data for instance indicated by VenueID is enabled

Resource URI

```
GET /api/genstatus/{venueId}
```

Example: Request for status of generation for LSE instance

```
GET /api/genstatus/LSE HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Reply in case the generation is running

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result" : "Running"
}
```

Example: Reply in case the generation is NOT running

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result" : "NotRunning"
}
```

Example: Reply in case a venue does not exist

```
HTTP/1.1 502 Bad Gateway
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result" : "Could not resolve destination instance with AAAAA
identifier"
}
```

Market Phase Halt

See this section for more information about market phases.

Halt Market Phase

Halt current market phase.

Resource Properties

Path	Type	Description
allowCancels	Boolean	Defines whether order cancellation by OrderCancelRequest (35=F) is allowed during the Open Halted phase. • `true` - OrderCancelRequest (35=F) can be processed. If the order is cancelled, the market simulator's response is ExecutionReport (35=8), in which OrdStatus is Canceled (39=4). • `false` - OrderCancelRequest (35=F) will be rejected - orders cannot be cancelled. The market simulator's response is OrderCancelReject (35=9), in which OrdStatus is Rejected (39=8).

Resource URI

```
PUT /api/halt/{venueId}
```

Example: Halt market for LSE instance

```
PUT /api/halt/LSE HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost

{
    "allowCancels": false
}
```

Example: Positive reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result" : "Market successfully halted"
}
```

Example: Reply if the phase is already halted

```
HTTP/1.1 409 Conflict
Connection: Close
Content-Length: XXX

{
    "result" : "The market is already halted."
}
```

Example: Reply if there is no active phase

```
HTTP/1.1 404 Not Found
Connection: Close
Content-Length: XXX

{
    "result" : "There is no phase to halt."
}
```

Example: If the current phase cannot be halted

```
HTTP/1.1 409 Conflict
Connection: Close
Content-Length: XXX

{
    "result" : "Unable to halt the phase."
}
```

Resume Market Phase

Resume the phase that was halted by a halt request.

Resource URI

```
PUT /api/resume/{venueId}
```

Example: Resume market for LSE instance

```
PUT /api/resume/LSE HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Positive reply

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result" : "The market was successfully resumed."
}
```

Example: Reply if the phase is not currently halted

```
HTTP/1.1 409 Conflict
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result" : "There is no halt request to terminate."
}
```

Persisted State

Store Market State for Single Venue

Store current state of active traded listings for current venue to a persisted state file path configured for the venue.

Status Code	Response
201 CREATED	`{ "result": "Matching engine state has been successfully persisted." }`

Status Code	Response
403 FORBIDDEN	`{ "result": "Persistence is disabled." }`
409 CONFLICT	`{ "result": "The persistence file path is empty." }`
409 CONFLICT	`{ "result": "The persistence file path is unreachable." }`
409 CONFLICT	`{ "result": "An error occurs when opening the persistence file." }`
409 CONFLICT	`{ "result": "An error occurs when writing to the persistence file." }`

Resource URI

```
POST /api/store
```

Example: Store state for current venue

```
POST /api/store HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Positive reply

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result": "Matching engine state has been successfully
persisted."
}
```

Store current state of active traded listings for a specific venue to a persisted state file path configured for the venue.

Resource URI

```
POST /api/store/{venueId}
```

Example: Store state for venue LSE

```
POST /api/store/LSE HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Positive reply

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result": "Matching engine state has been successfully
persisted."
}
```

Recover Market State for Single Venue

Recover state of active traded listings for current venue from a persisted state file configured for the venue.

Status Code	Response
201 CREATED	`{ "result": "Matching engine state has been successfully recovered." }`
403 FORBIDDEN	`{ "result": "Persistence is disabled." }`
409 CONFLICT	`{ "result": "The persistence file path is empty." }`
409 CONFLICT	`{ "result": "The persistence file path is unreachable." }`
409 CONFLICT	`{ "result": "An error occurs when opening the persistence file." }`
409 CONFLICT	`{ "result": "The persistence file is malformed." }` or `{ "result": "The persistence file is malformed: {}" }` where “{}” is a placeholder for the details of the error.

Resource URI

```
POST /api/recover
```

Example: Recover state for current venue

```
POST /api/recover HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Positive reply

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result" : "Matching engine state has been successfully
recovered"
}
```

Recover state of active traded listings for a specific venue from a persisted state file configured for the venue.

Resource URI

```
POST /api/recover/{venueId}
```

Example: Recover state for venue LSE

```
POST /api/recover/LSE HTTP/1.1
Accept: application/json;charset=UTF-8
Host: localhost
```

Example: Positive reply

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=UTF-8
Content-Length: XXX

{
    "result" : "Matching engine state has been successfully recovered"
}
```