API Manual ETPA

Table of Contents

Version History *4*

1. Use of the API and Server Sent Events *5*

1.2 Generate API Key **5**

1.3 Acceptance Environment **6**

1.4 Production Environment **6**

2. The API *7*

API pagination 8

2.1 On Hold Order API **9**

2.1.1 GET 9
2.1.2 GET /{orderId} 9
2.1.3 POST 9
2.1.4 DELETE 9

2.2 Order API V2.0 ..... **10**

2.2.1 GET 10
2.2.2 GET /{OrderID} 10
2.2.3 POST 10
2.2.4 PUT /{OrderID} 11
2.2.5 DELETE /{OrderID} 11

2.3 Order Status API **12**

2.3.1 GET /{OrderID} 12
2.3.2 GET /{OrderID}/current 13

2.4 Platform Status API **15**

2.4.1 GET /announcement 15
2.4.2 GET /announcement/{id} 15
2.4.3 GET /announcement/getActiveAnnouncements 15

2.5 Reporting API V2.0 ..... **16**

2.5.1 GET Orders-trades 16
2.5.2 GET Orders-trades/{tradeId} 16
2.5.3 GET Pv-party-participant 16

2.6 Trade API V2.0 ..... **16**

2.6.1 GET Trades 16
2.6.2 GET trades/{tradeID} 17
2.6.3 GET trades/recent 17

2.7 User API **17**

2.7.1 GET Individual 17
2.7.2 GET Participants 17

2.8 Wallet API V2.0 ..... **17**

2.7.1 GET Balance 18
2.7.2 GET Transactions 18

2.9 Status code **18**

3. Server Sent Events (SSE) *19*

3.1 Connecting to the SSE **19**

3.2 Announcements 19
3.3 Orderbook 20
3.3.1 Intraday orderbook 20
3.3.2 Ex-post orderbook 21
3.3.3 GOPACS orderbook 22
3.4 Trades 23
3.4.1 Intraday trades 24
3.4.2 Ex-post trades 25
3.4.3 GOPACS Trades 25
4. Websocket DEPRECATED from 01-02-2023 27
4.1 Connecting to the WebSocket server 27
4.2 General 27
4.3 Orders 28
4.4 Trades 30
4.5 Websocket Status 30
5. FAQ (Frequently asked questions) 32
1. Do you also need an API Key for the WebSocket? 32
***2.* Getting a 200 error while connecting to the correct WebSocket URL. 32**
3. Connected to the websocket server, but not getting any response. 32
Appendix A 33
Appendix B 34

Version History

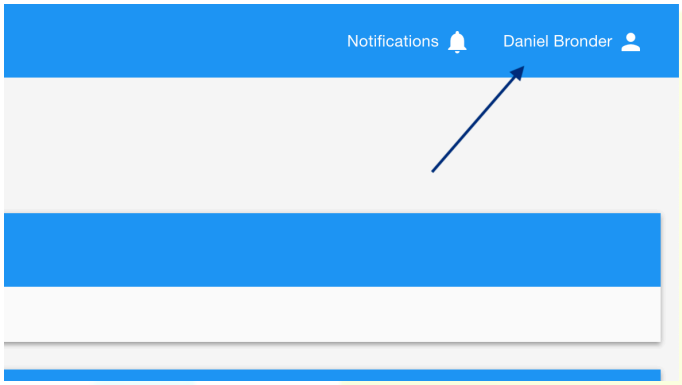
Version	Change
12-11-2019	Small explanation changes of Order Status API and add deprecation date for Order API V1.0.
05-12-2019	Add field for custom expiration time in orders.
04-03-2020	Add new Trade API and add explanation of API pagination.
11-03-2020	Add Reporting API V2.0
22-04-2020	Update explanation about pagination and versions. Added wallet API V2.0
09-11-2020	Add Matched status to documentation
19-11-2020	Add status code
09-08-2021	Update with new orderId parameters for trade and reporting API
14-09-2021	Update with new API. Trades/{tradeId} and orders-trades/{tradeId}
09-12-2021	Add explanation Rate Limiting
11-01-2022	Add explanation websocket status
25-01-2022	Add explanation about the Recent Trades API
30-03-2022	Adding On Hold Order API documentation
16-12-2022	Removal of Standard Orderbook data
27-01-2023	Add SSE implementation
07-02-2023	Add Platform Status API
17-03-2023	Add GOPACS SSE and improve Announcement SSE

1. Use of the API and Server Sent Events

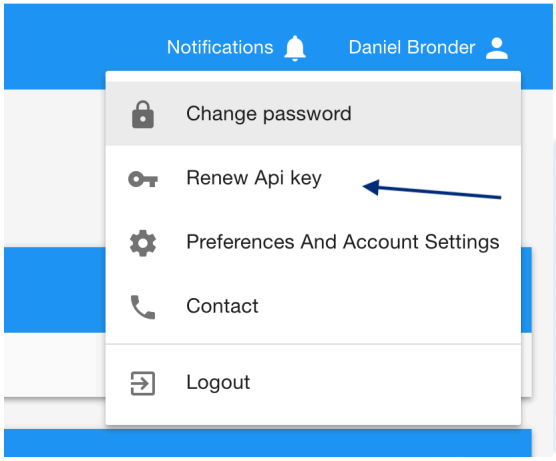
The API and WebSocket are two different ways to connect to the ETPA platform without using the Graphical User Interface (GUI) of the platform

1.2 Generate API Key

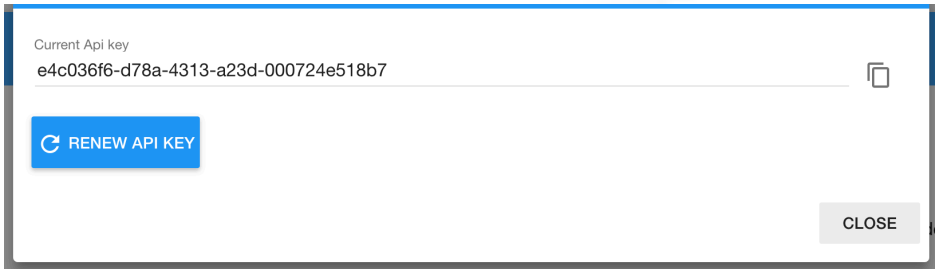
To use the API and Server Sent Events (SSE), you will need an API key. An API Key can be generated when you have an account. This can be done by login – click on name – renew API key.



Picture 1 - Generate API key 1/3



Picture 2 - Generate API key 2/3



Picture 3 - Generate API key 3/3

1.3 Acceptance Environment

Within ETPA there are two different environments that can be used. One of them is the acceptance (ACC) environment. The acceptance environment can be found here: https://acc-trading.etpa.nl. The acceptance environment is an environment where every participant and/ or broker can get familiar with the application, API and WebSocket. We highly recommend using the API and/or SSE of acceptance first before connecting your software to the production (PROD) environment.

We have 2 versions the available. Version (V1.0) is available for the following API:

Order-status and User which can be found at
https://acc-trading.etpa.nl/public-api/1.0/electricity/[API]

Version 2 (V2.0) is used for the following API's:

Order, Trade, Reporting and Wallet which can be found at
https://acc-trading.etpa.nl/public-api/2.0/electricity/[API]

On the acceptance environment you can get familiar with the API with Swagger interface: https://acc-trading.etpa.nl/swagger-ui.html (this is only on ACC and not on PROD).

At ETPA we can provide a tradingbot for testing capabilities. To test your software in different situations and scenario's. Please send an e-mail to support@etpa.nl with your request.

1.4 Production Environment

When all the code of your software with our ACC environment is working as it should and as expected, the endpoint URL can be changed (declare it therefor at one place in your code) from the to the following PROD end-point:

API: https://trading.etpa.nl/public-api/{version}/electricity/[API]
SSE : https://trading.etpa.nl/[CHANNEL]

Make sure that for this environment you use the correct API key of a user that has a PROD account and the correct rights (user role). API key of ACC will not work in PROD.

2. The API

There is a variation in versions of the APIs. Version 2 is used by Order, Trade, Wallet (transactions) and Reporting API. Version 1 is used by Order-status, Wallet (balance) and User API

The url is as following:

../{version}/electricity

Within ETPA there are four different APIs that can be used. Certain account roles can only access certain APIs as shown in Table 1.

API/User	Trade	Reporting	Wallet	View only
On Hold Order	Yes	No	No	No
Order V2.0	Yes	No	No	No
Order status	Yes	No	No	No
Trade V2.0	Yes	No	No	No
Reporting V2.0	Yes	Yes	Yes	No
User	Yes	Yes	Yes	No
Wallet V2.0	Yes	Yes	Yes	No

Table 1 - Access API

At some APIs some parameters are possible. These parameters are not required but could help with filtering the result.

Parameter	Type	Options	Description
My	Boolean	nothing, true or false	
ParticipantId	String	Nothing	Only necessary for brokers and pv participants
OrderID	String		
Timeblock	String	nothing, INTRADAY, EXPOST and GOPACS	
Start/ Since	Integer		EPOCH time or ISO-8601
End/ Until	Integer		EPOCH time or ISO-8601
Filter (Reporting API)	String	nothing, CREATION or TRANSACTION	This can only be used if start and end are not empty
Filter (Trades API)	String	nothing, STARTED, END or EXECUTED	This can only be used if start and end are not empty

Id	String		The participant Id used to get the information from a specific participant
Cursor	String	Nothing	The id which can be used to collect the next page of data
Count	Integer	Between 1 and 100. Default is 100	The amount of data you would like to get.

Table 2 - API parameters

Note: If the option is nothing the parameter must be empty.

API pagination

The Order, Reporting, Wallet and Trading API make use of pagination. This means that the data is limited. The reason for this, is because these API can give you a lot of data and therefore give a timeout. To solve this, we have limited the amount of data you are able to retrieve from the application. This amount can be set with the new parameter called count. The data is sorted on time, this means that the latest entry is the first in the entry in the list.
If you would like to retrieve more data, you can use the cursor parameter with the nextCursor value. This will give the next amount of data back.

2.1 On Hold Order API

The on-hold orders API provides you the possibility to hold orders before they are going to be traded. You can hold your orders and once the orders are on hold they are not going to be traded with the orderbook.

2.1.1 GET

This will return all the orders which are currently on-hold.

2.1.2 GET /{orderId}

This will return one on-hold order

2.1.3 POST

In order to create an on-hold order and normal order should exist first. To convert an order using the API you need to provide a list with the normal order Ids. This will look as following.

```
["id1", "id2", "id3', …]
```

The request should be sent as application/json. The request should look like this:

```
POST https://acc-trading.etpa.nl/public-api/1.0/electricity/on-hold-orders
accept: application/json
api_key: --------------------
Content-Type: application/json

[
  "13106fa2-1e23-404d-a3e4-fc45208f5eda"
]
```

2.1.4 DELETE

The request allows you to delete the on-hold order.

2.2 Order API V2.0

The following things are changed from version 1:

- You will get an error code and an error message when the order is incorrect.
- You will get an OrderID back from the post request to check the status of your order.

2.2.1 GET

With a get request you are able to get all the orders that are currently in the orderbook. With this request you are able to add two additional parameters. These are:

My and Timeblock (See Table 2)

2.2.2 GET /{OrderID}

With the Get/{id} you can get a specific order from the orderbook. The id is the id of the order.

2.2.3 POST

With the Post request you are able to send an order to the orderbook.

When you want to match an opposing order in the orderbook, you create a new order with the same price, time period and volume (or partial volume if needed) to match with it. The matching engine will match your order with the opposing order in the orderbook in the back-end.

When posting an order, you will need to use the participantId. This id can be retrieved from the user API.

When you post an order you will get an OrderId back from the system to check to current status of the Order. The OrderId can be used in the order status API.

An order will look like this:

```
{
  "allowedToBeUsedForIdcons": false,
  "customExpirationTime": 1465682400000,
  "ean": 871685920001768800,
  "end": 1465682400000 (epoch) or 2019-03-04T05:30:01+00:00 (ISO-8601),
  "metadata": {
    "key1": "value1",
    "key2": "value2"
  },
  "orderType": "BUY / SELL",
  "participantId": "292c3db8-3ee1-4999-bbed-d579225d1596",
  "price": 35,
  "quantity": 2,
  "start": 1465596000000 (epoch) or 2019-03-04T05:45:01+00:00 (ISO-8601),
  "timeblock": "INTRADAY/ EXPOST/ GOPACS"
}
```

NOTE: End/ Start times are in Epoch time (milliseconds). You can generate an epoch time from here: https://www.epochconverter.com/

NOTE: allowedToBeUsedForIdcons can only be used when the EAN is correct and the participant is able to create orders for IDCONS.

NOTE: customExpirationTime can't be used for EXPOST orders

See Appendix A for more information about the fields

2.2.4 PUT /{OrderID}

With the PUT request you can edit a specific order. The id is the id of the order you would like to edit. You also have to add the order you have edited.

2.2.5 DELETE /{OrderID}

This request will delete a specific order. To do this you will need to id of order you would like to delete.

2.3 Order Status API

The Order status API will give the current status or the history of the status of the order. The Order status API can only be accessed by people who have trading rights.

The Order Status API can be accessed by **`../orders/status`**

2.3.1 GET /{OrderID}

This GET request will give you the history of the order status from a specific order. It can be accessed by **`/{OrderID}`**

The different statuses are:

Created: Order has been created
Updated: Order has been updated (happened when a partial matched happens)
Completed: Order has fully matched
Cancelled: Order has been cancelled
Failed: Order is invalid
Matched: The order is matched with an Idcons order.

Reason: The reason why the order has failed/ cancelled or the it will tell you the remaining quantity.

When the order is partial matched the order will still have the status updated.

Scenarios:

Scenario 1


```
{
      "status": "CREATED",
      "reason": null,
      "createdTime": 1604938483028
    },
    {
      "status": "MATCHED",
      "reason": null,
      "createdTime": 1604938619459
    },
    {
      "status": "UPDATED",
      "reason": "Order updated because of a partial match. The remaining
quantity is : [20.0]",
      "createdTime": 1604938619962
    }
```


This indicates that the order is created then matched with idcons and after the trade a new order is created with the remaining capacity.

Scenario 2:


```
{
      "status": "CREATED",
      "reason": null,
      "createdTime": 1604578894789
    },
    {
      "status": "MATCHED",
      "reason": null,
      "createdTime": 1604578910875
    },
    {
      "status": "COMPLETED",
      "reason": null,
      "createdTime": 1604578911301
    }
```


This indicates that the order is created then fully matched with idcons. So no remaining capacity is left.

Scenario 3:


```
{
      "status": "CREATED",
      "reason": null,
      "createdTime": 1604939274281
    },
    {
      "status": "UPDATED",
      "reason": "Order updated because of a partial match. The remaining
quantity is : [1]",
      "createdTime": 1604939275526
    }
```


This indicates that the order is created and then partially matched.

Scenario 4:


```
{
      "status": "CREATED",
      "reason": null,
      "createdTime": 1604939218739
    },
    {
      "status": "COMPLETED",
      "reason": null,
      "createdTime": 1604939220373
    }
```


This indicates that the order is created and then fully matched.

2.3.2 GET /{OrderID}/current

This GET request will give you the status of the Order. It can be accessed by **`/{OrderID}/status`**

2.4 Platform Status API

URL: `/1.0/announcement/platform-status`

The platform Status API will give you information about the status of the ETPA Platform

2.4.1 GET /announcement

This URL will give you all the announcements of the application

2.4.2 GET /announcement/{id}

This URL will give you an announcement based on an ID.

2.4.3 GET /announcement/getActiveAnnouncements

This URL will give you all the active announcements.

2.5 Reporting API V2.0

URL: `2.0/electricity/reporting/order-trades`

The reporting API is used for reporting purposes. The reporting API can be used by users who have trade, wallet or reporting access.

2.5.1 GET Orders-trades

Within the Reporting API there are two options. The first option is to get all the orders and trades. The endpoint of this is: The endpoint of this is: ..**`/order-trades..`**

This API makes use of Rate limiting. Please see Appendix B for the explanation of rate limiting.

With this request you are able to add two additional parameters. These are:

Start, End, Filter (Reporting) count and cursor (see Table 2*)*

2.5.2 GET Orders-trades/{tradeId}

This URL of the tradeAPI will only give you the trades based on the traded.

2.5.3 GET Pv-party-participant

The second option within the reporting API is to retrieve all the trades from the participants from the pv-party. This enable pv-parties to get the trade information of their connected pv-party clients, no order information is given to the pv-party. The connection of participants to a PV-party is done by the ETPA admin in our system. The endpoint of this is: **`../pv-party-participants.`**

Start and End (see Table 2*)*

2.6 Trade API V2.0

The trade API can only be used by participants who have trading rights. The API can be accessed from ..**`/public-api/2.0/electricity/trades.`**

2.6.1 GET Trades

In the new Trade API we have introduced pagination. The reason is that when you apply no filters your request won't be time-out and you won't be overloaded with data.

When you don't provide any parameters, the application will return with the last 100 trades and give you the option to get the next 100 trades. To do this you use the cursor parameter with nextCursor value that has been given to you from the response.

There are 2 parameters added in the trade API. These are: cursor and count (see Table 2).

2.6.2 GET trades/{tradeID}

This API will give you the trade back from the traded.

2.6.3 GET trades/recent

The recent trades will only your trades back from the last 2 days. The participantId is only useful for brokers.
The pagination works slightly different than the other APIs. It will return the first 1000 results and after that you can add the page number as a parameter to get the remaining results.

2.7 User API

URL: `2.0/electricity/users`

The user API is the API which will give you all the information about an individual and all the information about the participants. The User API can be accessed by Trade, Report and Wallet users. The endpoint of the user API is as follows: **`../users.`**

2.7.1 GET Individual

This request will give you the information about you as an individual. To get the information about the individual add **`../individual`** to the endpoint of the API.

2.7.2 GET Participants

This request will give you the information of the participants that are representing you. To retrieve this information, you will need to add the **`../participants`** to the endpoint of the User API.

From this response you will get an id. This id is your participantId.

2.8 Wallet API V2.0

URL: `2.0/electricity/wallets`

The Wallet API provides wallet data about a participant and it will give you the transactions. The Wallet can be used by participant who has reporting, trade or wallet access. The endpoint of the wallet API is: **`../wallets.`**

2.7.1 GET Balance

This endpoint will give you the balance in json format.

2.7.2 GET Transactions

This get request will return all the transactions that have been made from a participant. The endpoint for this request is: `../transactions/.` The transactions call makes use of pagination. Therefore, the data is limited to 100. For more information about pagination see this explanation.

2.9 Status code

The API can give you multiple status code. The following status code are most common:

Code	Explanation
200	The request has been processed successfully
204	The request has been processed correctly, but there is no data available
400	The request is invalid
403	You are either not allowed to do it or the IP you are using is not known at ETPA

3. Server Sent Events (SSE)

We have introduced SSE as a replacement for the current websockets. Server Sent Events allows you to immediately receive updates from the server. Once you are connected you will be able to receive all the updates from the different channels you are subscribed to. The SSE will also send you in some cases an initial state. This is to reduce amount of API request.

3.1 Connecting to the SSE

For connecting to the SSE you need to identify yourself with the API Key. This API Key is placed in the header of the request. Both acceptance and production are different environments, so when changing environments, you need to change both the URL and API Key. If you have trouble accessing the SSE endpoint, please send an e-mail to support@etpa.nl for an example script of connecting to the SSE channels.

- Endpoint URL: `https//:[acc-]trading.etpa.nl/public-sse/*`
- API header: `api_key`
- Useful link: https://javascript.info/server-sent-events

3.2 Announcements

URL: `/announcements`

For the announcements we have set an initial state to retrieve all the current ACTIVE announcements.
For the Announcement SSE you will receives a message for every newly created or deleted and/or expired announcement in the application. To make a distinction between created and deleted announcements, the “active” field from the message can be used.

New announcement example:

```
[{
  "id": "147f6be5-a0bf-461c-9423-08988e1f47a5",
  "message": "Test announcement",
  "created": 1660913263900,
  "start": 1660913252399,
  "end": 1661518052399,
  "active": true
}]
```

Deleted announcement example

```
[{
  "id": "147f6be5-a0bf-461c-9423-08988e1f47a5",
  "message": null,
  "created": null,
  "start": null,
  "end": null,
```

```
  "active": false
}
]
```

3.3 Orderbook

For the orderbook channels (ex-post, intraday and GOPACS) we have set an initial state. The initial state will send you all the orders which are currently in the orderbook. The initial state message will be one message containing all the orders which are currently in the orderbook. After the initial state is send it will automatically send you all the new updates of the orderbook.

The differentiation between newly created and deleted orders can be done by using the field: 'type'.

- INFO: The order has been created
- WARNING: The order has been cancelled
- REFRESH: The order has been updated due to partial match

NOTE: Updating an order will result in one order deleted (WARNING) and one order created message (INFO). When the order is updated the orderId changes. You can keep track of the order with the front-end ID.

You will automatically receive all the necessary information. Some fields will be automatically masked. These fields are:

- participantId
- ean
- allowedToBeUsedForIdcons
- individualFullName
- individualId
- metadata.

3.3.1 Intraday orderbook

URL: `/intraday-orderbook`

New intraday order example:

```
[
  {
    id: 'd369b9cd-dc25-4a8f-b13f-82d9fb97d742',
    type: 'INFO',
    order: {
      id: 'd369b9cd-dc25-4a8f-b13f-82d9fb97d742',
      frontendId: '0830d163-e74c-4a6d-aff7-1b308618bc05',
      price: 1,
      quantity: 1,
      product: 'ELECTRICITY',
      timeblock: 'INTRADAY',
      type: 'BUY',
      start: 1679050800000,
```

```
      end: 1679054400000,
      participantId: 'd348e1d2-64e6-49c4-97b7-10f863c8c1b3',
      created: 1679045864264,
      priority: 1679045864264,
      ean: '',
      allowedToBeUsedForIdcons: false,
      individualFullName: 'Daniël Otter',
      individualId: 'd01a03ad-caa1-4500-807e-e1ee337fba5a',
      originalQuantity: 1,
      customExpirationTime: 1679049900000,
      metadata: null
    }
  }
]
```

Delete intraday order example:

```
[
  {
    id: 'd369b9cd-dc25-4a8f-b13f-82d9fb97d742',
    type: 'WARNING',
    order: {
      id: 'd369b9cd-dc25-4a8f-b13f-82d9fb97d742',
      frontendId: '0830d163-e74c-4a6d-aff7-1b308618bc05',
      price: 1,
      quantity: 1,
      product: 'ELECTRICITY',
      timeblock: 'INTRADAY',
      type: 'BUY',
      start: 1679050800000,
      end: 1679054400000,
      participantId: 'd348e1d2-64e6-49c4-97b7-10f863c8c1b3',
      created: 1679045864264,
      priority: 1679045864264,
      ean: '',
      allowedToBeUsedForIdcons: false,
      individualFullName: 'Daniël Otter',
      individualId: 'd01a03ad-caa1-4500-807e-e1ee337fba5a',
      originalQuantity: 1,
      customExpirationTime: 1679049900000,
      metadata: null
    }
  }
]
```

3.3.2 Ex-post orderbook

URL: `/expost-orderbook`

This channel can only be accessed when the participant has ex-post rights enabled.

New ex-post order example:

```
[
  {
    id: '66455162-c497-4ca7-b869-bc330271ddee',
    type: 'INFO',
    order: {
      id: '66455162-c497-4ca7-b869-bc330271ddee',
      frontendId: '8f0587ba-26d4-41eb-8e7e-1ba410d24a4a',
      price: 1,
```

```
      quantity: 1,
      product: 'ELECTRICITY',
      timeblock: 'EXPOST',
      type: 'BUY',
      start: 1679041800000,
      end: 1679042700000,
      participantId: 'd348e1d2-64e6-49c4-97b7-10f863c8c1b3',
      created: 1679046210503,
      priority: 1679046210502,
      ean: null,
      allowedToBeUsedForIdcons: false,
      individualFullName: 'Daniël Otter',
      individualId: 'd01a03ad-caa1-4500-807e-e1ee337fba5a',
      originalQuantity: 1,
      customExpirationTime: null,
      metadata: null
    }
  }
]
```

Ex-post order deleted message:

```
[
  {
    id: '66455162-c497-4ca7-b869-bc330271ddee',
    type: 'WARNING',
    order: {
      id: '66455162-c497-4ca7-b869-bc330271ddee',
      frontendId: '8f0587ba-26d4-41eb-8e7e-1ba410d24a4a',
      price: 1,
      quantity: 1,
      product: 'ELECTRICITY',
      timeblock: 'EXPOST',
      type: 'BUY',
      start: 1679041800000,
      end: 1679042700000,
      participantId: 'd348e1d2-64e6-49c4-97b7-10f863c8c1b3',
      created: 1679046210503,
      priority: 1679046210502,
      ean: null,
      allowedToBeUsedForIdcons: false,
      individualFullName: 'Daniël Otter',
      individualId: 'd01a03ad-caa1-4500-807e-e1ee337fba5a',
      originalQuantity: 1,
      customExpirationTime: null,
      metadata: null
    }
  }
]
```

3.3.3 GOPACS orderbook

URL: `/gopacs-orderbook`

This channel can only be accessed when the participant has GOPACS rights enabled.
New GOPACS order example:

```
[
{
```

```
  "id": "4ce41e23-1b7e-4276-995a-c75f7c4b1855",
  "type": 'INFO',
  "order": {
    "id": "4ce41e23-1b7e-4276-995a-c75f7c4b1855",
    "frontendId": "da84cae7-e078-4bb3-a9b8-958fefdfa791",
    "price": 1,
    "quantity": 1,
    "product": "ELECTRICITY",
    "timeblock": "GOPACS",
    "type": "BUY",
    "start: " 1672149600000,
    "end": 1672153200000,
    "participantId": "b548d968-fc4b-4c87-a23e-6834156b6a36",
    "created": 1672148235169,
    "priority": 1672148235168,
    "ean": "1111111111111111",
    "allowedToBeUsedForIdcons": true,
    "individualFullName": "Owner of participant Zero",
    "individualId": "4b2ba585-f256-4d3a-8387-4315e805de5b",
    "originalQuantity": 1,
    "customExpirationTime": 1672148700000,
    "metadata": null
  }
}
]
```


Delete GOPACS order example:


```
[
{
  "id": "4ce41e23-1b7e-4276-995a-c75f7c4b1855",
  "type": 'WARNING',
  "order": {
    "id": "4ce41e23-1b7e-4276-995a-c75f7c4b1855",
    "frontendId": "da84cae7-e078-4bb3-a9b8-958fefdfa791",
    "price": 1,
    "quantity": 1,
    "product": "ELECTRICITY",
    "timeblock": "GOPACS",
    "type": "BUY",
    "start: " 1672149600000,
    "end": 1672153200000,
    "participantId": "b548d968-fc4b-4c87-a23e-6834156b6a36",
    "created": 1672148235169,
    "priority": 1672148235168,
    "ean": "1111111111111111",
    "allowedToBeUsedForIdcons": true,
    "individualFullName": "Owner of participant Zero",
    "individualId": "4b2ba585-f256-4d3a-8387-4315e805de5b",
    "originalQuantity": 1,
    "customExpirationTime": 1672148700000,
    "metadata": null
  }
}
]
```


3.4 Trades

For the trades channels we also set an initial state. This is based on the start of the day (00:00). The initial state will be send as one message in list. After this you will automatically retrieve all the new trades.

Some fields will be masked based on security reasons. These fields are:

Buyer-sensitive information
- buyerId
- buyerEan
- orderIdBuy
- buyOrderMetadata.

Seller-sensitive information:
- sellerId
- sellerEan
- orderIdSell
- sellOrderMetadata
-

Congestion-sensitive information:
- congestionId
- comment.

3.4.1 Intraday trades

URL: `/intraday-trades`

Example intraday trade:


```
[
{
  "id": "63ac7e4bf2e6cf733cfeb94d",
  "type": "INFO",
  "trade": {
    "id": "63ac7e4bf2e6cf733cfeb94d",
    "tradeId": "02e3c81a-96c3-42be-896c-d96ec3ba9ae0",
    "productType": "ELECTRICITY",
    "timeblock": "INTRADAY",
    "buyerId": "5fcb140c-624c-4499-b9f0-8441d51bf8a1",
    "sellerId": "",
    "buyerGridOperator": false,
    "sellerGridOperator": false,
    "orderIdBuy": "3ea7194e-e171-49e1-8bcb-bea05a407dc2",
    "orderIdSell": "",
    "buyerEan": "",
    "sellerEan": "",
    "buyOrderMetadata": null,
    "sellOrderMetadata": {},
    "quantity": 1,
    "start": 1672254000000,
    "end": 1672257600000,
    "executed": 1672248906346,
    "price": 1,
    "type": "intra-day",
```

```
    "comment": "",
    "congestionId": "",
    "duration": 1,
    "isCongestionTrade": false
  }
}
]
```

3.4.2 Ex-post trades

URL: `/expost-trades`

This channel can only be accessed when the participant has ex-post rights enabled.

```
{
  id: '63d8f9264801b70bb77af8bd',
  type: 'INFO',
  trade: {
    id: '63d8f9264801b70bb77af8bd',
    tradeId: 'ada69bb6-9a01-4c5f-a061-287760472a65',
    productType: 'ELECTRICITY',
    timeblock: 'EXPOST',
    buyerId: 'd348e1d2-64e6-49c4-97b7-10f863c8c1b3',
    sellerId: '',
    buyerGridOperator: false,
    sellerGridOperator: false,
    orderIdBuy: '726a0371-2498-4c08-9c96-8d1a5653fff4',
    orderIdSell: '',
    buyerEan: null,
    sellerEan: '',
    buyOrderMetadata: {},
    sellOrderMetadata: {},
    quantity: 10,
    start: 1675135800000,
    end: 1675136700000,
    executed: 1675163942374,
    price: 14,
    type: 'ex-post',
    comment: '',
    congestionId: '',
    duration: 0.25,
    isCongestionTrade: false
  }
}
```

3.4.3 GOPACS Trades

URL: `/gopacs-trades`

This channel can only be accessed when the participant has GOPACS rights enabled.

```
{
  id: '63d8f9264801b70bb77af8bd',
  type: 'INFO',
  trade: {
    id: '63d8f9264801b70bb77af8bd',
    tradeId: 'ada69bb6-9a01-4c5f-a061-287760472a65',
```

```
    productType: 'ELECTRICITY',
    timeblock: 'GOPACS',
    buyerId: 'd348e1d2-64e6-49c4-97b7-10f863c8c1b3',
    sellerId: '',
    buyerGridOperator: false,
    sellerGridOperator: false,
    orderIdBuy: '726a0371-2498-4c08-9c96-8d1a5653fff4',
    orderIdSell: '',
    buyerEan: null,
    sellerEan: '',
    buyOrderMetadata: {},
    sellOrderMetadata: {},
    quantity: 10,
    start: 1675135800000,
    end: 1675136700000,
    executed: 1675163942374,
    price: 14,
    type: 'gopacs',
    comment: '',
    congestionId: '',
    duration: 0.25,
    isCongestionTrade: true
  }
}
```

4. Websocket DEPRECATED from 01-02-2023

At ETPA we provide also WebSocket. This means you can automatically receive information about general information, orders and trades. The information about the WebSocket on acceptance can be found here as well: https://acc-trading.etpa.nl/#/socketInfo or for production: https://trading.etpa.nl/#/socketInfo.

Implementing the normal GET orderbook calls is still recommended next to this WebSocket interface, due to possible internet connections errors (WebSockets message might therefore not be received at the client side). This will lead to a difference between the current known state of the orderbook, held by your system based on the WebSocket communication, and the real known state of the orderbook at the ETPA side. Some business logics based on these orderbook GET call will ensure that the systems stay in sync with each other at all times.

To use the WebSocket on the acceptance environment you have to subscribe to the following URL: **wss://acc-trading.etpa.nl/public-api/1.0/electricity/websocket**

To use the WebSocket on the production environment you have to subscribe to the following URL: **wss://trading.etpa.nl/public-api/1.0/electricity/websocket**

4.1 Connecting to the WebSocket server

For both the acceptance and production environment you will need to add your API key into the header of the request. When the connection with the API Key is established you will need to send a message to the WebSocket server to make sure you are connected. The WebSocket message must contain the following message:

```
CONNECT\naccept-version:1.1,1.0\n\n\u0000
```

When you have made a successful connection, you can subscribe to a specific channel. You can do this by sending a message to the WebSocket server. This needs to be the following message:

```
SUBSCRIBE\nid: Sub - 0\ndestination:/{channel}\n\n\u0000.
```

Now you can receive messages from a specific channel.

4.2 General

Within the general information there are four different channels. These are: **/refresh**, **/queue/public**, **/announcements** and **/participant/(your participantID)/queue**

The **/refresh** channel lets the user know if the GUI needs to be refreshed.

The **/queue/public** channel lets the user know about all the information about orders and trades.

The **`/announcements`** channel lets the user know if there are any announcements.

The /**`participant/{your participantID}/queue`** channel is specific channel for participants. Participant specific orders and trades will be sent to this channel. This channel will give you the orderId (Id) and tradeId (when traded). These Id can be used in the trades and reporting API

4.3 Orders

There are four channels which sends information about orders. These are: **`/state`**, **`/intradayorderbook`** and **`/expostorderbook.`**

The **`/state`** channel lets the user know if creating or modifying orders has been enabled or disabled. This channel will return either true or false.

The **`/standardorderbook`** channel lets the user know when a standard order has been created, cancelled or edited. **-> Is deprecated**

The **`/intradayorderbook`** channel lets the user know when an intraday order has been created, cancelled or edited.

The **`/expostorderbook`** channel lets the user know when an ex-post order has been created, cancelled or edited.

An order in the WebSocket will look like this:

```
{
 "id":"39d34519-6b3a-4a57-b3f8-ef42567d5593",
 "type":"INFO",
 "action":"ORDERS",
 "time":1480430752452,
 "message":null,
 "title":null,
 "order": {
     "id":"39d34519-6b3a-4a57-b3f8-ef42567d5593",
     "price":3,
     "quantity":1,
     "product":"ELECTRICITY",
     "timeblock":"BASELOAD",
     "type":"BUY",
     "start":1480460400000,
     "end":1480546800000,
     "participantId":"02866e9b-9359-4b68-b409-9473404b125e",
     "created":1480340400000,
     "priority":1480340400000
   }
 }
```

Code block 1 - Order from WebSocket

4.4 Trades

There is only one channel which sends information about trades. This is the ***/trades*** channel. It will let the user know every time a trade is made. A trade in the WebSocket will look like this:

```
{
 "action":"TRADES",
 "id": "c456226f-5dba-4f40-bcde-7b3b9d027fbb",
 "message":null,
 "order":null,
 "time":1480430752452,
 "title":null,
 "trade": {
     "end":1480546800000,
     "excecuted":1480430752452
     "price":"3",
     "quantity":1,
     "start":1480460400000,
     "duration":24,
     "timeblock":"BASELOAD"
   },
 "type":"INFO",
 }
```

Code block 2 - Trade from WebSocket

4.5 Websocket Status

In the websocket you can get 3 different statuses. These are:

Status	Order is:
INFO:	Created
WARNING:	Deleted, Cancelled, Fully Matched
REFRESH	Updated due to partial match

Example for updating an order:

1. Order manually updated (not due to partial match)

Let's say you create an order with 10MW and €10,- you will get an INFO (with id **x** and front-end Id **x**). Then you will change the capacity to 5 MW. You will receive a WARNING (with id **x** and front-end id **x**)**.** You will finally receive a new INFO message (with id **y** and front-end **x**). The front-end id doesn't change unless the order is fully deleted.

5. FAQ (Frequently asked questions)

1. Do you also need an API Key for the WebSocket?

Yes, the API Key is used for authentication purposes.

2. Getting a 200 error while connecting to the correct WebSocket URL.

When you are using a different library, which doesn't support SockJS you'll need to add an extra /websocket to your URL. Now you will let the application know that you want to connect to WebSocket instead of the SockJS WebSocket connection.

3. Connected to the websocket server, but not getting any response.

Before subscribing to specific channels, you will need to make sure that you are connected. To this you will need to send a message to the WebSocket server with the following message:

```
Websocket.send("CONNECT\naccept-version:1.1,1.0\n\n\u0000").
```

You will get a response from the WebSocket server with the message that you are connected.

Appendix A

Fields used for creating an order

Parameter	Type	Description	Required
allowedToBeUsedForIdcons	boolean	If the order is allowed to be used for idcons purposes and only possibly when EAN is correct and the participant is allowed to create orders for IDCONS.	NO
customExpirationTime	integer	Option for custom expiration time. The default is always 15 minutes before delivery. The time should always be in quarters (so only 00,15,30,45). This parameter should be defined in epoch time in milliseconds. This option **cannot** be used when creating an ex-post order	NO
ean	integer	The ean code of the order	NO
end	integer	The end time of the order in epoch time in milliseconds or in ISO-8601 date format.	YES
metadata	Hashmap with key and value	Additional information that can be used for internal administration	NO
ordertype	String	The type of the order: BUY or SELL	YES
participantId	String	The id of the participant	YES
price	integer	The price of the order	YES
quantity	integer	The quantity of the order	YES
start	integer	The end time of the order in epoch time in milliseconds or in ISO-8601 date format.	YES
timeblock	String	The timeblock of the order: INTRADAY, EXPOST	YES

Appendix B

Rate limiting explained:

You will have a maximum of 30 request available and after every 2 seconds you will get another request, so let's say you do a request then you will 29 requests remaining and after 2 seconds another request will be added, so you will have 30 requests again. The total of request cannot be more than 30. If you do exceed the requests, you will get the following error:

```
{ "status": 429, "error": "Too Many Requests", "message": "You have
exhausted your API Request Quota" }
```

In the header there will be 2 keys which will tell you how many requests you have left and how long you will have to wait before you can do another request these keys are called:

`X-Rate-Limit-Retry-After-Seconds`

and

`X-Rate-Limit-Remaining.`

If you have any questions, please send a mail to daniel.bronder@etpa.nl