# SMART SYSTEMS REST API QUERIES

## System Queries

### get Time <GET>

Usage: <http://IP:Port/api/getTime>

Description: Returns the local time on the server

## Asset Queries

### Get Asset Hierarchy <GET>

Usage: <http://IP:Port/api/getAssetHierarchy>

Description: Returns the smart systems asset hierarchy

### Get Asset Types <GET>

Usage: <http://IP:Port/api/getAssetTypes>

Description: Returns the smart systems asset types

### Get Assets <GET>

Usage: <http://IP:Port/api/getAssets>

Description: Returns the smart systems assets

### Create Asset <GET>

Usage: [http://IP:Port/api/createAsset?type={Asset Type}&name={Asset Name](http://IP:Port/api/createAsset?type=%7bAsset%20Type%7d&name=%7bAsset%20Name)}

Example: <http://IP:Port/api/createAsset?type=Pump&name=PumpABC>

Description: Creates a new asset based on Pump Type

### Rename Asset <GET>

Usage: [http://IP:Port/api/createAsset?name={Current Asset Name}&newname={New Asset Name](http://IP:Port/api/createAsset?name=%7bCurrent%20Asset%20Name%7d&newname=%7bNew%20Asset%20Name)}

Example: <http://IP:Port/api/createAsset?type=PumpABC&name=PumpXYZ>

Description: Renames the Asset

### DELETE Asset <GET>

Usage: [http://IP:Port/api/deleteAsset?name={Asset Name](http://IP:Port/api/deleteAsset?name=%7bAsset%20Name)}

Example: <http://IP:Port/api/createAsset?name=PumpXYZ>

Description: Deletes Asset

## Nameplate Queries

### Get Nameplate <GET>

Usage: [http://IP:Port/api/getNameplate?name={Asset Name](http://IP:Port/api/getNameplate?name=%7bAsset%20Name)}

Example: <http://IP:Port/api/getNameplate?name=fan>

In order to get **Global Template Nameplate** values, please pass **“GlobalTemplate”** for name. This will return default template values.

Description: Returns nameplate information of the selected equipment. Web client can only change the Value and send it back to Smart Systems using setNameplate query which is explained in the next section. The value changed by user through web client has to be between Min and Max values. If the Min or Max value is NaN, there is no need to check the corresponding limit. If both Min and Max are NaN, there is no need to check any of them. Set Nameplate already checks the limits. However, it is strongly recommended to check the values before submitting them to Smart System.

[

    {

        "Value": 5000.0,

        "Unit": "Hz",

        "Min": 1000.0,

        "Max": 32000.0,

        "Method": "",

        "Parameters": [],

        "Type": 0,

        "Calculated": **false**,

        "Enabled": **true**,

        "Assembly": "Core",

        "Name": "SR",

        "Title": "Sampling Rate"

    },

    {

        "Value": 10.0,

        "Unit": "s",

        "Min": 1.0,

        "Max": 20.0,

        "Method": "",

        "Parameters": [],

        "Type": 0,

        "Calculated": **false**,

        "Enabled": **true**,

        "Assembly": "Core",

        "Name": "Duration",

        "Title": "Duration"

    },

    {…………………………………..}]

### Set nameplate <POST>

Usage: [http://IP:Port/api/setNameplate?name={Asset Name](http://IP:Port/api/setNameplate?name=%7bAsset%20Name)}

In order to update **Global Template Nameplate** values, please pass **“GlobalTemplate”** for name. This will save default template values.

Description: Overwrites the nameplate values changed by the user. User is not allowed to change any other parameters than the value by checking the Min and Max values. User can update a single nameplate parameter by sending a jsonobject in the body of the query or alternatively send an array of nameplate values as shon in the example below.

Example: <http://IP:Port/api/setNameplate?name=fan>

* Open Postman Click "New Request"
* Set Method & URL POST http://localhost:5000/api/ setNameplate?name=fan
* Content-Type: application/json
* Add JSON Body

[

    {

        "**Value": 8000.0,**

        "Unit": "Hz",

        "Min": 1000.0,

        "Max": 32000.0,

        "Method": "",

        "Parameters": [],

        "Type": 0,

        "Calculated": **false**,

        "Enabled": **true**,

        "Assembly": "Core",

        "Name": "SR",

        "Title": "Sampling Rate"

    },

    {

        "Value": 5.0,

        "Unit": "s",

        "Min": 1.0,

        "Max": 20.0,

        "Method": "",

        "Parameters": [],

        "Type": 0,

        "Calculated": **false**,

        "Enabled": **true**,

        "Assembly": "Core",

        "Name": "Duration",

        "Title": "Duration"

    },

    {…………………………………..}]

* Click "Send"

## Parameters Queries

### Get PARAMETERS <GET>

Usage: [http://IP:Port/api/getParameters?name={Asset Name](http://IP:Port/api/getParameters?name=%7bAsset%20Name)}

Example: [http://IP:Port/api/getParameters?name=fan](http://IP:Port/api/getNameplate?name=fan)

This command is mainly used to retrieve thresholds and Acknowledgement for each asset or global template parameters. Thresholds contains 8 values.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| Down | Down | Down | Down | Up | Up | Up | Up |
| Out of Range | High/Repair | Medium/Inspect | Low/Warning | Low/Warning | Medium/Inspect | High/Repair | Out of Range |

**Test Types:**

enum TESTTYPE { NONE = 0, MAX, MIN, BOTH }

**None**: Do not test

**Max**: Test only from the upper side

**Min**: Test only from Down side

**Both**: Test from both Up and Down sides

In order to get **Global Template Parameters** values, please pass **“GlobalTemplate”** for name. This will return default template values.

Description: Returns Parameters information of the selected equipment. Web client can only change the Default Thresholds and send it back to Smart Systems using setParameters query which is explained in the next section. The Default Thresholds changed by user through web client has to be always in ascending order. If threshold value is NaN, that means this value is not being used. For example, if the Test Type is Max, all the down threshold values will be NaN. Client should not change the NaN values. Set Parameters already checks these rules. However, it is strongly recommended to check the values before submitting them to Smart System.

Please note that Out of range thresholds are implemented for future use. Therefore, Out of range thresholds will be always NaN for now.

[{

"Parameterized": true,

"Method": "dc",

"Unit": "V",

"Training": 1,

"TestType": 3,

"DefaultThresholds": ["NaN", -0.04, -0.03, -0.02, 0.02, 0.03, 0.04, "NaN"],

"ThresholdParameter": "[OP:mul]NominalVoltage",

"ExtendingParameters": "",

"Acknowledged": false,

"Description": "A DC voltage signal can lead to an increase in the DC offset of the current signal. This, in turn, raises the temperature in the core and windings, which may eventually result in insulation degradation or short-circuits in the windings.",

"Enabled": true,

"Assembly": "VSource",

"Name": "dcv",

"Title": "DC Voltage",

"AssemblyID": "VOL"

}, {

"Parameterized": true,

"Method": "unbalance",

"Unit": "%",

"Training": 1,

"TestType": 1,

"DefaultThresholds": ["NaN", "NaN", "NaN", "NaN", 3.0, 6.0, 9.0, "NaN"],

"ThresholdParameter": "",

"ExtendingParameters": "",

"Acknowledged": false,

"Description": "Voltage unbalance refers to any deviation in the voltage and current waveforms from a perfect sinusoidal shape, whether in terms of magnitude or phase shift. Under ideal conditions, with only linear loads connected to the system, the phases of the power supply should be 120 degrees apart in phase angle, and the magnitudes of their peaks should be equal.",

"Enabled": true,

"Assembly": "VSource",

"Name": "unbv",

"Title": "Voltage Unbalance",

"AssemblyID": "VOL"

}, {

"Parameterized": true,

"Method": "crestfactor",

"Unit": "",

"Training": 1,

"TestType": 1,

"DefaultThresholds": ["NaN", "NaN", "NaN", "NaN", 1.8, 2.0, 2.2, "NaN"],

"ThresholdParameter": "",

"ExtendingParameters": "",

"Acknowledged": false,

"Description": "The voltage crest factor is the ratio of the peak value of a sinusoidal waveform to its rms value. A high crest factor can lead to increased heating in electrical components, potential stress on circuit protection devices, reduced efficiency, and shortened lifespan of equipment due to the higher peak currents involved.",

"Enabled": true,

"Assembly": "VSource",

"Name": "cfv",

"Title": "Voltage Crest Factor",

"AssemblyID": "VOL"

}]

### Set Parameters <POST>

Usage: [http://IP:Port/api/setParameters?name={Asset Name](http://IP:Port/api/setParameters?name=%7bAsset%20Name)}

In order to update **Global Template Parameters** values, please pass **“GlobalTemplate”** for name. This will save default template values.

Description: Overwrites the default thresholds values and Acknowledgement changed by the user. User is not allowed to change any other parameters than the default thresholds and Acknowleded. User can update a single parameter’s default threshold and Acknowledgement by sending a jsonobject in the body of the query or alternatively send an array of parameters as shown in the example below.

Example: <http://IP:Port/api/setNameplate?name=fan>

* Open Postman Click "New Request"
* Set Method & URL POST http://localhost:5000/api/ setNameplate?name=fan
* Content-Type: application/json
* Add JSON Body

[{

"Parameterized": true,

"Method": "unbalance",

"Unit": "%",

"Training": 1,

"TestType": 1,

"DefaultThresholds": ["NaN", "NaN", "NaN", "NaN", **5.0, 8.0, 12.0**, "NaN"],

"ThresholdParameter": "",

"ExtendingParameters": "",

"Acknowledged": **true**,

"Description": "”

"Enabled": true,

"Assembly": "VSource",

"Name": "unbv",

"Title": "Voltage Unbalance",

"AssemblyID": "VOL"

}, {

"Parameterized": true,

"Method": "crestfactor",

"Unit": "",

"Training": 1,

"TestType": 1,

"DefaultThresholds": ["NaN", "NaN", "NaN", "NaN", **1.5**, 2.0, 2.2, "NaN"],

"ThresholdParameter": "",

"ExtendingParameters": "",

"Acknowledged": false,

"Description": "",

"Enabled": true,

"Assembly": "VSource",

"Name": "cfv",

"Title": "Voltage Crest Factor",

"AssemblyID": "VOL"

}]

* Click "Send"

## Settings

### Get Settings <GET>

Usage: <http://IP:Port/api/getJsonFile>

Example: <http://IP:Port/api/getSettings>

Description: Returns the Settings in json format.

### Set json File <POST>

Usage: <http://IP:Port/api/set>Settings

Example: <http://IP:Port/api/setSettings>

* Open Postman Click "New Request"
* Set Method & URL POST http://localhost:5000/api/setSettings
* Content-Type: application/json
* Add JSON Body

{

    "DBUrl": "http://localhost:8086",

    "DBToken": "3PWMjdMHy4u5xvImLigB6HC2VBxWcvG6Ll7c2JEpeqkzkYGxhMZkZCC74oegFNg==",

    "DBOrganization": "ntek",

    "RetentionPeriod": 36,

    "EventHistory": 10,

    "TrendRecordCount": 0,

    "NumberOfDaysToKeepLogFiles": 30,

    "NumberOfDaysToKeepBackupFiles": 365,

    "DataPath": "C:\\Users\\Public\\Documents\\smartsystems\\data",

    "FileProcessType": 0,

    "logLevel": 0

}

* Click "Send"

Description: Overwrites the Settings file and Updates the Setting values on the fly.

### Get Profile <GET> THIS COMMAND REQUIRES NTEK PRIVILAGES

Usage: <http://IP:Port/api/getProfile>

Example: <http://IP:Port/api/getProfile>

Description: Returns the Profile in json format.

### Set json File <POST> THIS COMMAND REQUIRES NTEK PRIVILAGES

Usage: <http://IP:Port/api/setProfile>

Example: <http://IP:Port/api/setProfile>

* Open Postman Click "New Request"
* Set Method & URL POST http://localhost:5000/api/setProfile
* Content-Type: application/json
* Add JSON Body

{

    "Company": 1,

    "OutlierSigma": 16.0,

    "Sigmas": [

        -100000.0,

        -14.0,

        -10.0,

        -6.0,

        6.0,

        10.0,

        15.0,

        100001.0

    ],

    "BoundaryScale": [

        1.0,

        0.05,

        1.0,

        1.0

    ],

    "Debug": **false**

}

* Click "Send"

Description: Overwrites the Profile file and Updates the Profile values on the fly.

## Diagnostic Queries

### Get Diagnostic <GET>

Usage: [http://IP:Port/api/getDiagnostic?name={Asset Name](http://IP:Port/api/getDiagnostic?name=%7bAsset%20Name)}

Example: [http://IP:Port/api/ getDiagnostic?name=fan](http://IP:Port/api/%20getDiagnostic?name=fan)

A screenshot of a computer program

AI-generated content may be incorrect.

Description: This query returns the diagnostic information in two separate sections. These are BarGraph and Tree List.

* ParentID and ID are used to create hierarchical structure.
* Names and Titles can be used to display the parameters in Bar graph and tree list. Name is the short form of the title.
* Status value indicates the Status of the parameters.
* Value is the value of the parameter. If the Value is NaN, that means this parameter cannot have a value to display. For instance, none of the BarGraph parameters will have a value. All of them will be NaN because they are displayed based on Status. Diagnostic status can be one of the following values.

Status: NoData=0, OK = 1, Warning=2, Inspect=3, Repair=4

* ListTree provides more information and it includes sub parameters. The top level in tree list is the diagnostic parameter (such as Load, Ground Fault, Unbalance, etc.). Under each diagnostic parameter, the parameters which are used to calculate the diagnostic parameter will be listed. Parameters can have sub parameters. For instance, rmsi is a parameter which feeds Load diagnostic parameter and rmsi has three sub parameters rmsiA, rmsB, and rmsC. Rmsi parameters status and value is calculated from the max of status and value of its sub parameters.

A screenshot of a computer

AI-generated content may be incorrect.

### Get Power quality

Usage: [http://IP:Port/api/getPQ?name={Asset Name](http://IP:Port/api/getPQ?name=%7bAsset%20Name)}

Example: [http://IP:Port/api/ getPQ?name=fan](http://IP:Port/api/%20getPQ?name=fan)

A screenshot of a computer

AI-generated content may be incorrect.

Description: This query returns the power quality information. It is structured in exactly the same way with Diagnostic. The only difference is the Status of the power quality parameters use the following status values.

Status: NoData = 0, OK = 1, Low=2, Medium=3, High=4

A screenshot of a computer

AI-generated content may be incorrect.

## User Management Queries

### Login <POST>

Usage: [http://IP:Port/api/login](http://IP:Port/api/login?username=%7bUserName%7d&password=%7bPassword%7d)

Example: [http://IP:Port/api/login](http://IP:Port/api/login?username=admin&password=admin)

JSON BODY: { username :"user1", password :"user1"}

### Logoff <POST>

Usage: <http://IP:Port/api/logoff>

Example: <http://IP:Port/api/logoff>

JSON BODY: {}

### Get Users <GET>

Usage: <http://IP:Port/api/getUsers>

Example: <http://IP:Port/api/getUsers>

### Create User <POST>

Usage: <http://IP:Port/api/createUser>

Example: <http://IP:Port/api/createUser>

JSON BODY: { username : "user1", password :"user1", level :2 }

*User Levels: Admin:0, Operator: 1, User: 2*

### Change User Level <POST>

Usage: <http://IP:Port/api/changeUserLevel>

Example: <http://IP:Port/api/changeUserLevel>

JSON BODY: { username :"user1", level :0 }

### Delete User <POST>

Usage: <http://IP:Port/api/deleteUser>

Example: <http://IP:Port/api/deleteUser>

JSON BODY: { username :"user1"}

### Change Password <POST>

Usage: <http://IP:Port/api/changePassword>

Example: <http://IP:Port/api/changePassword>

JSON BODY: { username :"user1", oldpassword :"user1", password :"admin" }

## Trending Queries

### Get Trend Hiearchy<GET>

Usage: http://IP:Port/api/getTrendHierarchy?name={Asset Name}&type={TYPE}

Example: [http://IP:Port/api/getTrendHierarchy?name=fan&type=diagnostic](http://IP:Port/api/getTrendHierarchy?name=fan&type=diagnostic%20)

**TYPE:** Type can be set to only **Diagnostic**, **PowerQuality**, or **Parameter**

Diagnostic: returns the Parameters and Extended Parameters (Sub Parameters) used in Diagnostic

PowerQuality: returns the Parameters and Extended Parameters (Sub Parameters) used in PowerQuality

Parameter: returns **ALL** the Parameters and Extended Parameters (Sub Parameters)

This query returns the parameters hierarchy as shown in the json data below. Client application should use this json data to create the parameters hierarchy and let the use select PARAMETERs=10 and/or EXTPARAMETERs=11 and send these selected parameters to GetTrendData command to get the trend data for the corresponding parameters. TREENODETYPE is the enumerator for NodeType.

*public enum TREENODETYPE { NONE, MASTER, RECYCLEBIN, ROOT, NODE, MACHINE, ASSET, ASSEMBLY, DIAGNOSTIC, POWERQUALITY, PARAMETER, EXTPARAMETER, FAULT, EVENT, MAINTENANCE, DATABASE, TEMPLATE }*

[{

"ParentID": 2,

"ID": 7,

"Name": "Turbulence",

"Title": "Turbulence ",

"Status": 4,

"AssemblyID": "FAN",

"Value": 0.484976565411424,

"NodeType": 11,

"Path": "Fan",

"Acknowledged": false

}, {

"ParentID": 2,

"ID": 8,

"Name": "BladePass",

"Title": "Blade Pass ",

"Status": 4,

"AssemblyID": "FAN",

"Value": 0.0,

"NodeType": 11,

"Path": "Fan",

"Acknowledged": false

}, {

"ParentID": 9,

"ID": 19,

"Name": "MechanicalUnbalance",

"Title": "Mechanical Unbalance ",

"Status": 2,

"AssemblyID": "MOT",

"Value": -57.1246799324493,

"NodeType": 11,

"Path": "Fan | Motor",

"Acknowledged": false

}]

### Get Trend Data<POST>

Usage: <http://IP:Port/api/getTrendData>

Example: <http://IP:Port/api/getTrendData>

Create the json data below to get the trend data from Smart Systems. Client application should set a start date and end date for the query and also parameter ids which are selected by the user.

{

"ParametersIds": [13, 28],

"StartDate": "2024-11-22T21:28:41.9779368-05:00",

"EndDate": "2025-03-24T21:28:41.9779445-04:00"

}

C# Class for Trend Request:

public class TrendRequest

{

public TrendRequest() { }

public List<int> ParametersIds { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

}

**Response:**

Response is return as **CombinedTrendData.**

If Threshold is null or empty, that means there is no threshold data. Therefore, threshold should not be plotted in this case.

Trend Data includes data to plot for each parameter requested.

TreeItem in Trend data includes the information about the parameter like parameter name, title, unit etc.

**public class CombinedTrendData**

{

public CombinedTrendData() { Parameters = new List<TrendData>(); }

public List<TrendData> Parameters { get; set; }

public List<TrendThresholds> Thresholds { get; set; } = null;

}

**public class TrendData**

{

public TreeItem UID { get; set; }

public List<DataPointTime> Data { get; set; }

}

**public class DataPointTime**

{

public DataPointTime() { }

public DateTime XAxis { get; set; }

public double? YAxis { get; set; }

}

## Service Operations

### Service Operations<GET>

Usage: [http://IP:Port/api/serviceOp&cmd={Command}&name={ServiceName}](http://IP:Port/api/serviceOp&cmd=%7bCommand%7d&name=%7bServiceName%7d)

Example: <http://IP:Port/api/serviceOp&cmd=start&name=db>

This query sends the command to Database Service (db) or Smart System Application Service (app)

**Commands:** start, stop, restart, status

**Service Name:** db, app

## Downloads

### Get Folder<GET>

Usage: [http://IP:Port/api/getFolder&name={FolderName}](http://IP:Port/api/getFolder&name=%7bFolderName%7d)

Example: <http://IP:Port/api/getFolder&name=dbbackup>

**Folder Name:** all, log, project, backup, dbbackup

This query compresses the requested folder and returns as a zip file.

**Folder Names:**

**Dbbackup:** backups influxdb, compress the backup file to a single zip file and sends back to client

**Log:** compress the log folder including all the log files to a single zip file and sends back to client

**Project:** compress the project folder to a single zip file and sends back to client

**Backup:** compress the project backup folder to a single zip file and sends back to client

**All:** compress database, log, project, and backup folders to a single zip file and sends back to client

## Faults & Events

### Get FauLts<GET>

Usage: [http://IP:Port/api/getFaults? name={Asset Name](http://IP:Port/api/getFaults?%20name=%7bAsset%20Name)}

Example: [http://IP:Port/api/getFaults? name=fan](http://IP:Port/api/getFaults?%20name=fan)

Description: Returns the faults in Bar Graph and Tree List format similar to GetPQ and GetDiagnostics

### Get Events<GET>

Usage: [http://IP:Port/api/getEvents? name={Asset Name](http://IP:Port/api/getEvents?%20name=%7bAsset%20Name)}

Example: [http://IP:Port/api/getEvents? name=fan](http://IP:Port/api/getEvents?%20name=fan)

Description: Returns the faults in Bar Graph and Tree List format similar to GetPQ and GetDiagnostics

### Get COMM<GET>

Usage: [http://IP:Port/api/getcomm? name={Asset Name](http://IP:Port/api/getcomm?%20name=%7bAsset%20Name)}&waveform={boolean}

Example: [http://IP:Port/api/getEvents? name=fan](http://IP:Port/api/getEvents?%20name=fan)&waveform=true

Description: returns the Commisioning results of SV500

Waveform is optional. If waveform = true, it will return the wavform. If the waveform is not passed or set to false, it will not return waveforms.

Returned data format is structedred as whon below

CommisioningResults

{

string SummaryReport

Waveform

{

double[] t

double[][] Vwave

double[][] Iwave

double[] f

double[] Vfft

double[] Ifft

}

}