Temp Stick API Documentation

Getting Started

Warning: to use the Temp Stick API you will need experience with programming and using REST APIs.

Working with the Temp Stick API

X-API-KEY

All requests made by your application must include the `X-API-Key` HTTP header.

```
X-API-KEY: YOUR_ACCCOUNT_API_KEY
```

GZIP

Data from the API is compressed using gzip and encoded as JSON, your application must support decompressing this format.

Temperature

All readings and settings regarding temperature are presented in **Celsius**.

The formula converting Celsius to Fahrenheit is: `(CELSIUS * 1.8) + 32`

Example function in PHP

```php
function celsius_to_fahrenheit($celsius, $round = 1){
  return round(($celsius * 1.8) + 32, $round);
}
```

Dates and time

All timestamps returned by the API are in the UTC time zone (denoted with "Z" or time offset +00:00).

Getting Your API Key

1. Log into your Temp Stick account
2. Go to the Account Page [https://temperaturestick.com/sensors/account] .
3. Select the "Developers" tab.
4. Click "Show Key" to reveal your API Key

Example GET request:

Below are three different ways to make a request the API: PHP, Javascript and a basic CURL request.

PHP

```php
<?php
// initialize curl
$curl = curl_init();


curl_setopt_array($curl, array(
  CURLOPT_URL => 'https://tempstickapi.com/api/v1/sensors/all', // the er
  CURLOPT_RETURNTRANSFER => true, // get the result of the CURL request
  CURLOPT_ENCODING => '', // auto decode the result (the API returns gzip
  CURLOPT_TIMEOUT => 0, // no time out
  CURLOPT_FOLLOWLOCATION => true, // follow redirects
  CURLOPT_CUSTOMREQUEST => 'GET', // this is a GET HTTP request
  CURLOPT_HTTPHEADER => array(
    'X-API-KEY: YOUR_API_KEY' // authenticate with X-API-Key (replace YOU
  ),
));

// get the response
$response = curl_exec($curl);
```

```
curl_close($curl);

// print the JSON response
echo $response;

?>
```

Javascript Example

```
// set the header and include the X-API-KEY
var requestHeaders = new Headers();

// (replace YOUR_API_KEY with the key from the Developer tab)
requestHeaders.append("X-API-KEY", "YOUR_API_KEY");

var requestOptions = {
  method: 'GET', // this is a GET request
  headers: requestHeaders, // include the custom header with X-API-Key
  redirect: 'follow' // follow redirects
};

// GET the HTTP response and log in the console
fetch("https://tempstickapi.com/api/v1/sensors/all", requestOptions)
  .then(response => response.text())
  .then(result => console.log(result))
  .catch(error => console.log('error', error));
```

CURL Example

```
curl --location --request GET 'https://tempstickapi.com/api/v1/sensors/all' -A "" \
--header 'X-API-KEY: YOUR_API_KEY'
```

A note for CURL:

Commandline CURL's default User-Agent header includes `curl` which is automatically blocked by the server firewall resulting in a "406 Not Acceptable" error. Use the `-A` parameter to change the default user agent. `-A ""` is allowed.

Example POST (update) request:

When changing settings via POST endpoints, the data should be an HTTP POST using `multipart/form-data`.

PHP

```php
$curl = curl_init();

curl_setopt_array($curl, array(
  CURLOPT_URL => 'https://tempstickapi.com/api/v1/sensor/YOUR_SENSOR_ID',
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_ENCODING => '',
  CURLOPT_CUSTOMREQUEST => 'POST',
  CURLOPT_POSTFIELDS => [
    'sensor_name' => 'Test',
    'send_interval' => '3600',
    'alert_interval' => '1800',
    'use_alert_interval' => '0',
    'alert_temp_below' => '',
    'alert_temp_above' => '',
    'alert_humidity_below' => '',
    'alert_humidity_above' => '',
    'connection_sensitivity' => '3',
    'use_offset' => '0',
    'temp_offset' => '0',
    'humidity_offset' => '0',
  ],
  CURLOPT_HTTPHEADER => array(
    'X-API-KEY: YOUR_API_KEY',
  ),
));

$response = curl_exec($curl);
```

```
curl_close($curl);
echo $response;
```

Common Issues

While troubleshooting and/or learning the API, it is advised to utilize Postman (https://www.postman.com/). Postman is a free application that facilitates the testing of APIs and enables a clear view of their output without the need for coding. Its user-friendly interface allows for the convenient testing of parameters and headers.

406 Not Acceptable

The security policy on the API server prohibits certain keywords (such as "sudo") or malformed requests. Requests with headers that contain keywords commonly used for malicious applications are automatically blocked. The following are some common triggers:

- User-Agent: by default CURL uses the User-Agent `curl/[version]`, you will need to use the –A parameter to change the User-Agent.
- GET requests with data in the body (such as `GET --data-raw ' '`)

Content-Type

Requests should use the following Content-Type headers:

- GET: `text/plain`
- POST: `multipart/form-data`

Alerts

Alerts | Get Alert

Get settings for a single alert

GET

```
https://tempstickapi.com/api/v1/alerts/:alert_id
```

Parameter

Field	Type	Description
alert_id	**Number**	The ID of your alert

- Success Response: [#success-examples-Alerts-Get_Alert-1_0_0-0]


```
{
    "type": "success",
    "message": "get alert",
    "data": {
        "id": "alert_id",
        "user_id": "user_id",
        "alert_name": "Connection",
        "reading_type": "connection",
        "reading_condition": "lost",
        "reading_value": "0.00",
        "alert_message": "[sensor_name]: connection [trigger_value]",
        "always_active": "1",
        "sensors_global": "1",
        "enabled": "1",
        "push_notifications": "1",
        "email_text_notifications": "0",
        "connection_sensitivity": "0",
        "sensors": [
            "sensor_id"
        ],
        "contacts": {
            "contact_id": {
                "id": "contact_id",
                "send_email": 1,
                "send_text": 1
```

```
            }
        },
        "active_when": {
            "0": {
                "alert_day": "0",
                "alert_time_start": 12,
                "alert_time_end": 24
            },
            "2": {
                "alert_day": "2",
                "alert_time_start": 1,
                "alert_time_end": 13
            }
        }
    }
}
```

Alerts I Get Alerts

Get all your alerts

GET

```
https://tempstickapi.com/api/v1/alerts/all
```

- Success Response: [#success-examples-Alerts-Get_Alerts-1_0_0-0]

```
{
  "type": "success",
  "message": "get alerts",
  "data": [
    {
      "id": "",
      "user_id": "",
      "alert_name": "Temperature",
      "reading_type": "temperature",
      "reading_condition": "above",
      "reading_value": "7.20",
```

```
        "alert_message": "[sensor_name]: [trigger_type] [trigger_condition]
        "always_active": "1",
        "sensors_global": "1",
        "enabled": "1",
        "active_when": [],
        "contacts": [],
        "sensors": [],
      }
    ]
}
```

Alerts I Get Sensor Notifications

Get the last seven days of alerts generated by the sensor

GET

```
https://tempstickapi.com/api/v1/sensor/notifications/:sensorId
```

Parameter

Field	Type	Description
sensorId	**String**	

Query Parameter(s)

Field	Type	Description
page	**Number**	*Default value: `0`*
items_per_page	**Number**	Maximum 100 items per page *Default value: `10`*

- Success Response: [#success-examples-Alerts-Get_Sensor_Notifications-1_0_0-0]

```
{
    "type": "success",
    "message": "get sensor notifications",
    "data": {
        "items": [
            {
                "id": "",
                "user_id": "",
                "alert_id": "",
                "event_type": "temperature",
                "event_message": "above 45 F (71.8 F)",
                "event_data": "{\"alert_condition\":\"above\",\"alert_val
                "created": "2022-05-10 22:49:41Z"
            }
        ],
        "count": "350"
    }
}
```

Alerts | Get User Notifications

Get the last seven days of alerts generated by all sensors

GET

```
https://tempstickapi.com/api/v1/user/notifications
```

Query Parameter(s)

Field	Type	Description
page	**Number**	*Default value: 0*
items_per_page	**Number**	Maximum 100 items per page

Field	Type	Description
		Default value: `10`

- Success Response: [#success-examples-Alerts-Get_User_Notifications-1_0_0-0]


```
{
    "type": "success",
    "message": "get user notifications",
    "data": {
        "items": [
            {
                "id": "",
                "user_id": "",
                "sensor_id": "",
                "alert_id": "",
                "event_type": "temperature",
                "event_message": "above 45 F (69.6 F)",
                "event_data": "{\"alert_condition\":\"above\",\"alert_val
                "created": "2022-05-11 17:37:21Z",
                "sensor": {}
            }
        ],
        "count": "350"
    }
}
```


Sensors

Sensors I Get Readings

Get the readings for a sensor given a specified period, *note: readings are shown in Celsius*

GET

```
https://tempstickapi.com/api/v1/sensor/:sensor_id/readings
```

- Basic usage: [#examples-Sensors-Get_Sensor_Readings-1_0_0-0]
- Custom date range: [#examples-Sensors-Get_Sensor_Readings-1_0_0-1]

```php
<?php

$curl = curl_init();

$example_parameters = [
  'setting' => 'last_week', // get readings for last week
  'offset' => 21600 // America/Denver (UTC offset in seconds)
];

curl_setopt_array($curl, array(
  CURLOPT_URL => 'https://tempstickapi.com/api/v1/sensor/YOUR_SENSOR_ID/r
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_ENCODING => '',
  CURLOPT_TIMEOUT => 0,
  CURLOPT_CUSTOMREQUEST => 'GET',
  CURLOPT_HTTPHEADER => array(
  'X-API-KEY: YOUR_API_KEY'
  ),
));

$response = curl_exec($curl);

curl_close($curl);
echo $response;
```

```php
<?php

$curl = curl_init();

$example_parameters = [
  'setting' => 'custom', // use a start and end date
  'start' => '2022-01-01', // start date range
```

```
  'end' => '2022-01-30', // end date range
  'offset' => 21600 // America/Denver (UTC offset in seconds)
];

curl_setopt_array($curl, array(
  CURLOPT_URL => 'https://tempstickapi.com/api/v1/sensor/YOUR_SENSOR_ID/r
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_ENCODING => '',
  CURLOPT_TIMEOUT => 0,
  CURLOPT_CUSTOMREQUEST => 'GET',
  CURLOPT_HTTPHEADER => array(
  'X-API-KEY: YOUR_API_KEY'
  ),
));

$response = curl_exec($curl);

curl_close($curl);
echo $response;
```

Parameter

Field	Type	Description
sensor_id	**String**	The ID of the sensor you want to get readings from

Query Parameter(s)

Field	Type	Description
offset	**Number**	The timezone offset in seconds from UTC used to calculate the specified time period
setting	**String**	The time period you want to grab readings from *Default value:* `today`

Field		Type	Description
			Allowed values: "`today, 24_hours, yesterday, last_week, 7_days, this_week, this_month, 30_days, last_month, three_months, custom`"
start	**optional**	**String**	The start date in YYYY-MM-DD format if "custom" is selected
end	**optional**	**String**	The end date in YYYY-MM-DD format if "custom" is selected

- Success Response: [#success-examples-Sensors-Get_Sensor_Readings-1_0_0-0]


```
{
  "type": "success",
  "message": "get messages",
  "data": {
    "start": "2022-05-11 04:00:00Z",
    "end": "2022-05-12 03:59:59Z",
    "readings": [
      {
      "sensor_time": "2022-05-11 22:26:39Z",
      "temperature": 22.06 (Celsius),
      "humidity": 27.45,
      "offline": 0
      }
    ]
  }
}
```

Sensors I Get Sensor

Get settings for a specific sensor, *note: readings are shown in Celsius*

GET

```
https://tempstickapi.com/api/v1/sensor/:sensor_id
```

Parameter

Field	Type	Description
sensor_id	**String**	The ID of the sensor you want to get readings from

- Success Response: [#success-examples-Sensors-Get_Sensor-1_0_0-0]


```
{
    "type": "success",
    "message": "get sensor",
    "data": {
        "version": "",
        "sensor_id": "",
        "sensor_name": "",
        "sensor_mac_addr": "",
        "owner_id": "",
        "type": "DHT",
        "alert_interval": "1800",
        "send_interval": "1800",
        "last_temp": 21,
        "last_humidity": 28.26,
        "last_voltage": 3,
        "battery_pct": 100,
        "wifi_connect_time": 1,
        "rssi": -37,
        "last_checkin": "2022-05-12 19:09:41-00:00Z",
        "next_checkin": "2022-05-12 19:39:41-00:00Z",
        "ssid": "",
        "offline": "0",
        "alerts": [],
        "use_sensor_settings": 0,
```

```
        "temp_offset": "0",
        "humidity_offset": "0",
        "alert_temp_below": "",
        "alert_temp_above": "",
        "alert_humidity_below": "",
        "alert_humidity_above": "",
        "connection_sensitivity": "3",
        "use_alert_interval": 0,
        "use_offset": "0",
        "last_messages": [
            {
                "temperature": 21,
                "humidity": 28.26,
                "voltage": "3.00",
                "RSSI": "-37",
                "time_to_connect": "1.0",
                "sensor_time_utc": "2022-05-12 19:09:41"
            }
        ]
    }
}
```

Sensors I Get Sensors

Get all the sensors assigned to the account, *note: readings are shown in Celsius*

GET

```
https://tempstickapi.com/api/v1/sensors/all
```

- Success Response: [#success-examples-Sensors-Get_Sensors-1_0_0-0]


```
{
  "type": "success",
  "message": "get sensors",
  "data": {
    "groups": [],
```

```
    "items": [
      {
        "version": "",
        "sensor_id": "",
        "sensor_name": "",
        "sensor_mac_addr": "",
        "owner_id": "",
        "type": "",
        "alert_interval": "",
        "send_interval": "",
        "last_temp": 25.4 (Celsius),
        "last_humidity": 21.77,
        "last_voltage": "3.48",
        "battery_pct": 100,
        "wifi_connect_time": "1",
        "rssi": "-57",
        "last_checkin": "2021-10-25 20:04:25",
        "next_checkin": "2021-10-25 21:04:25",
        "ssid": "",
        "offline": "1",
        "group": 0,
        "connection_sensitivity": "3",
        "temp_offset": 0,
        "humidity_offset": 0
      }
    ]
  }
}
```

Sensors | Update Sensor Settings

POST

```
https://tempstickapi.com/api/v1/sensor/:sensor_id
```

Parameter

Field	Type	Description
sensor_id	**String**	

Query Parameter(s)

Field	Type	Description
sensor_name	**String**	Give your sensor a name
send_interval	**Number**	In seconds, how often the sensor should wake up and send readings *Default value:* `300, 600, 1800, 3600`
use_alert_interval	**Number**	Set whether to enable the alert_interval setting (sensor will wake up more often if readings are above or below certain values) *Default value:* `0, 1`
alert_interval **optional**	**Number**	In seconds, how often the sensor should wake up when above or below certain conditions, this number must be less than or equal to send_interval *Default value:* `300, 600, 1800, 3600`
alert_temp_below **optional**	**Number**	Set as empty string "" if unused, use alert_interval value if temperature in celsius is less than this number

Field	Type	Description
alert_temp_above **optional**	**Number**	Set as empty string "" if unused, use alert_interval value if temperature in celsius is above this number
alert_humidity_below **optional**	**Number**	Set as empty string "" if unused, use alert_interval value if humidity is less than this number
alert_humidity_above **optional**	**Number**	Set as empty string "" if unused, use alert_interval value if humidity is above this number
connection_sensitivity	**Number**	How many times a sensor can miss a send_interval reading before triggering an offline alert *Default value:* `1-10`
use_offset	**Number**	Set whether to enable sensor calibration (offset) settings *Default value:* `0, 1`
temp_offset	**Number**	Number to offset temperature readings (this is normalized to celsius automatically if your account is set to use Fahrenheit)
humidity_offset	**Number**	Number to offset humidity readings

- Success Example [#success-examples-Sensors-Update_Sensor_Settings-1_0_0-0]

```
{
    "type": "success",
    "message": "update sensor",
    "data": {
        "sensor_name": "sensor_name",
        "send_interval": 1800,
        "alert_interval": 1800,
        "connection_sensitivity": "2",
        "use_offset": "1",
        "temp_offset": "1",
        "humidity_offset": "0",
        "use_sensor_settings": "0",
        "use_alert_interval": 0,
        "alert_temp_below": "",
        "alert_temp_above": "",
        "alert_humidity_below": "",
        "alert_humidity_above": ""
    }
}
```


User

User | Get Current User

Get your user information

GET

```
https://tempstickapi.com/api/v1/user
```

- Success Response: [#success-examples-User-Get_Current_User-1_0_0-0]

```
{
    "type": "success",
    "message": "Get user",
    "data": {
        "id": "",
        "email": "",
        "first_name": "",
        "last_name": "",
        "phone": "",
        "contact_email": "",
        "address_1": "",
        "address_2": "",
        "city": "",
        "state": "",
        "zip": "",
        "temp_pref": "F",
        "timezone": "America/New_York",
        "use_sensor_groups": "0",
        "chart_fill": "1",
        "send_reports": "1",
        "daily_reports": "1",
        "weekly_reports": "1",
        "monthly_reports": "1",
        "reports_specific_sensors": "1",
        "level": 10,
        "weekly_report_day": "1",
        "sensor_count": 1
    }
}
```

User | Get Email Reports

Get email report settings and any downloadable reports

GET

```
https://tempstickapi.com/api/v1/user/email-reports
```

- Success Response: [#success-examples-User-Get_Email_Reports-1_0_0-0]


```
{
    "type": "success",
    "message": "get email reports",
    "data": {
        "contacts": [],
        "sensors": [],
        "reports": {
            "daily": [
                {
                    "report_id": "",
                    "type": "daily",
                    "file": "daily-2022-05-12.zip",
                    "date_start": "2022-05-12 04:00:00",
                    "date_end": "2022-05-13 03:59:59",
                    "download": "URL_TO_DOWNLOAD"
                }
            ],
            "weekly": [],
            "monthly": []
        }
    }
}
```

User I Get Timezones

Get current list of supported timezones

GET

```
https://tempstickapi.com/api/v1/user/allowed-timezones
```

- Success Response: [#success-examples-User-Get_Timezones-1_0_0-0]

```
{
    "type": "success",
    "message": "get timezones",
    "data": {
        "America/New_York": "US Eastern",
        "America/Chicago": "US Central",
        "America/Denver": "US Mountain West",
        "America/Phoenix": "US Mountain (No DST)",
        "America/Los_Angeles": "US Pacific",
        "America/Anchorage": "US Alaska",
        "Pacific/Honolulu": "US Hawaii",
        "America/Glace_Bay": "(GMT-04:00) Atlantic Time (Canada)",
        "America/Argentina/Buenos_Aires": "(GMT-03:00) Buenos Aires",
        "America/Noronha": "(GMT-02:00) Mid-Atlantic",
        "Atlantic/Azores": "(GMT-01:00) Azores",
        "Europe/London": "(GMT) London",
        "Europe/Berlin": "(GMT+01:00) Europe (CET)",
        "Africa/Algiers": "(GMT+01:00) West Africa",
        "Africa/Cairo": "(GMT+02:00) Central Africa",
        "Africa/Addis_Ababa": "(GMT+03:00) East Africa",
        "Europe/Moscow": "(GMT+03:00) Moscow",
        "Asia/Dubai": "(GMT+04:00) Abu Dhabi",
        "Asia/Karachi": "(GMT+05:00) Pakistan",
        "Asia/Calcutta": "(GMT+05:30) Calcutta",
        "Asia/Dhaka": "(GMT+06:00) Dhaka",
        "Asia/Bangkok": "(GMT+07:00) Bangkok",
        "Asia/Shanghai": "(GMT+08:00) Asia",
        "Asia/Tokyo": "(GMT+09:00) Tokyo",
        "Australia/Brisbane": "(GMT+10:00) Brisbane",
        "Asia/Magadan": "(GMT+11:00) Magadan",
        "Pacific/Auckland": "(GMT+12:00) Auckland",
        "Pacific/Tongatapu": "(GMT+13:00) Nuku alofa",
        "Pacific/Kiritimati": "(GMT+14:00) Kiritimati"
    }
}
```

User | Update User Display Preferences

Change display preferences including time zone and temperature format

POST

```
https://tempstickapi.com/api/v1/user/display-preferences
```

Query Parameter(s)

Field	Type	Description
timezone	**String**	Time zone of the user, functionally used in determining if an alert should trigger if a time window is set *Default value:* `See /user/allowed-timezones`
temp_pref	**String**	Display alerts in fahrenheit or celsius, readings and settings are always done in celsius for consistency *Default value:* `F, C`
chart_fill **optional**	**Number**	Set whether to have the chart normalized on the Y-axis. *Default value:* `0, 1`

- Success Response: [#success-examples-User-Update_User_Display_Preferences-1_0_0-0]


```
{
    "type": "success",
    "message": "updated display preferences",
    "data": {
        "timezone": "America/New_York",
        "temp_pref": "F",
        "chart_fill": "1"
```

```
    }
}
```

Examples and Useful Code

Battery Percentage Calculation

At times, you may want to calculate the battery percentage of your sensor from the voltage sent by the sensor. This is most accurate with alkaline batteries which have a steady voltage decline while lithium / rechargeable batteries tend to hold voltage until failure.

```
const getBatteryPercentage = voltage => {

  // typical voltage range of sensors is 2.3v to 3.0v
  // (0% to 100%)

  const baseline = 2.30; // lowest allowed voltage
  const remaining_voltage = voltage - baseline;

  let voltagePct = remaining_voltage / 0.007;

  if(voltagePct > 100){ voltagePct = 100; }
  if(voltagePct < 0){ voltagePct = 0; }

  return Math.round(voltagePct);

}
```

Saving Readings and Alerts to Google Sheets

This NodeJS script uses the Temp Stick API to fetch data from your sensors and update the readings while observing each sensor's next check-in timestamp. Readings will be saved to the "Readings" tab of your sheet. If any alerts are triggered, they will be recorded in the "Alerts" tab of your Google spreadsheet.

Setup

- Create a Service Account JSON key to access Google Sheets
- Create a new Google Sheet (get the ID from the URL)
- Create the tabs "Readings" and "Alerts"
- "Readings" should have the following columns: "Sensor", "Temp C", "Temp F", "Humidity", "RSSI", "Time to Connect", "Battery", "Offline", "Timestamp"
- "Alerts" should have the following columns: "Sensor", "Alert", "Timestamp"

To set up a service account for Google Services, go to this guide [https://theoephraim.github.io/node-google-spreadsheet/#/guides/authentication?id=service-account] for instructions on obtaining a Service Key.

Keep in mind:

- Your Google keyfile JSON is sensitive information, similar to a password to your Google account. Do not expose it publicly.
- Similarly, your Temp Stick API key should be treated as confidential. It provides access to your Temp Stick account, hence don't include it directly in your code, especially if you're planning to share your code or upload it on platforms like Github. Always ensure your keys are securely stored and accessed.

Note

This is a basic example of integrating Google Sheets with Temp Stick. For a more robust application, you might want to consider saving your readings and alerts to a dedicated database (such as SQL or MongoDB). Using a database could provide more flexibility and efficiency than a spreadsheet for data management and retrieval.

This will not run in a web browser and requires NodeJS to use.

NodeJS Javascript

```
// Import necessary libraries
const {google} = require('googleapis'); // Google's Node.js client librar
const axios = require('axios'); // HTTP client
const fs = require('fs'); // Node.js File System module

// Define the path for a JSON file where the application will store data.
// The application will remember the last 1000 readings per sensor.
const dataFile = __dirname + '/data.json';

// Provide the API key for the Temp Stick service. This is reading a txt
const TEMP_STICK_API_KEY = fs.readFileSync(__dirname + "/temp-stick-api-k

// Configure the Google JWT (JSON Web Token) authorization object
const auth = new google.auth.JWT({
  // Point to your Google service account file
  keyFile: __dirname + '/config/temp-stick-datalogging.json',
  // Specify the scope that the JWT is allowed to access.
  // In this case, we're giving it access to Google Sheets.
  scopes: ['https://www.googleapis.com/auth/spreadsheets'],
});

// Initialize the Google Sheets API client
const sheets = google.sheets({version: 'v4', auth});

// Define the ID of the Google Sheets spreadsheet to be accessed. Replace
const spreadsheetId = '';

// Create a variable to store historical data.
let dataHistory = null;

// Check if the file specified by dataFile already exists
if(fs.existsSync(dataFile)) {
  // If it does, read the file and parse its JSON content into the dataHi
  dataHistory = JSON.parse(fs.readFileSync(dataFile));
} else {
  // If it doesn't, initialize dataHistory with an object that has an emp
  dataHistory = {
    alerts: [],
```

```
  };
}

// Define a function to save the current state of dataHistory back into t
const saveData = () => {
  fs.writeFileSync(dataFile, JSON.stringify(dataHistory));
}

// The getSensors function retrieves sensor data from the Temp Stick API
async function getSensors() {
  try {
    // Send a GET request to the Temp Stick API's 'sensors/all' endpoint
    const response = await axios.get('https://tempstickapi.com/api/v1/ser
      headers: {
        'X-API-KEY': TEMP_STICK_API_KEY // Include the Temp Stick API key
      },
      timeout: 0, // Do not set a timeout for the request
    });

    const res = response.data; // Extract the data from the response

    // If the API request was successful, return the list of sensor items
    // If not, return false
    if(res.type === 'success') {
      return res.data.items;
    } else {
      return false;
    }

  } catch (error) { // If there's an error in the try block, log it
    console.error(error);
  }
}

// The getAlertNotifications function retrieves alert notifications from
async function getAlertNotifications() {
  try {
    // Send a GET request to the Temp Stick API's 'user/notifications' er
    const response = await axios.get('https://tempstickapi.com/api/v1/use
      headers: {
```

```
        'X-API-KEY': TEMP_STICK_API_KEY // Include the Temp Stick API key
      },
      timeout: 0, // Do not set a timeout for the request
    });

    const res = response.data; // Extract the data from the response

    // If the API request was successful, return the list of notification
    // If not, return false
    if(res.type === 'success') {
      return res.data.items;
    } else {
      return false;
    }

  } catch (error) { // If there's an error in the try block, log it
    console.error(error);
  }
}

// The getMessages function retrieves messages for a specific sensor from
async function getMessages(sensorId) {
  try {
    // Send a GET request to the Temp Stick API's 'sensor/{sensorId}/read
    // The request will include a query string specifying the settings ar
    const response = await axios.get(`https://tempstickapi.com/api/v1/ser
      headers: {
        'X-API-KEY': TEMP_STICK_API_KEY // Include the Temp Stick API key
      },
      timeout: 0, // Do not set a timeout for the request
    });

    const res = response.data; // Extract the data from the response

    // If the API request was successful, return the list of readings
    // If not, return false
    if(res.type === 'success') {
      return res.data.readings;
    } else {
      return false;
```

```
    }

  } catch (error) { // If there's an error in the try block, log it
    console.error(error);
  }
}

// Function to add alerts to a Google Spreadsheet
async function addAlertsToSheet(alerts) {

  // Initialize an empty array to hold the data
  let data = [];

  // Loop through the alerts array
  for(alert of alerts) {

    // Combine the event type and event message into one string
    let message = alert.event_type + " " + alert.event_message;
    let created = alert.created;

    // Convert the created timestamp into Ontario time
    let ontarioTime = new Date(created).toLocaleString("en-US", {
      timeZone: "America/Toronto",
    });

    // Add a new row of data for each alert
    data.push([
      alert.sensor_id,
      message,
      ontarioTime,
    ])
  }

  // Sort the data array by date
  data = data.sort((a, b) => {
    return new Date(b[2]) - new Date(a[2]);
  });

  const sheetName = 'Alerts'; // replace with your sheet name
```

```
  // Get the spreadsheet details
  const spreadsheet = await sheets.spreadsheets.get({
    auth,
    spreadsheetId,
  });

  // Find the sheetId of the sheet with the given name
  const sheet = spreadsheet.data.sheets.find(sheet => sheet.properties.ti
  const sheetId = sheet.properties.sheetId;

  // Insert rows into the spreadsheet
  await sheets.spreadsheets.batchUpdate({
    auth,
    spreadsheetId,
    resource: {
      requests: [{
        insertDimension: {
          range: {
            sheetId: sheetId,
            dimension: 'ROWS',
            startIndex: 1,
            endIndex: 1 + data.length
          },
          inheritFromBefore: false
        }
      }]
    }
  });

  // Update the spreadsheet with the new data
  await sheets.spreadsheets.values.update({
    auth,
    spreadsheetId,
    range: sheetName + '!A2', // assuming that your data starts from colu
    valueInputOption: 'USER_ENTERED',
    resource: { values: data },
  });

}
```

```
// Function to add sensor readings to a Google Spreadsheet
async function addSensorValuesToSheet(newReadings) {

  // Initialize an empty array to hold the data
  let data = [];

  // Loop through the readings array
  for(reading of newReadings) {

    // Grab the necessary values from the reading
    let tempC = reading.temperature;
    let tempF = (tempC * 9/5 + 32).toFixed(1); // Convert the temperature
    let humidity = reading.humidity;
    let checkin = reading.sensor_time;

    // Convert the check-in time to Ontario time
    let ontarioTime = new Date(checkin).toLocaleString("en-US", {
      timeZone: "America/Toronto",
    });

    let rssi = reading.rssi;
    let time_to_connect = reading.time_to_connect;
    let battery_pct = getBatteryPercentage(reading.voltage); // Get the b

    // Add a new row of data for each reading
    data.push([
      reading.sensor_id,
      tempC,
      tempF,
      humidity,
      rssi,
      time_to_connect,
      battery_pct,
      reading.offline,
      ontarioTime,
    ])
  }

  // Sort the data array by date
  data = data.sort((a, b) => {
```

```
    return new Date(b[8]) - new Date(a[8]);
  });

  const sheetName = 'Readings'; // replace with your sheet name

  // Insert rows into the spreadsheet
  await sheets.spreadsheets.batchUpdate({
    auth,
    spreadsheetId,
    resource: {
      requests: [{
        insertDimension: {
          range: {
            sheetId: 0,
            dimension: 'ROWS',
            startIndex: 1,
            endIndex: 1 + data.length
          },
          inheritFromBefore: false
        }
      }]
    }
  });

  // Update the spreadsheet with the new data
  await sheets.spreadsheets.values.update({
    auth,
    spreadsheetId,
    range: sheetName + '!A2', // assuming that your data starts from colu
    valueInputOption: 'USER_ENTERED',
    resource: { values: data },
  });

}

// Function to process the alerts
const processAlerts = async () => {
  // Fetch alert notifications
  const alerts = await getAlertNotifications();
```

```
  // Initialize an empty array for new alerts
  const newAlerts = [];

  // Loop over all alerts
  for(let alert of alerts) {
    // Check if the alert already exists in the history
    let alertExists = false;
    for(let existingAlert of dataHistory.alerts) {
      if(existingAlert.id === alert.id) {
        alertExists = true;
        break;
      }
    }

    // If the alert doesn't exist, add it to the new alerts array
    if (!alertExists) {
      newAlerts.unshift(alert);
    }
  }

  // Add new alerts to the history and limit history size to 1000 entries
  dataHistory.alerts = [...newAlerts, ...dataHistory.alerts];
  if(dataHistory.alerts.length > 1000) {
    dataHistory.alerts = dataHistory.alerts.slice(0, 1000);
  }

  // Add new alerts to the Google Sheet and save data
  await addAlertsToSheet(newAlerts);
  saveData();
};

// Function to process sensor data
const processSensors = async () => {
  // Fetch sensors data
  const sensors = await getSensors();

  // Process alerts after 60 seconds to allow for data to come in
  setTimeout(processAlerts, 60000);

  // If sensors are available
```

```
if (sensors && sensors.length > 0) {
  let nextCheckIn = null;
  const now = new Date();
  const batchUpdate = [];

  // Loop over all sensors
  for (let sensor of sensors) {

    // If there's no history for this sensor, create a new array for it
    if(!dataHistory[sensor.sensor_id]) {
      dataHistory[sensor.sensor_id] = [];
    }

    // Get messages for each sensor
    const readings = await getMessages(sensor.sensor_id);
    const newReadings = [];

    // Loop over all readings
    for(let reading of readings) {
      reading.sensor_id = sensor.sensor_id;
      let readingExists = false;

      // Check if the reading already exists in the history
      for(let existingReading of dataHistory[sensor.sensor_id]) {
        if(existingReading.sensor_time === reading.sensor_time) {
          readingExists = true;
          break;
        }
      }

      // If the reading doesn't exist, add it to the new readings array
      if (!readingExists) {
        newReadings.unshift(reading);
        batchUpdate.push(reading);
      }
    }

    // Add new readings to the history and limit history size to 1000 e
    dataHistory[sensor.sensor_id] = [...newReadings, ...dataHistory[ser
    if(dataHistory[sensor.sensor_id].length > 1000) {
```

```
        dataHistory[sensor.sensor_id] = dataHistory[sensor.sensor_id].sli
      }

      // Calculate the next check-in time
      let sensorNextCheckIn = new Date(sensor.next_checkin + "Z");
      while (sensorNextCheckIn <= now) {
        sensorNextCheckIn = new Date(sensorNextCheckIn.getTime() + 60000)
      }
      if (!nextCheckIn || sensorNextCheckIn < nextCheckIn) {
        nextCheckIn = sensorNextCheckIn;
      }
    }

    // If there are any new readings, add them to the Google Sheet
    if(batchUpdate.length > 0) {
      await addSensorValuesToSheet(batchUpdate);
    }

    // Save the data
    saveData();

    // Calculate sleep time until the next check-in
    if (nextCheckIn) {
      let sleepTime = nextCheckIn.getTime() - now.getTime() + 60 * 1000;
      if(sleepTime < 0) {
        sleepTime = 15000;
      }

      // Delay the next sensor process by sleep time
      if (sleepTime > 0) {
        console.log(`Sleeping for ${sleepTime / 1000} seconds...`);
        setTimeout(processSensors, sleepTime);
      } else {
        // always sleep for 60 seconds although this should never process
        setTimeout(processSensors, 60000);
      }
    } else {
      console.log('No next check-in time. Exiting...');
    }
  } else {
```

```
    console.log('No sensors. Exiting...');
  }
};

// Start processing the sensors
processSensors();
```

Generated with apidoc [https://apidocjs.com] 0.53.0 - Thu Jul 20 2023 15:36:27 GMT-0600 (Mountain Daylight Time)