

MATS Push Client Application Design Guide

EPEX SPOT
02.06.2026

Content

1.	Introduction	5
1.1	Audience	5
1.2	Purpose	5
1.3	Document revision history	5
2.	Glossary	7
3.	Getting started	8
3.1	MATS user account and permissions	9
3.2	Security and Certificates	9
3.3	Message delivery guarantee	9
4.	Push architecture	10
5.	Connect	11
5.1	Connection lifecycle	11
5.2	Technical requirements	11
5.3	Push client session and Token validity	12
6.	Event lifecycle	13
7.	Disconnect	15
7.1	Heartbeat	15
7.2	Detection of automatic disconnection cases	16
8.	Event names & characteristics	17
8.1	Available custom event names	17
8.2	Event characteristics	18
9.	Subscribe to event with <i>filters</i>	20
9.1	Event Name: subscribe_orders	20
9.2	Event Name: subscribe_auction_sessions	22

9.3	Event Name: subscribe_auctions	23
10.	Notifications	25
10.1	Event Name: orders	25
10.2	Event Name: auction_sessions	28
10.3	Event Name: disconnect_messages	30
10.4	Event Name: disconnect_messages	31
10.5	Event Name: auctions	33
11.	cknowledgements	35
12.	Recovery mechanism	36
13.	Technical guidance	37
13.1	Socket.io libraries	37
13.2	Java	37
13.3	C sharp	37

1. Introduction

1.1 Audience

This Push Application Design Guide is meant for:

- developers implementing client applications who would like to use real time notifications via MATS Push solution.
- any other profile like project managers, or business analysts, who would like to understand the Push solution fundamentals.

1.2 Purpose

The purpose of this document is to describe the principles of the MATS Push API, giving enough elements and guidelines:

- to understand the MATS Push fundamentals,
- to build an efficient “push client” that would use the advantage of real time notifications.

1.3 Document revision history

Evolution	Date	Version	Author	Comments
Update	12/12/2024	1.1	EPEX SPOT API Customer Services	Include message delivery guarantee Include heartbeat mechanism Include list of acknowledgements Include recovery mechanism
Update	14/01/2025	1.2	EPEX SPOT API Customer Services	Include auctionSession related to events Include “module” as a possible filter Include filters for event: subscribeAuctionSession Describe the notification json schema for event: auctionSession Include Technical guidance
Update	29/04/2025	1.3	EPEX SPOT API Customer Services	Includes new event: disconnect_messages Renaming of existing event names Includes the details about automatic disconnection cases handling Update of Acknowledgement messages returned by Push server in case of success
Update	01/05/2025	1.4	EPEX SPOT API Customer Services	Update the Appendix – Change with the list of changes applicable for SIMU 1
Update	16/07/2025	1.5	EPEX SPOT API Customer Services	Update of json notification for event: order Include Change #3 for the overview of changes with next release in SIMU
Update	29/08/2025	1.6	EPEX SPOT API Customer Services	Correction of disconnectReason enum string values

Update	02/06/2026	1.7	EPEX SPOT API Customer Services	Include details about new events subscribe_auctions, auctions, unsubscribe_auctions Update of event "orders" with new fields user & actionReason
--------	------------	-----	------------------------------------	--

2. Glossary

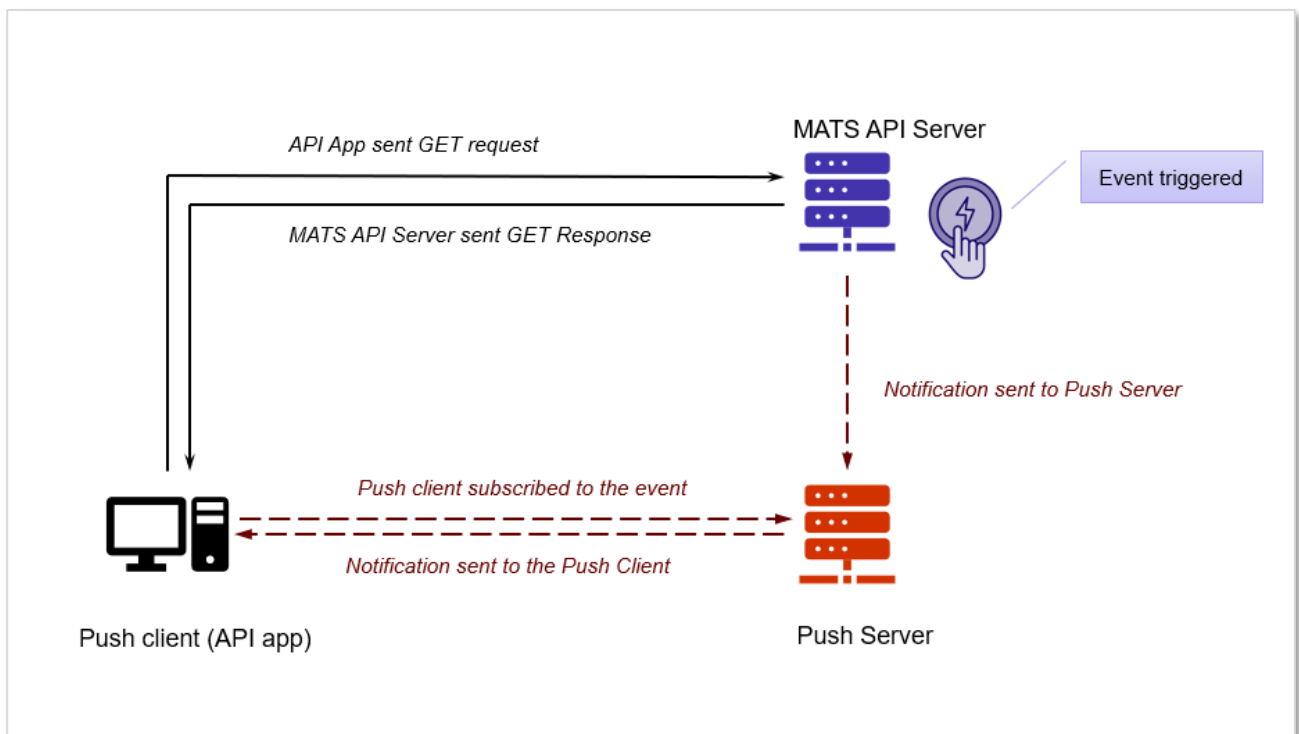
Term	Definition
MATS	Multiple Auction Trading System
MATS API	RESTful API with JSON requests and responses format. Enables customers to perform their trading and monitoring activities in the Day-Ahead and Intraday Power Auction Markets.
Push client	Any application (Front end or API apps built by customers) operated by a user who is either a Trader or a Non Market Participant (Market Data “Data Vendor” client), who is authorized to access the auction trading system.
Push Server	System which pushes real time notifications to end users, about dedicated “events”, based on specific functional rules.
Event	A technico-functional object which has a “functional” name. Based on business and functional aspects, there are several events available at Push server level. Notifications are pushed to specific <i>Event Name</i> which is being listened by a user.
Emit	The “action” referred to when a system is “pushing” a notification to an event name.
Listen	The “action” referred to when a user/system is listening to an “event name” so that it can retrieve the notifications which are pushed by the Push server.

3. Getting started

The MATS Push solution is using the **Socket.IO** technology. The primary advantage of using push notifications is that to make polling requests to the MATS REST API based on event-driven instead of time-driven (periodic “loops” because the API app does not know when it is required to refresh its data).

The below diagram depicts a simple high level process using the push mechanism:

- In real time, push client(s) connected to the push server are notified about the “event” triggered on the MATS API server (e.g. new order on the user’s scope, publication of market results).
- When receiving a notification from push server, MATS API applications must react by sending a GET request to the MATS REST API Server to refresh their data, based on the functional scope indicated in the push notification.
- Benefits:
 - by removing the need of periodic polling requests, this push mechanism significantly helps reduce the load on both customers’ API apps and the MATS API server.
 - The only requests that remain are the one sent:
 - when starting the app (“initialization phase”)
 - and for sporadic ad hoc needs (e.g. resulting from a user clicking on an API screen to refresh the app GUI, or from a need to force the retrieval of market results).



3.1 MATS user account and permissions

Customers who wish to build a push client application must have a valid API user account configured in MATS trading system, with the relevant trading permissions configured.

This is a mandatory pre-condition that must be fulfilled to be able to connect to the push server and therefore consume push notifications.

Please refer to the *MATS Fundamentals* document in the Functional package for more details about user configuration and portfolio set up.

3.2 Security and Certificates

No client certificate is required to connect to the Push server. As stated in the above section, the only mandatory pre-requisite is to have a valid API user account configured in MATS.

However, customers may have to retrieve the public Server certificate from the Push server and add it into their trust store. **TLS v1.2** is supported if an SSL verification is done during technical connect to the web socket.

3.3 Message delivery guarantee

The message delivery guarantee is **at most once**:

- Messages are delivered once, and if there is a system failure, messages may be lost and are not redelivered.

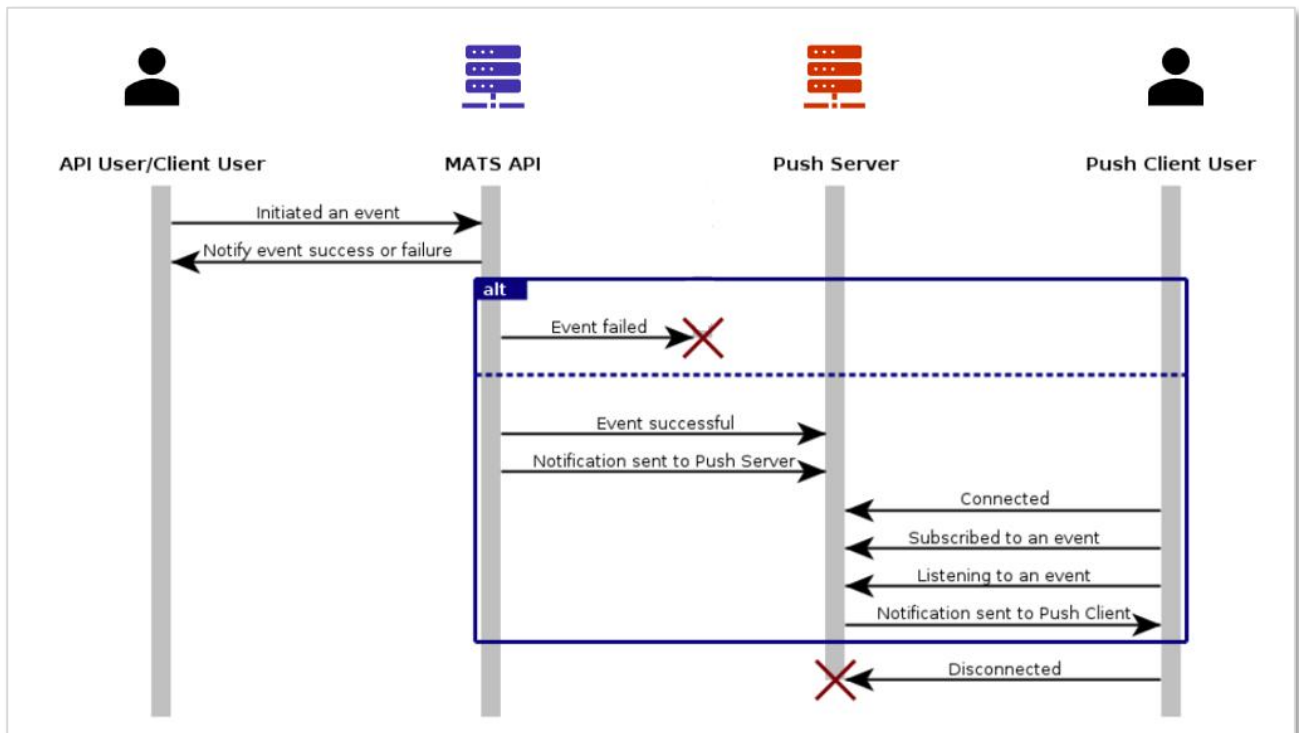
In case of disconnection, app restart etc, please refer **section 12** for the recommended recovery mechanism.

4. Push architecture

This diagram shows the client app two components that must be implemented:

- The REST API client/user on the left side,
- The Push client/(same) user on the right side.

Though both components will most likely be implemented in the same application, it is possible to implement two separate apps, as long as the Push client can share received notifications with the REST API client.



Note: as shown on the above diagram, failed events on the MATS backend side (e.g. order submission being rejected) are not forwarded to the Push server. Only successful events are sent to the Push server. Push clients will never receive a notification from a failed event.

5. Connect

5.1 Connection lifecycle

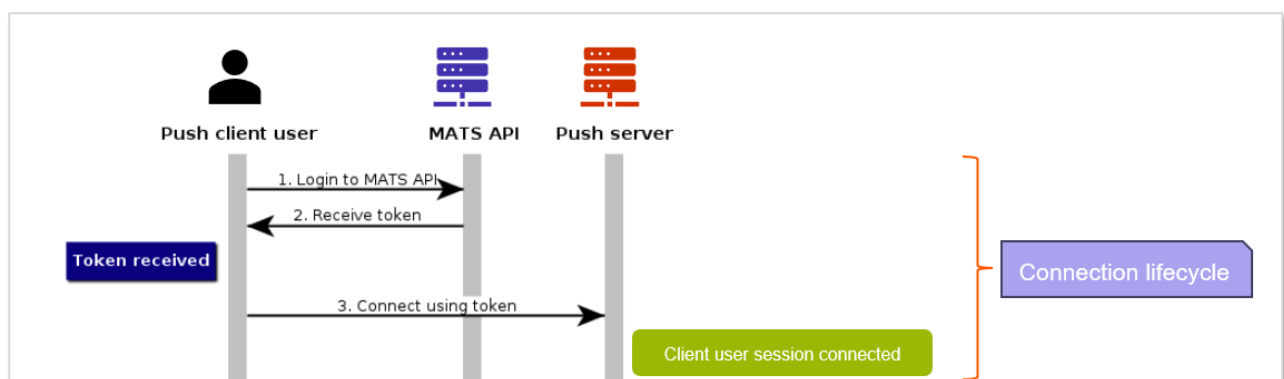
As a first step, the user must log in MATS using the MATS REST API *Connect* request, using the user Login & Password: the JSON response contains a **token**.

Example:

```
{
  "data": {
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiU9.eyJpYXQiOiJlZS...",
    "group": "TRADER",
    "userName": "APIUSER",
    "login": "APIUSER",
    "tokenDuration": 1440,
    "alreadyConnected": true,
    "passwordTimeIntervalBeforeExpiry": 7444930
  }
}
```

Thanks to this token, the push client can connect to the push server using the **event name: connect** (the user name is not required, since implicitly contained in the token).

Once a connection to the push server is successfully established, the push client is “**ready to start the Event lifecycle**” (further described in **section 6**).



5.2 Technical requirements

- A WebSocket connection to the push server must be created.
- To connect to the push server, use the *Auth:Token* attribute (not *bearer*). Refer to **section 13** for more details about the technical specifics.

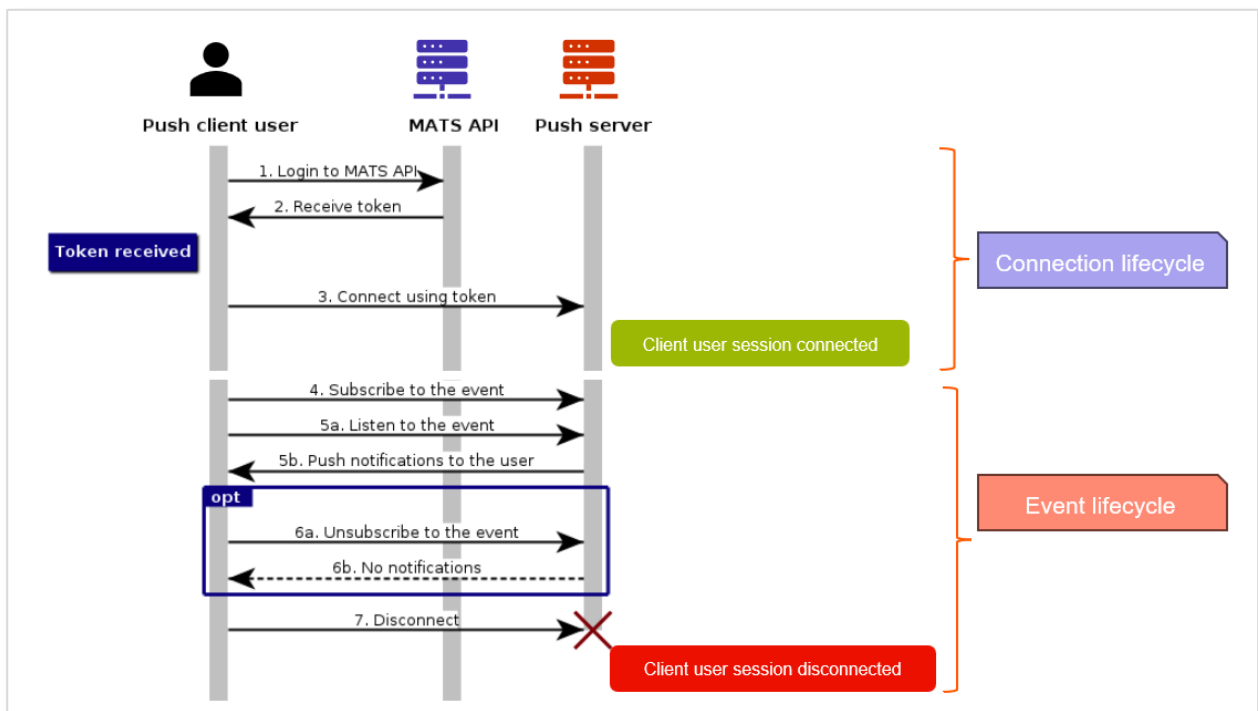
5.3 Push client session and Token validity

Rule	Description
1	Push client must be connected to the push server using the “access_token”. The token validity to the push server is same as provided by the MATS Authentication API (“expires_in” in the Login response) <pre> { "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJHdjNpYXNj", "expires_in": 86400, "refresh_expires_in": 86400, "refresh_token": "eyJhbGciOiJIUzUxMiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJHdjNpYXNj", "token_type": "Bearer", "id_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJHdjNpYXNj", "not-before-policy": 0, "session_state": "c307dfc5-5e52-47e7-83c0-15d526e640ea", "scope": "openid custom-profile" }</pre> Once the token validity is reached, the user session ends and the token becomes invalid : the connection to the push server ends automatically. Refer section 7.2 for more details about how to detect such cases.
2	Maximum one session can be active on the push server for a given user. This implies only one token can be used per user to connect to push server.
3	If a user tries to connect to the push server with different tokens while there is already an active token connected to the push server, an error with code: PSHIF0003 will be raised.

6. Event lifecycle

Once a user is successfully connected to the push server (as described in **section 5.1: Connection lifecycle**), then push client must follow the below steps in order to receive notifications on an **event name**.

- **Subscribe to the event:**
 - A Push client must “emit” to the “subscribe event name”. This process is mandatory to let the push server know that the push client would like to receive notifications on the event name that will be listened to (next step).
 - Refer to **section 8** for the list of subscribe event names (e.g. *subscribe_orders*).
- **Listen to the event:**
 - A Push client must listen to the event name. This process is mandatory so that the push client can receive and consume the notifications sent by the push server on the specific event name.
 - Refer to **section 8** for the list of listening event names (e.g. *orders*).
- **Unsubscribe from the event:** (optional)
 - A push client can unsubscribe from the event name. Once a user is unsubscribed, it will no longer receive any notification on the “listening event”.
 - Unsubscribing from an event does not disconnect a push client session from the push server.
 - Refer to **section 8** for the list of unsubscribe event names (e.g. *unsubscribe_orders*).



*The above described steps are **limited to the current client user session connection**. After the client user is disconnected from the push server, the user must re-initiate the process again as described above.*

7. Disconnect

In order to close the connection to the Push server, the Push client user session must use the **event name: disconnect** push client user session:

- This will terminate the user session & token from the push server.
- The Push server will no longer send any notification to the user.

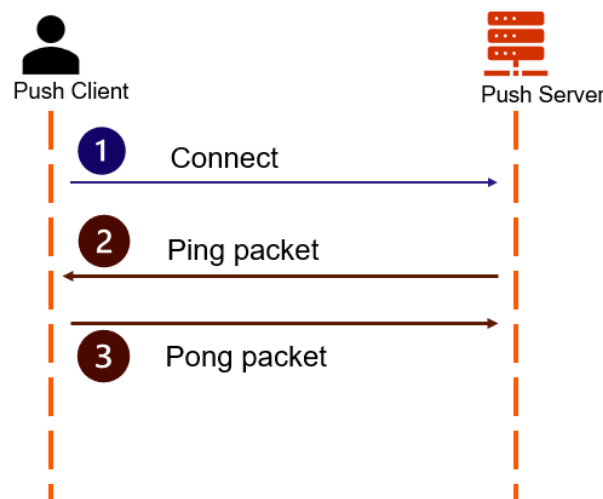
Point of Attention

Disconnecting (logging out) from the Push server does not lead to a disconnection from MATS REST API.

7.1 Heartbeat

A heartbeat mechanism enables the Push client to **monitor whether the connection between the server and the client is still up and running**.

This heartbeat mechanism is natively implemented within socket.io libraries: as long as your application uses a socket.io third party library, it will automatically send a PONG packet when receiving a PING packet from the Push server, as explained below:



Once the connection is completed, the heartbeat mechanism is automatically started:

- At a given interval (the **pingInterval** value set and sent by the Push server during the handshake) the server sends a **PING** packet and the client has a few seconds (the **pingTimeout** value) to send a **PONG** packet back.
- As soon as the server does not receive a **PONG** packet back in due time, it considers that the connection is closed and terminates the user session & token from the Push

server. The Push client will no longer receive notifications. In this situation, client must perform the recovery mechanism as described in **section 12**.

- Conversely, if the client does not receive a **PING** packet within ***pingInterval* + *pingTimeout***, it should consider that the connection is closed. In this situation, client must perform the recovery mechanism as described in **section 12**.

To get the default values set by the server during the handshake when creating a Socket.IO connection, you can listen for the *connect* event and then access the *socket.io* object on the client. This object contains information about the connection, including the default values sent by the server.

We recommend to keep:

- the ***pingTimeout*** Server default value as **20 seconds** on client side.
- the ***pingInterval*** Server default value as **25 seconds** on client side.

The above recommended configurable values are modifiable if customers see the need to change the values. Should you need to modify these values, please get in touch with us.

It is advised to catch disconnections, including those due to ping timeouts at the client side programming. This is achievable by using the *socket.on(Socket.EVENT_DISCONNECT, callback)* event listener function.

7.2 Detection of automatic disconnection cases

We strongly recommend to listen to the ***event: disconnect_messages*** to be informed about automatic disconnection of push client user from push server when:

- *Token has expired*
- *User is kicked out from the MATS trading system*
- *User logged out (disconnected) himself from MATS trading system*

Refer **section 10.3** for more details about the notification structure.

Please note that custom ***event: disconnect_messages*** is not triggered due to “core” disconnection errors such as ping timeout, manual disconnection by the push client using *socket.disconnect()*, any possible connection error due to network failure etc. In order to catch these reason, we advise you to use the native *socket.io* library. More details are available [here](#).

8. Event names & characteristics

8.1 Available custom event names

The below table gives an overview of available **custom event names**, indicating which one must be emitted or listened to by a push client.

Category	Event Name	Push client		Push Server	
		Emit	Listen	Emit	Listen
Subscribe Event	subscribe_orders	Yes	N/A	N/A	Yes
Listening Event	orders	N/A	Yes	Yes	N/A
Unsubscribe Event	unsubscribe_orders	Yes	N/A	N/A	Yes
Subscribe Event	subscribe_auction_sessions	Yes	N/A	N/A	Yes
Listening Event	auction_sessions	N/A	Yes	Yes	N/A
Unsubscribe Event	unsubscribe_auction_sessions	Yes	N/A	N/A	Yes
Subscribe Event	subscribe_auctions	Yes	N/A	N/A	Yes
Listening Event	auctions	N/A	Yes	Yes	N/A
Unsubscribe Event	unsubscribe_auctions	Yes	N/A	N/A	Yes
Listening Event	disconnect_messages	N/A	Yes	Yes	N/A

*Please note that **custom event names** are case sensitive (i.e. **orders** ≠ **Orders**).*

Therefore, user must use event names as described above.

8.2 Event characteristics

Event Name	Characteristics
subscribe_orders	• Event on which a push client can subscribe to, so that it can listen to event: orders for receiving notifications. • This is mandatory as a the push client will receive notifications only if the client subscribed to the event "subscribe_orders".
orders	• Event on which push client can listen to receive notifications about the following order events: ○ New order ○ Modified order ○ Deleted order
unsubscribe_orders	• Event on which a push client can unsubscribe, so that it can unsubscribe from event: orders. This is an optional step. • After a push client has unsubscribed, the user will no longer receive any notification from the push server.
subscribe_auction_sessions	• Event on which a push client can subscribe to, so that it can listen to event: auction_sessions for receiving notifications. • This is mandatory as a the push client will receive notifications only if the client subscribed to the event "subscribe_auction_sessions".
auction_sessions	• Event on which push client can listen to receive notifications about the following order events: ○ Auction session status change (OPEN, CLOSED, PUBLISHED, CANCELLED) ○ Result status (UNAVAILABLE, PRELIMINARY, FINAL)
unsubscribe_auction_sessions	• Event on which a push client can unsubscribe, so that it can unsubscribe from event: auction_sessions. This is an optional step. • After a push client has unsubscribed, the user will no longer receive any notification from the push server.
subscribe_auctions	• Event on which a push client can subscribe to, so that it can listen to event: auctions for receiving notifications.

	This is mandatory as a the push client will receive notifications only if the client subscribed to the event "subscribe_auctions".
auctions	Event on which push client can listen to receive notifications about the following auction status events related to one or several auctions: - Switch to MAINTENANCE - Switch to RESPONSIVE
unsubscribe_auctions	Event on which a push client can unsubscribe, so that it can unsubscribe from event: auctions. This is an optional step. - After a push client has unsubscribed, the user will no longer receive any notification from the push server.

9. Subscribe to event with *filters*

This chapter describes the “filters” that a push client can *emit* on a subscribe event. These filters are functional elements (e.g. *auctionSessionId*, *auctionName*, etc.) which allow the push client to reduce the scope of notifications which it would like to receive.

Based on its business needs, a push client can use filters or not:

- *emit on a subscribe* event so that the client should receive all notifications without any filter.
- *emit on a subscribe* event with “one or several filters”, so that the client receives notifications according to the “filters” that have been sent to the push server.

Point of Attention

Technically, at most one socket.io instance is possible. This means it is **not possible** to have several socket.io instances with **different combinations of subscription filters “active” for the same event at the same time.**

For example:

If the push client app subscribed to an event with filters [{"auctionSessionId":1}] as a first step and then the push client subscribed to the same event with filters [{"auctionSessionId":2, "module": "LINEAR_ORDER"}], client will receive only notifications based on the latest subscribed filters [{"auctionSessionId":2, "module": "LINEAR_ORDER"}]

9.1 Event Name: **subscribe_orders**

9.1.1 Available for

- Trader

9.1.2 Subscribe with no filter

If a client wants to receive all notifications (without any filter), then an empty JSON must be passed as an attribute to the **event name: subscribe_orders**.

9.1.2.1 JSON schema

```
[ {} ]
```

9.1.3 Subscribe with filters

If a client wants to receive notifications based on specific filters, then it can:

- choose either one of the below filters,
- or a combination of two or more filters

The filters must be passed as an attribute in JSON format as described below to the **event name: subscribe_orders**.

9.1.3.1 JSON schema

```
[
  {
    "auctionSessionId": 1,
    "module": "LINEAR_ORDER",
    "portfolio": "PORTFOLIO-T01",
    "biddingLevel": "NL-TENNETNL"
    "duration": 60
  }
]
```

9.1.3.2 JSON schema description

Element	Data type	Description	Example
auctionSessionId	Integer	Technical id of the auction session for which the user wishes to receive notifications when an order event happens.	1
module	String	LINEAR_ORDER BLOCK_ORDER SCALABLE_COMPLEX_ORDER Indicates the order type for which the user wishes to receive notifications	LINEAR_ORDER
portfolio	String	Portfolio name for which the user wishes to receive notifications when an order event happens on this specific portfolio.	PORTFOLIO-T01
biddingLevel	String	Bidding level name for which the user wishes to receive notifications when an order event happens on this specific bidding level.	NL-TENNETNL
duration	Integer	Duration for which the user wishes to receive notifications when an order event happens on this specific duration.	60

9.2 Event Name: **subscribe_auction_sessions**

9.2.1 Available for

- Trader, Non Market Participant

9.2.2 Subscribe with no filter

If a client wants to receive all notifications (without any filter), then an empty JSON must be passed as an attribute to the **event name: subscribe_auction_sessions**.

9.2.3 Subscribe with filters

If a client wants to receive notifications based on specific filters, then it can:

- choose either one of the below filters,
- or a combination of two or more filters

The filters must be passed as an attribute in JSON format as described below to the **event name: subscribe_auction_sessions**.

9.2.3.1 JSON schema

```
[
  {
    "auctionSessionId": 1,
    "auctionName": "CORE",
    "auctionSessionStatus": "OPEN",
    "auctionDate": "2025-02-15"
  }
]
```

9.2.3.2 JSON schema description

Element	Data type	Description	Example
auctionSessionId	Integer	Technical id of the auction session for which the user wishes to receive notifications when an auction session event happens.	1
auctionName	String	Unique auction name for which the user wishes to receive notifications when an auction session event happens	CORE
auctionSessionStatus	String	OPEN CLOSED PUBLISHED CANCELLED Auction session status for which the user wishes to receive notifications. This means when the auction session status is updated to "OPEN", user will receive notifications for status change: "OPEN"	OPEN
auctionDate	String	Auction date for which user wishes to receive notifications. <i>Format: yyyy-mm-dd and the time zone is UTC.</i>	2025-05-15

9.3 Event Name: subscribe_auctions

9.3.1 Available for

- Trader, Non Market Participant

9.3.2 Subscribe with no filter

If a client wants to receive all notifications (without any filter), then an empty JSON must be passed as an attribute to the **event name: subscribe_auctions**

9.3.3 Subscribe with filters

If a client wants to receive notifications based on specific filters, then it can:

- choose either one of the below filters,
- or a combination of two or more filters

The filters must be passed as an attribute in JSON format as described below to the **event name: subscribe_auctions**

9.3.3.1 JSON schema

```
[
  {
    "auctionName": "CORE",
    "auctionStatus": "OPEN",
  }
]
```

9.3.3.2 JSON schema description

Element	Data type	Description	Example
auctionName	String	Unique auction name for which the user wishes to receive notifications when an auction session event happens	CORE
auctionStatus	String	MAINTENANCE RESPONSIVE Auction status for which the user wishes to receive notifications. This means when the auction status is updated to "MAINTENANCE", user will receive notifications for status change: "MAINTENANCE"	MAINTENANCE

10. Notifications

10.1 Event Name: orders

JSON schema of the notification that is pushed by the push server to the **event name: orders**

10.1.1 Available for

- Trader

10.1.2 User permissions

- Notifications sent to the push client users are based on the user's trading permissions configured in the MATS trading system

10.1.3 JSON schema

```
[
  {
    "user": "USERA",
    "notificationId": 1001,
    "notificationDateTime": "2024-11-11T14:00:00.000Z",
    "module": "LINEAR_ORDER",
    "action": "SUBMIT",
    "actionReason": "ACTION_BY_USER",
    "auctionSessionId": 1,
    "portfolio": "PORTFOLIO-T01",
    "biddingLevel": "NL-TENNETNL"
    "duration": 60,
    "orderIds": [100000000012341]
  }
]
```

10.1.4 JSON schema description

Element	Data Type	Description	Example
user	String	User name who submitted/modified/deleted the order(s)	USERA
notificationId	Integer	Technical id of the notification. This is provided only for informative purpose. <i>Remark: Could be useful during the communication with EPEX SPOT Support team in case of issues and/or clarification needs.</i>	1001
notificationDateTime	String	Date and time in UTC at which the notification has been triggered. <i>Format: yyyy-mm-ddThh:mm:ss.sssZ</i>	2024-11-11T14:00:00.000Z
module	String	LINEAR_ORDER BLOCK_ORDER SCALABLE_COMPLEX_ORDER FUTURE_TO_SPOT_ORDER Indicates to which order type the notification is related.	LINEAR_ORDER
action	String	SUBMIT DELETE CONFIRM - SUBMIT: in case of a new order(s) submission and / or modification of one or several existing orders. - DELETE: in case an existing order(s) is deleted. - CONFIRM: in case of confirmation of an existing order. Valid only for FUTURE_TO_SPOT_ORDER module	SUBMIT
actionReason	String	ACTION_BY_USER TRADING_LIMIT_BREACH AUCTION_CONFIGURATION_UPDATE BACKOFFICE_CONFIGURATION_UPDATE ACTION_BY_USER: The event is triggered due to an action performed by a user. This is	ACTION_BY_USER

		possible for action: SUBMIT DELETE CONFIRM. • TRADING_LIMIT_BREACH: Indicates that orders were deleted due to trading limit breach. This is applicable only for action: DELETE. • AUCTION_CONFIGURATION_UPDATE: Indicates that orders were deleted due to auction configuration update (e.g., min price update). This is applicable only for action: DELETE. • BACKOFFICE_CONFIGURATION_UPDATE: Indicates that orders were deleted due to backoffice configuration update. This is applicable only for action: DELETE.	
auctionSessionId	Integer	Technical id of the auction session on which the order has been submitted, modified or deleted.	1
portfolio	String	Portfolio name of the related order.	PORTFOLIO-T01
biddingLevel	String	Bidding level name of the related order.	NL-TENNETNL
duration	Integer	Duration of the related order.	60
orderIds	array[Integer]	List of order id(s) that has been submitted, deleted or confirmed	100000000012341

10.2 Event Name: auction_sessions

JSON schema of the notification that is pushed by the push server to the **event name: auction_sessions**

10.2.1 Available for

- Trader, Non Market Participant

10.2.2 User permissions

- Notifications sent to the push client users are based on the user's trading permissions configured in the MATS trading system

10.2.3 JSON schema

```
[
  {
    "notificationId": 1005,
    "notificationDateTime": "2025-01-01T11:00:00.000Z",
    "auctionSessionId": 1,
    "auctionSessionDateTime": "2025-02-15T11:00:00.000Z",
    "auctionName": "CORE",
    "auctionSessionStatus": "OPEN",
    "resultStatus": "UNAVAILABLE"
  }
]
```

10.2.4 JSON schema description

Element	Data Type	Description	Example
notificationId	Integer	Technical id of the notification. This is provided only for informative purpose. <i>Remark: Could be useful during the communication with EPEX SPOT Support team in case of issues and/or clarification needs.</i>	1005
notificationDateTime	String	Date and time in UTC at which the notification has been triggered. <i>Format: yyyy-mm-ddThh:mm:ss.sssZ</i>	2025-01-01T11:00:00.000Z
auctionSessionId	String	Technical id of the auction session	1
auctionSessionDateTime	String	Date and time in UTC of the auction session <i>Format: yyyy-mm-ddThh:mm:ss.sssZ</i>	2025-02-15T11:00:00.000Z
auctionName	String	Unique auction name	CORE
auctionSessionStatus	String	OPEN CLOSED PUBLISHED CANCELLED - OPEN: Status indicate that order book is open - CLOSED: Status indicate that order book is closed - PUBLISHED: Status indicate that results is published - CANCELLED: Status indicate that auction session is cancelled and results is unavailable	OPEN
resultStatus	String	UNAVAILABLE PRELIMINARY FINAL - UNAVAILABLE: Result is not yet available. This happens when auction session is in OPEN CLOSED CANCELLED status - PRELIMINARY: Preliminary result is published. This status is only applicable for Coupled markets - FINAL: Final result is published. This status is applicable for both coupled & local markets.	UNAVAILABLE

10.3 Event Name: **disconnect_messages**

JSON schema of the notification that is pushed by the push server to the **event name: disconnect_messages**. This notification is triggered when:

- *Token has expired*
- *User is kicked out from the MATS trading system*
- *User logged out (disconnected) himself from MATS trading system*

10.3.1 Available for

- Trader, Non Market Participant

10.3.2 User permissions

- No specific user permission is checked or required.

10.3.3 JSON schema

```
{
  "notificationId": 52,
  "notificationDateTime": "2025-04-25T08:37:01Z",
  "disconnectionDateTime": "2025-04-25T08:37:16Z",
  "disconnectReason": "LOG_OUT_FROM_TRADING_SYSTEM"
}
```

10.4 Event Name: disconnect_messages

JSON schema of the notification that is pushed by the push server to the **event name: disconnect_messages**. This notification is triggered when:

- *Token has expired*
- *User is kicked out from the MATS trading system*
- *User logged out (disconnected) himself from MATS trading system*

10.4.1 Available for

- Trader, Non Market Participant

10.4.2 User permissions

- No specific user permission is checked or required.

10.4.3 JSON schema

```
{  
  "notificationId": 52,  
  "notificationDateTime": "2025-04-25T08:37:01Z",  
  "disconnectionDateTime": "2025-04-25T08:37:16Z",  
  "disconnectReason": "LOG_OUT_FROM_TRADING_SYSTEM"  
}
```

10.4.4 JSON schema description

Element	Data Type	Description	Example
notificationId	Integer	Technical id of the notification. This is provided only for informative purpose. <i>Remark: Could be useful during the communication with EPEX SPOT Support team in case of issues and/or clarification needs.</i>	1005
notificationDateTime	String	Date and time in UTC at which the notification has been triggered. <i>Format: yyyy-mm-ddThh:mm:ss.sssZ</i>	2025-04-25T08:37:01Z
disconnectionDateTime	String	Date and time in UTC at which user disconnected	2025-04-25T08:37:16Z
disconnectReason	String	LOGOUT_FROM_TRADING_SYSTEM TOKEN_EXPIRED - LOG_OUT_FROM_TRADING_SYSTEM: Reason provided when user manually logged out (disconnected himself) from MATS trading system or the user is forcefully kicked out from MATS trading system - TOKEN_EXPIRED: Reason provided when the token of the push client user has expired and therefore push server disconnected the user automatically.	CORE

10.5 Event Name: auctions

JSON schema of the notification that is pushed by the push server to the **event name: auctions**

10.5.1 Available for

- Trader, Non Market Participant

10.5.2 User permissions

- Notifications sent to the push client users are based on the user's trading permissions configured in the MATS trading system

10.5.3 JSON schema

```
[
  {
    "notificationId": 1005,
    "notificationDateTime": "2025-01-01T11:00:00.000Z",
    "auctionName": "CORE",
    "auctionStatus": "MAINTENANCE"
  }
]
```

10.5.4 JSON schema description

Element	Data Type	Description	Example
notificationId	Integer	Technical id of the notification. This is provided only for informative purpose. <i>Remark: Could be useful during the communication with EPEX SPOT Support team in case of issues and/or clarification needs.</i>	1001
notificationDateTime	String	Date and time in UTC at which the notification has been triggered. <i>Format: yyyy-mm-ddThh:mm:ss.sssZ</i>	2024-11-11T14:00:00.000Z
auctionName	String	Unique auction name for which the user wishes to receive notifications when an auction session event happens	CORE
auctionStatus	String	MAINTENANCE RESPONSIVE	MAINTENANCE

11. cknowledgements

Please find below table for the list of acknowledgments returned by the push server when the client emit on the events listed in the table.

Event Name	Acknowledgement
subscribe_orders	Successfully subscribed on Event: orders with filter: <filters used by the push client>
unsubscribe_orders	Successfully unsubscribed from Event: orders
subscribe_auction_sessions	Successfully subscribed on Event: auction_sessions with filter: <filters used by the push client>
unsubscribe_auction_sessions	Successfully unsubscribed from Event: auction_sessions
subscribe_auctions	Successfully subscribed on Event: auctions with filter: <filters used by the push client>
unsubscribe_auctions	Successfully unsubscribed from Event: auctions

12. Recovery mechanism

It is possible that the push client is no longer connected to the push server. This can happen due to various reasons. Few reasons are listed below:

- Manual disconnection by push client from push server
- Connection between client & server is closed due to the pingTimeout (refer **section 7.1**)
- Push client app server restart
- Push client session is terminated by the push server (for ex: token validity is exceeded, refer **section 5.3.**)
- Network issues

If the Push client is temporarily disconnected from the Push server and reconnects later, there is no mechanism (automatically or manually) available on Push server to recover the notifications specifically missed during the offline/disconnection period.

If there is a temporary disconnection between Push client & Push server for a certain period of time, then

- **Step 1:** Push client must connect to the Push server and start the event lifecycle as explained in **section 5 & section 6** as soon as possible to reduce the risk of a missed event between the moment where the response was built and the event subscription.
- **Step 2:** REST API application on client side must send the relevant REST API GET requests (e.g. Retrieve linear orders, Retrieve block orders, etc.) to the MATS API to retrieve the latest data from the trading system.

13. Technical guidance

13.1 Socket.io libraries

Customers who use Java or C Sharp as programming languages to develop the Push mechanism, must import third party socket.io libraries in the respective programming languages. Some references to these websites where third party libraries are available is listed below:

Programming Language	Reference Website
Java/Maven	https://mvnrepository.com/artifact/io.socket/socket.io-client
C Sharp	https://www.nuget.org/packages/SocketIOClient/#readme-body-tab

13.2 Java

Please refer to the API package Sample Test Client Java Code.

13.3 C sharp

Customers who are using C sharp as the programming language to develop the Push client app must follow the below guidance:

The “**token**” must be declared as type **Dictionary** so that when serialized the “**token**” is provided in a Key/value structure called Dictionary<String,String> as part of the “**Auth**” attribute. The code snippet related to the WebSocket protocol and connection to Push server by passing the Auth: Token is provided below as a reference.

```
Dictionary<string, string> dict = new()
{
    ["token"] = "user_token"
};

var client = new SocketIOClient.SocketIO("mats_push_url", new
SocketIOClient.SocketIOOptions

{
    Transport = SocketIOClient.Transport.TransportProtocol.WebSocket,
    Auth = dict
});

client.OnConnected += (sender, e) =>
{
    Console.WriteLine("Client connected");
};
```