

How to use Data Sharing API

V4

Date	2023/03/24
Author	Gopi Ande
Verified	
Revision	4



Table of Contents

1. Requirements	3
2. Authentication	3
2.1. /auth [POST]	3
3. EndPoints Descriptions	4
3.1. /feedback [POST]	4
3.2. /graphId [GET]	6
3.3. /entityId [GET]	7
3.4. /asset [GET]	8
3.5. /amplitude [POST]	9
3.6. /analytics/notification [POST]	10
3.7. /analytics/graph [POST]	10
3.8. /analytics/graph [GET]	12
3.9. /analytics/health [POST]	13
4. Limitations	15



1. Requirements

- You need an HTTP client.
- The base url for all requests is <https://nanoprecisedataservices.com/data-sharing/api/v1>. you need to append the end point to this url to get the data
- You need dashboard login credentials

2. Authentication

2.1. /auth [POST]

1. Make a request to /auth endpoint using the username and password given to you to get the bearer token. The credentials for this are **same as our dashboard login credentials**.
2. You will get a bearer token along with expiry time of that token. So, you can make as multiple requests using the same token before it expires. After that you have to get the new token to request the data again.
3. Example (Requesting auth token from postman):

DataSharing / Auth

POST https://nanoprecisedataservices.com/data-sharing/api/v1/auth

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 "username": "<dashboard-username>",
2
3 "password": "<dashboard-password>"
4
```

Body Cookies Headers (14) Test Results Status: 200 OK Time: 248 ms Size: 1.28 KB Save Response

Pretty Raw Preview Visualize JSON

```
1
2 "expiresAt": "2022-11-05T15:25:05.417000+00:00",
3 "token": "eyJhbGciOiJIUzI1NiJ9.eyJ1IjoiZS1jYmFtZtZSI6IjRFTeslLCJwYXNzIjoi2k13Ka7r9CBkCmgkQpnh1utS670KQ"
4
```



3. EndPoints Descriptions

3.1. /feedback [POST]

To send feedback to Nanoprecise regarding the given notification shared through notification endpoint to improve the fault detection

Request body should contain the following parameters:

Parameter	Description	Type	enum
feedbackCode	To differentiate between feedback for different companies. Each company will receive a feedback code offline to use here	integer	
acknowledgementId	Integer value between [0,1,2] 0 New, 1 Acknowledged, 2 Closed/Resolved type: integer	integer	[1, 2, 3]
notificationId	Associating the feedback to specific notification	string	
workOrderType	Once value from Set of work order types. Types: ['Preventive', 'Damage', 'Corrective', 'Safety', 'Upgrade', 'Electrical', 'Project', 'Inspection', 'Meter reading', 'Other']	string	
workOrderStatusId	Integer value between [0,1,2] 0 Open, 1 in progress, 2 closed	integer	[1, 2, 3]
workOrderScheduleTimestamp	When work order is scheduled to happen	integer	
workOrderCreationTimestamp	When work order was created	integer	
workOrderdescription	Description of work order	string	



Example:

POST ▼ https://nanoprecisedataservices.com/data-sharing/api/v1/feedback

Params Authorization ● Headers (11) **Body** ● Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON** ▼

```
1  |
2  | ..... "notificationId": "638784b5-c99a-4499-9b2b-d5b797696784",
3  | ..... "acknowledgementId": 1,
4  | ..... "feedbackCode": 100,
5  | ..... "workOrderType": "Preventive",
6  | ..... "workOrderStatusId": 1,
7  | ..... "workOrderdescription": "Test Work Order",
8  | ..... "workOrderCreationTimestamp": 1666510767,
9  | ..... "workOrderScheduleTimestamp": 1666510767
10 |
```

Body Cookies Headers (13) Test Results

Pretty Raw Preview Visualize **JSON** ▼

```
1  |
2  | "message": "request was received successfully",
3  | "status": "successful"
4  |
```



3.2. /graphId [GET]

It provides the Id to use for getting the data from /analytics/graph endpoint. The returned object id field's value needs to be used as graphId in /analytics/graph. You shouldn't pass anything in the request body for this request.

Example:

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: https://nanoprecisedataservices.com/data-sharing/api/v1/graphId
- Body: none
- Response Body (JSON):

```
1 {
2   "battery": {
3     "id": "battery",
4     "type": "float",
5     "unit": "Battery (V)",
6     "value": "Battery"
7   },
8   "flux": {
9     "rms": {
10      "id": "flux-rms",
11      "type": "float",
12      "unit": "mT (milli Tesla)",
13      "value": "RMS"
14    }
15  },
16  "humidity": {
17    "id": "humidity",
18    "type": "float",
19    "unit": "%rH",
20    "value": "Humidity"
21  },
22  "sound": {
23    "rms": {
24      "id": "sound-rms",
25      "type": "float",
26      "unit": "Sound (dB)",
27      "value": "RMS"
28    }
29  },
30  "speed": {
31    "id": "speed",
32    "type": "float",
33    "unit": "Speed (RPM)",
34    "value": "Speed"
35  },
36  "temperature": {
```



3.3. /entityId [GET]

It provides the Id to use for getting the data from /analytics/health endpoint. The returned object id field's value needs to be used as entityTypeId in /analytics/health. You shouldn't pass anything in the request body for this request.

Example:

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: https://nanoprecisedataservices.com/data-sharing/api/v1/entityId
- Body: none (selected)
- Response Body (JSON):

```
1  {
2    "company": {
3      "id": "company",
4      "value": "Company Health"
5    },
6    "component": {
7      "id": "component",
8      "value": "Component Health"
9    },
10   "equipment": {
11     "id": "equipment",
12     "value": "Equipment Health"
13   },
14   "machine": {
15     "id": "machine",
16     "value": "Machine Health"
17   },
18   "plant": {
19     "id": "plant",
20     "value": "Plant Health"
21   },
22   "version": 1
23 }
```



3.4. /asset [GET]

Returns an array of JSON objects. Each return object has name fields for company, plant, equipment and component along with tagId. tagId is a unique identifier for each component. You shouldn't pass anything in the request body for this request.

If the assets in Nanoprecise system are linked to customer's assets, then you also will get a key called `externalId` and the value will be the assetId of the asset in customer's system. Note that this id has to be setup in the Nanoprecise system through configuration portal or the DIY mobile app.

The results of this endpoint are required for all `/analytics/` endpoints where tagIds or equipment, component, plantIds are used in the request body.

Example:

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: `https://nanoprecisedataservices.com/data-sharing/api/v1/asset`
- Body: none
- Response: A JSON array of three objects. The first two objects are identical, and the third is partially visible.

```
1 [
2   {
3     "companyCode": "ATCE",
4     "companyId": "ATCE",
5     "companyName": "Advanced Technology Center",
6     "componentId": 80000,
7     "componentName": "Motor DE",
8     "equipmentId": 8347,
9     "equipmentName": "Fan Supply",
10    "machineId": 20000000896,
11    "machineName": "Fan Supply",
12    "plantId": 182,
13    "plantName": "DEMO dashboard",
14    "sensorId": null,
15    "tagId": "ATCEZDXA08347SVERR80000W2Z1"
16  },
17  {
18    "companyCode": "ATCE",
19    "companyId": "ATCE",
20    "companyName": "Advanced Technology Center",
21    "componentId": 80001,
22    "componentName": "Motor NDE",
23    "equipmentId": 8347,
24    "equipmentName": "Fan Supply",
25    "machineId": 20000000896,
26    "machineName": "Fan Supply",
27    "plantId": 182,
28    "plantName": "DEMO dashboard",
29    "sensorId": "30000c2a690be407",
30    "tagId": "ATCEZDXA08347SVERR80001IP5S"
31  },
32  {
33    "companyCode": "ATCE",
34    "companyId": "ATCE",
35    "companyName": "Advanced Technology Center",
```




3.5. /amplitude [POST]

Returns the a JSON object with tagIds as properties and an array of active amplitude keys as values. This gives the types of amplitude data enabled for each tagId. The request body must have **tagIdList** field with a list of tag values as value as shown in the below screenshot.

Example:

The screenshot shows a REST client interface for a POST request to `https://nanoprecisedataservices.com/data-sharing/api/v1/amplitude`. The request body is a JSON object with a `tagIdList` field containing two tag IDs: `"ATCEZDXA08347SVERR8000W2Z1"` and `"ATCEZDXA08347SVERR80001IP5S"`. The response is a JSON object with two properties corresponding to the tag IDs, each containing an array of vibration amplitude keys.

```
1 {
2   "tagIdList": ["ATCEZDXA08347SVERR8000W2Z1", "ATCEZDXA08347SVERR80001IP5S"]
3 }
```

Body Cookies Headers (13) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "ATCEZDXA08347SVERR8000W2Z1": [
3     "vibration-amplitude-bpfi_ge",
4     "vibration-amplitude-bpfo_ge",
5     "vibration-amplitude-looseness",
6     "vibration-amplitude-bpfo_acc",
7     "vibration-amplitude-bpfo",
8     "vibration-amplitude-bsf",
9     "vibration-amplitude-bpfi_acc",
10    "vibration-amplitude-bsf_ge",
11    "vibration-amplitude-bsf_acc",
12    "vibration-amplitude-bpfi"
13  ],
14  "ATCEZDXA08347SVERR80001IP5S": [
15    "vibration-amplitude-bpfi_ge",
16    "vibration-amplitude-bpfo_ge",
17    "vibration-amplitude-rotor_bar",
18    "vibration-amplitude-looseness",
19    "vibration-amplitude-stator",
20    "vibration-amplitude-bpfo_acc",
21    "vibration-amplitude-bpfo",
22    "vibration-amplitude-bsf",
23    "vibration-amplitude-bpfi_acc",
24    "vibration-amplitude-misalignment",
25    "vibration-amplitude-unbalance",
26    "vibration-amplitude-bsf_ge",
27    "vibration-amplitude-bsf_acc",
28    "vibration-amplitude-bpfi"
29  ]
30 }
```



3.6. /analytics/notification [POST]

Returns an array of notifications for each tagId in the tagIdList passed in the request body.

Required fields in the request body are tagIdList, timestampFrom, timestampTo as shown in the below example.

Example:

The screenshot shows a REST client interface with a POST request to `https://nanoprecisedataservices.com/data-sharing/api/v1/analytics/notification`. The request body is a JSON object with the following fields:

```
1 {
2   "tagIdList": ["ATCEZDXA08347SVERR80001IP55"],
3   "timestampFrom": 1666510767,
4   "timestampTo": 1668720767
5 }
```

The response is a JSON array containing one notification object. The response status is 200 OK, with a time of 824 ms and a size of 2.58 KB. The response body is shown in a pretty-printed JSON format:

```
1 [
2   {
3     "alarmValueObject": {
4       "value": 15336.43
5     },
6     "faultType": [
7       {
8         "type": "unbalance"
9       }
10    ],
11    "graphKey": [
12      {
13        "key": "vibration-amplitude-unbalance"
14      }
15    ],
16    "message": [
17      "Fault Notification:\n-----\nCompany Name: Advanced Technology Center\nPlant Name: DEMO dashboard\nEquipment Name: Fan Supply\nComponent Name: Motor NDE\nDate and Time: October 08, 2022 08:47 PM (GMT)\n\nDetected Fault(s): Unbalance\nRemaining Useful Life: 31%\nRoot Causes: \n - Unbalance\n 1. Dirt accumulation\n 2. Damaged Fan Blades\n 3. Static or dynamic unbalance\n 4. Improper installation\n 5. Process induced imbalance\n 6. Bent rotating shaft\n 7. Poor design /poor quality material\n\nSuggested Actions: \n - Unbalance\n 1. Clean rotor and check balance\n 2. Check for damaged blades\n 3. Check for the balancing of the rotating equipment\n 4. Check for any material deposit/wearout and see if it requires to do balancing\n 5. Check for flow distribution\n 6. Check shaft run-out\n 7. Check for any structural issues (resonance/looseness)\n\nLink To Dashboard: https://nanoprecisedataservices.com/components/ODM0Nw==/ZXF1aXB7W50X25hbWU=/MTgy\n\nIf you have any questions or comments regarding this email, please reply to customersupport@nanoprecisec.com email, and do not reply directly to this email"
18    ],
19    "notificationId": [
20      "638784b5-c99a-4499-9b2b-d5b797696784"
21    ],
22    "rul": [
23      15336.43
24    ],
25    "stage": [
26      3
27    ]
28  }
29 ]
```

3.7. /analytics/graph [POST]

Returns the data for a specific graph. For each graph a separate request needs to be sent. Get the graphIds from /graphId endpoint and tagIds from /asset endpoint.

Required fields in the request body are tagIdList, graphId, timestampFrom, timestampTo as shown in the below example.



Example:

POST ▼ https://nanoprecisedataservices.com/data-sharing/api/v1/analytics/graph

Params Authorization ● Headers (11) **Body ●** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON ▼**

```
1 [
2   ... "tagIdList":["ACME1696a724b2d2421783095d3fbfc9da91", "-ACME041c05efb4104064b29c08a9ebbbb556"],
3   ... "graphId":"vibration-rms-acceleration",
4   ... "timestampFrom":1666510767,
5   ... "timestampTo":1668720767
6 ]
```

Body Cookies Headers (13) Test Results

Pretty Raw Preview Visualize **JSON ▼**

```
1 [
2   {
3     "data": {
4       "x": [
5         0.008555498173744003,
6         0.012922139391076734,
7         0.008637158566876432,
8         0.008623904349474866,
9         0.008887945998913403,
10        0.009709019278851636,
11        0.008603193913493179,
12        0.00864388157684881,
13        0.008657740674768733,
14        0.0086407679400159,
15        0.008632840656947532
16      ],
17      "y": [
18        0.008436793489759906,
19        0.008681540418236983,
20        0.00846291400446809,
21        0.00845165174688458,
22        0.008704928644060412,
23        0.008869626883346476,
24        0.00842114341883828,
25        0.008428519080118905,
26        0.008450331485398173,
27        0.008414818604219169,
28        0.008405908515989964
29      ],
30      "z": [
31        0.00927696133803654,
32        0.00860701576107117
```



3.8. /analytics/graph [GET]

This returns the same results as the POST request for /analytics/graph endpoint. But the request type is different, and the request parameters are passed in the URL.

You must pass same the parameters in the URL here.

Note that, you cannot pass more than 10 tagIds in each request for the GET request type where is in POST request, you can pass up to 50 tagIds.

Example:

The screenshot shows a REST client interface for a GET request to the endpoint `/analytics/graph`. The request parameters are:

Key	Value	Description
<input checked="" type="checkbox"/> graphId	vibration-rms-acceleration	
<input checked="" type="checkbox"/> timestampFrom	1677275721	
<input checked="" type="checkbox"/> timestampTo	1679694923	
<input checked="" type="checkbox"/> tagIdList	ACMETEST95008TE51111776TES5,ACMEb...	

The response is a JSON object with a status of 200 OK, a response time of 375 ms, and a size of 2.1 KB. The response body is:

```
1 {
2   "data": {
3     "x": [
4       2.1945717793909774,
5       2.2104436951501167,
6       2.4243482928128386,
7       2.3248472210061286,
8       2.089324636741401,
9       2.1330264149214035,
10      2.172756492595426,
11      2.0707077303507058.
```



3.9. /analytics/health [POST]

To get the health status of a certain entity. Entities are different levels of a company, which includes Company, Plant, Machine (if enabled), Equipment, and Component. You can get these Ids from /asset endpoint. The following examples are given for component and equipment. You can also query similarly at plant level and machine level(if enabled)

Example: querying health data of components by sending componentId via `Ids` field and the `entityTypeId` as **component**

The screenshot shows a REST client interface with a POST request to `https://nanoprecisedataservices.com/data-sharing/api/v1/analytics/health`. The request body is a JSON object with the following structure:

```
1 {
2   "Ids": ["ATCEZDXA08347SVERR80000W2Z1", "ATCEZDXA08347SVERR80001IP5S"],
3   "entityTypeId": "component"
4 }
```

The response body is a JSON array of two objects:

```
1 {
2   "_id": "ATCEZDXA08347SVERR80000W2Z1",
3   "faultType": null,
4   "healthStatus": "Healthy",
5   "lifePercentage": 82.33496425405282,
6   "rul": 496531.49,
7   "stage": 1,
8   "suggestion": null,
9   "timestamp": 1662690500,
10  "utilizationFactor": 0.9364161849710982
11 },
12 {
13  "_id": "ATCEZDXA08347SVERR80001IP5S",
14  "faultType": null,
15  "healthStatus": "No data",
16  "lifePercentage": "No data",
17  "rul": null,
18  "stage": 2,
19  "suggestion": null,
20  "timestamp": 1666212422,
21  "utilizationFactor": 0.9706774519716885
22 }
23 }
```



Example: querying health data of equipment by sending equipment Ids via `Ids` field and the `entityTypeId` as **equipment**

POST ▼ <https://nanoprecisedataservices.com/data-sharing/api/v1/analytics/health>

Params Authorization ● Headers (11) **Body ●** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON ▼**

```
1 {
2   ... "Ids": "[8347,8348]",
3   ... "entityTypeId": "equipment"
4 }
```

Body Cookies Headers (13) Test Results

Pretty Raw Preview Visualize **JSON ▼**

```
1 [
2   {
3     "_id": 8347,
4     "healthStatus": "healthy",
5     "lifePercentage": 82.33496425405282,
6     "rul": 496531.49,
7     "suggestion": null,
8     "timestamp": 1666212422
9   },
10  {
11    "_id": 8348,
12    "healthStatus": "No data",
13    "lifePercentage": "learning",
14    "rul": null,
15    "suggestion": null,
16    "timestamp": 1655376555
17  }
18 ]
```



4. Limitations

- One cannot request for more than 30 days of data at a time
- When time duration is required, starting time should not be older than 31 days ago
- “timestampFrom” in the requests wherever applicable can neither be older than 31 days nor be a future date
- Maximum number of API requests is 15000/day per each user
- When number of tags are specified, it should not be more than 50 tags in each request