Open API - MPCB OCEMS Portal

Overview

This Open API document is used to integrate multiple software clients with the Maharashtra Central Server Software. All communication between the Central Server and the acquisition clients at the industry site is done through HTTP-based REST APIs.

All the APIs are authenticated. Any approved software client that follows the specified Open API can upload data to the Central Server.

Supported Operations

All clients should support full integration with all these operations.

- Real Time Data Upload
- Delayed Data Upload
- Remote Analyzer Configuration
- Remote Analyzer Calibration
- Analyzer Diagnostic Fetch

Key Concepts

The following are the key concepts to be followed while working with the Open API

- **Site ID**: Unique Site ID identifying the specific industry.
- **Monitoring ID**: Each Site has multiple monitoring stations. Each monitoring station will be assigned a unique monitoring ID relative to the Site
- **Analyser ID**: Each analyser make and model will be assigned a unique Analyser ID
- **Parameter ID**: Each monitored parameter will have a common unified ID across all industries.

Client-Side Software Requirement

Each client software implementing the API should also comply with "C*lient-Side Software Requirement published by MPCB*"

Key API Requirements

- Each site client software must collect data from the analyser based on the defined polling frequency. The ideal data sampling frequency from the analyser is 10 seconds. Data transmission to the Central Server should occur at a frequency of 1 minute. The raw data and linearized data must be transmitted to the server along with the data quality code and the captured timestamp.

- **Optional for Pilot Phase:**

  The captured data should be sent to the server immediately after encrypting it using the digital private key. This private key must be kept secure and should not be tampered with or shared with anyone. The client software should have a way to generate an industry-specific digital signature to establish data connectivity with the Central Server Software. The industry should obtain the digital signature as soon as possible, provide the necessary details, and use it for data encryption to ensure secure and tamper-proof data transmission.
- The transmitted data should be encrypted with zipped data in ISO-7168. All API request data transfer should be using a REST Service over HTTP protocol.
- The client site software should wait for successful upload and read subsequent instructions (Remote calibration, Configuration update, Diagnostics information etc.) from the Central Server Software
- Upon receiving instructions on Remote calibrations or Configuration update, the site client software should invoke the Remote Calibration or Configuration update services to download the corresponding configurations.
- In case of any communication failure or any delayed data transmitted beyond a period of 15 minutes, site software should store the data locally and upload to the Delayed Data Upload URL and not to the Real Time upload URL. This is to ensure that the delayed data is captured separately at the Central Server and can be tracked for any integrity issues.
- All clients should transmit data captured directly from the analyser from the site location. Any data transmission from different locations will be rejected by the server.
- All requests from the client to the server should be authenticated with requests only. Any unauthenticated requests will be discarded or not processed.

Basic Organization of API

https://onlinecems.ecmpcb.in/mpcb

Resource	Description	Route	Request type
Data upload	This is for uploading data to the central server from the client. Any authenticated client with proper credentials can upload data to the server using this API. Only real time data (delay of max 2 min) will be accepted through /realtimeupload URL, and any delayed data should be uploaded using /delayedUpload URL	/realtimeUpload /delayedUpload	POST
Configuration Download	This is for downloading the configuration from the server. Any approved client software can download the configuration from the server using this API,	/getConfig	POST
Fetch Client Configuration	When the ConfigurationUpdateFlag is set to “True” in the response of the Realtime Upload or Delayed Upload, client needs to provide the current configurations set at the	/uploadConfig	POST

	Analyser.		
Acknowledge Configuration Download	When the ConfigurationDownloadFlag flag is set to “True” in the response of the Realtime Upload or Delayed Upload, the client software should use the /getConfig URL to download the configuration from the Central Server. Once the configuration is updated in the Analyser, the client should update the Central Server with the status of the configuration update. Till the status is updated to success, client will be asked continuously to update the configuration by setting the ConfigurationDownload flag to “True”.	/completedConfig	POST
Calibration Download service	When the RemoteCalibrationUpdateFlag is set to “True”, the client software should use this URL to download the configuration required for calibration. The remote calibration data and sequence should be updated to the Calibrator locally	/getcalibrationconfig	POST
Calibration Update Acknowledgement	After successful download of the Remote Calibration Configuration and updating the local calibrator or analyser who will be performing the calibration, the client software should acknowledge the status of calibration status using this URL	/updatecalibrationconfig	POST
Diagnostic Upload service	When the DiagnosticUpdateFlag is set to “True”, the client software should use this URL for uploading the diagnostic information, including any internal state of the analyser as per the analyser make and model.	/uploadDiagnosticInfo	POST

Central Server Software Open API

Authentication Mechanism for the API

Each API request header should have the following information

A. **Timestamp**

B. **Authorization** - Encrypted data from the site which has the authentication digest. Each request should send authentication digest with the following encrypted data

- site_id - Unique Site Id provided by the Maharashtra Pollution Control Board for each site for authentication
- software_version_id - Software version set by the Central Server for the industry
- time_stamp_data - Timestamp Format: "%Y-%m-%dT%H:%M:%SZ"
- site_private_key

For example:

- <site_id>,<software_version_id>,<time_stamp_data>,<site_private_key>

Sample Data:

'site_1234,ver_1.0,2026-02-12T16:28:13Z,c2l0ZV8yMTg4LHZlcl7xLjAs5OjAy####'

The above concatenated string must be encrypted using the AES encryption mechanism and included in the request header as:

Authorization: Basic <encrypted_value>

The authentication digest is decrypted using the Site Private Key. The timestamp is ensured to be not more than 15 minutes (configurable) from the current timestamp. The software version is verified against the registered software version with the Central Server Software. This ensures the data is encrypted just before transmission, and the client's program has access to Site Private Key and the current registered software version. The registered software version will be updated from Central Server Software time to time and hence does not depend on client software version.

Data Upload

The standard response format is described below. Any approved client software with proper credentials can send data to the central server using this API.

Data Upload Format

The API supports 2 different types of data format for data upload. The data upload follows an ISO-7168 format zip file, or a simplified delimited or fixed width file format.

The zip file uploaded to the server will be in multipart/form-data format. The data should be sent in a zip format. The uploaded zip file will have two files, namely 1. Data File, 2. Metadata File. The Data file should be encrypted using the Site Private Key. The zip file should be uploaded to the server with proper authentication using the key. Else the response with HTTP 401 with "Authentication Failure" will be returned.

The metadata file will specify the file formats (ISO-7168, CSV, FixedWidth) etc., and the data file should comply with the same. This gives flexibility to support different file formats based on the analyser or client's software capability.

However, all files have to follow the basic guidelines

1. Data should encrypted
2. File should zipped
3. Metadata file should provide the file specification and format

4. Header should have the encryption digest for decryption of the data

Note:

The metadata.csv file should consist of the following header:

'SITE_ID, SITE_UID, MONITORING_UNIT_ID, ANALYZER_ID, PARAMETER_ID, PARAMETER_NAME, RAWREADING, READING_UNIT, DATA_QUALITY_CODE, VALIDATED_READING, UNIX_TIMESTAMP, CALIBRATION_FLAG, MAINTENANCE_FLAG'

Data File Naming Convention*:*

The data file must follow this filename format:
Eg: siteId_monitoringId_timestamp.csv
Where:

- **siteId** = site_1234
- **monitoringId** = Stack1
- **timestamp** format = %Y%m%d%H%M%S

sample: site_1234_Stack1_20260213122520.csv

The generated data file (e.g., site_1234_Stack1_20260213122520.csv) must:

- Contain data strictly in the same column order as defined in metadata.csv
- Be encrypted using the **private key**

Sample: site_1234,site_1234,Stack1,analyzer_8,parameter_10,PM,15,unit_15,U,15,1770965988,0,0

After preparing the data in this exact metadata format, the entire file content must be encrypted using the provided private key and AES-CBC configuration, and the final encrypted output must be Base64 encoded.

Instructions:

Please use the following configuration for AES encryption:

- **Algorithm**: AES
- **Mode**: CBC (Cipher Block Chaining)
- **Signature Encoding**: Base64
- **Block Size**: 32 bytes
- **Key Size**: 32 bytes
- **Padding Character**: #
- **Initialization Vector (IV)**: '\x00'
- **IV Default Size**: 16 bytes

  Ensure that the plaintext is padded using the # character to match the required block size before encryption. The final encrypted output should be encoded using Base64.

After generating:

- metadata.csv
- site_1234_Stack1_20260213122520.csv (encrypted data file)

Both files must be compressed into a single **ZIP file**.

ZIP File Structure

The ZIP file should contain:

- Metadata.csv
- site_1234_Stack1_20260213122520.csv

Example ZIP filename: site_1234_Stack1_20260213122520.zip
After creating the ZIP file:

- Use the designated **Upload URL (POST API)**
- Include required authentication headers

Request Details

Upload data to Central Server

This method uploads data to the server. The requests will be authenticated and hence should have the authentication header as described in section "Authentication Mechanism for the API"

https://onlinecems.ecmpcb.in/mpcb/realtimeUpload

OR

https://onlinecems.ecmpcb.in/mpcb/delayedUpload

Method: **POST**
Parameters: The file to be uploaded should be send as the parameter.

Returns: Response JSON which contains the status as either **success** or **failure**

Note: realtimeUpload URL will take only data that is captured from the analyser during the last poll frequency defined by regulator. Anything delayed should be uploaded to delayedUpload URL

If the upload is successful, the following response will be returned.

{

"status": "Success",

"serverConfigLastUpdatedTime":

"<time>", "ConfigurationDownloadFlag":

"<Flag>", "ConfigurationUpdateFlag":

"<Flag>", "RemoteCalibrationUpdateFlag

": "<Flag>", "DiagnosticUpdateFlag":

"<Flag>", "statusMessage": "file uploaded

successfully."

}

Where the **<time>** is the last updated time of server configurations and **<Flag>** is a Boolean value depending upon whether the site configuration is updated or not.

Flag can have values “True” or “False”

Eg:

{

"status": "Success",

"serverConfigLastUpdatedTime": "2015-02-24T13:21:19Z",

"ConfigurationDownloadFlag": "True",

"ConfigurationUpdateFlag": "False",

"RemoteCalibrationUpdateFlag ": "True",

"DiagnosticUpdateFlag": "False",

"statusMessage": "file uploaded successfully. "

}

If the upload fails, the following response will be returned.

{

"status": "Failed",

"statusMessage": "No files were uploaded."

}

Configuration Download from server

The configuration download request helps the client software understand the format and parameters which should be transferred to the server. This request enables the client software to download the entire configuration for the monitoring station. This configuration should be synchronized with the analyser.

This method downloads the configuration from the server.

Central Server Software Open API

http://ipaddress:port/MPCBServer/getConfig

Path: **getConfig**

Method: **POST**

Parameters: The site id will be passed as the parameter.

Returns: The response json contains, the configuration in case of success or failure message in case of failure.

Request body:

{

"siteId": <site-id>, "monitoringid":

<monitor-id>

}

If the configuration download request is success, the following response will come. Any approved client software with proper credentials can download the configurations from the central server using this api. Software with improper credentials will be blocked. The request should have the valid authenticated headers.

Request Format

The request for site configuration update will be in the following format.

```
{
   "siteId": “site_108”, "monitoringid":
   “ETP_PLANT”
}
```

Response Format

```
{
   "status": "Success",
   "serverConfigLastUpdatedTime": <ServerConfigUpdatedLastTime>, "SiteDetails":
   {
      "siteName": <SiteName>,
      "siteLabel": <SiteLabel>,
      "siteConfigLastUpdatedTime": <SiteConfigUpdatedLastTime>,
      "siteId": <site id>,
         “customparameters” :{}
   },
   "CollectorDetails":[ { “CollectorType”: <>,
      “CollectorName”: <>,
      "ConfiguredChannels": <>,
      "PollingStep": <polling step>,
      "ChecksumStatusBit": <checksum bit>,
      "Address": <address>,
      "HeartBeat": <heartbeat>,
      "DataFormatBits": "00",
      "Port": <port>,
      "CommunicationTimeOut": <communication bit>
      “customparameters” :{}
  }],
   "configJson": {
      "monitoringType": {
         "required": "True",
         "padding": "-",
         "start_pos": 55,
         "end_pos": 64,
         "type": "string",
         "alignment": "left"
      },
      "monitoringId": {
         "required": "True",
         "padding": "-",
         "start_pos": 65,
         "end_pos": 84,
         "type": "string",
         "alignment": "left"
      },
      "QualityCode": {
         "required": "True",
         "padding": "*",
```

```
        "start_pos": 42,
        "end_pos": 43,
        "type": "string",
        "alignment": "left"
    },
    "SensorTime": { "required":
        "True",
        "padding": "-",
        "start_pos": 44,
        "end_pos": 54,
        "type": "string",
        "alignment": "left"
    },
    "parameterId": { "required":
        "True",
        "padding": "-", "start_pos": 85,
        "end_pos": 100, "type":
        "string", "alignment":
        "left"
    },
    "parameterName": { "required":
        "True",
        "padding": "*", "start_pos": 11,
        "end_pos": 25, "type":
        "string", "alignment":
        "left"
    },
"Reading": {
        "required": "True",
        "padding": "*", "start_pos": 26,
        "end_pos": 41, "type":
        "string", "alignment":
        "left"
    },
    "id": {
            "required": "True",
             "padding": "-", "start_pos": 1,
        "end_pos": 8, "type":
        "string", "alignment":
        "left"
    },
    "sensorChannel": { "required":
        "True",
        "padding": "-", "start_pos": 9,
        "end_pos": 10, "type":
        "string", "alignment":
        "left"
    },
    "analyzerId": { "required":
        "True",
        "padding": "-", "start_pos":
        101,
        "end_pos": 115, "type":
        "string", "alignment":
        "left"
    }
},
```

```
"AcquisitionSystemDetails": { "AcquisitionVersion":
  <Version Number>, "AcquisitionSystem": <Acquisition
  System Name>
},
"SensorA": {
  "collectorType": <Monitoring Type>,
  "monitoringType": <Monitoring Type>,
  "monitoringId": <Monitoring Id>,
  "ChannelNo": "0",
  "GaugeMinimum": "",
  "CoefficientA": "", "parameterId":
  <parameter id>,
  "GaugeMaximum": "",
  "MeasurementUnit": <measurement unit>,
  "compPort": "",
  "parameterName": <parameter name>,
  "CoefficientB": "",
  "analyzerId": <analyzer id>
  “customparameters” :{}
},
.
.
"SensorN": {
  "monitoringType": <Monitoring Type>,
  "monitoringId": <Monitoring Id>,
  "ChannelNo": "0",
  "GaugeMinimum": "",
  "CoefficientA": "1",
  "parameterId": <parameter id>,
  "GaugeMaximum": "",
  "MeasurementUnit": <measurement unit>,
  "compPort": "",
  "parameterName": <parameter name>,
  "CoefficientB": "0",
  "analyzerId": <analyzer id>,
  “customparameters” :{}
}
}
```


If the configuration request status is failed, the response will be


```
{
  "status": "Failed"
}
```

Fetch Configuration From Client

This method will be invoked by the client to upload the current configuration in the analyser to the Central Server Software when the ConfigurationUpdateFlag is set to “True”

http://ipaddress:port/MPCBServer/uploadConfig

Path: **uploadConfig**

Method: **POST**

Parameter: The configuration of the Site in the json format

Returns: The response json contains, success in case of success or failure message in case of failure.

Request body:

```
{
  "Command": "ConfigFetch",
  "serverConfigLastUpdatedTime": <ServerConfigUpdatedLastTime>, "SiteDetails":
  {
    "siteName": <SiteName>,
    "siteLabel": <SiteLabel>,
    "siteConfigLastUpdatedTime": <SiteConfigUpdatedLastTime>,
    "siteId": <site id>,
    "monitoringId": <monitoring id>,
      “customparameters” :{}
  },
  "CollectorDetails":[ { “CollectorType”: <>,
    “CollectorName”: <>,
    "ConfiguredChannels": <>,
    "PollingStep": <polling step>,
    "ChecksumStatusBit": <checksum bit>,
    "Address": <address>,
    "HeartBeat": <heartbeat>,
    "DataFormatBits": "00",
    "Port": <port>,
    "CommunicationTimeOut": <communication bit>
    “customparameters” :{}
 }],
  "configJson": {
    "monitoringType": {
      "required": "True",
      "padding": "-",
      "start_pos": 55,
      "end_pos": 64,
      "type": "string",
      "alignment": "left"
    },
    "monitoringId": {
      "required": "True",
      "padding": "-",
      "start_pos": 65,
      "end_pos": 84,
      "type": "string",
      "alignment": "left"
```

```
    },
    "QualityCode": {
      "required": "True",
      "padding": "*",
      "start_pos": 42,
      "end_pos": 43,
      "type": "string",
      "alignment": "left"
    },
    "SensorTime": { "required":
      "True",
      "padding": "-",
      "start_pos": 44,
      "end_pos": 54,
      "type": "string",
      "alignment": "left"
    },
    "parameterId": {
      "required": "True",
      "padding": "-",
      "start_pos": 85,
      "end_pos": 100,
      "type": "string",
      "alignment": "left"
    },
    "parameterName": {
      "required": "True",
      "padding": "*",
      "start_pos": 11,
      "end_pos": 25,
      "type": "string",
      "alignment": "left"
    },
  "Reading": {
      "required": "True",
      "padding": "*",
      "start_pos": 26,
      "end_pos": 41,
      "type": "string",
      "alignment": "left"
    },
    "id": {
         "required": "True",
          "padding": "-", "start_pos":
      1,
      "end_pos": 8, "type":
      "string", "alignment":
      "left"
    },
    "sensorChannel": {
      "required": "True",
      "padding": "-",
      "start_pos": 9,
      "end_pos": 10,
      "type": "string",
      "alignment": "left"
```

```
        },
        "analyzerId": { "required":
          "True",
          "padding": "-",
          "start_pos": 101,
          "end_pos": 115,
          "type": "string",
          "alignment": "left"
        }
    },
    "AcquisitionSystemDetails": { "AcquisitionVersion":
      <Version Number>, "AcquisitionSystem": <Acquisition
      System Name>
    },
    "SensorA": {
      "collectorType": <Monitoring Type>,
      "monitoringType": <Monitoring Type>,
      "monitoringId": <Monitoring Id>,
      "ChannelNo": "0",
      "GaugeMinimum": "",
      "CoefficientA": "", "parameterId":
      <parameter id>,
      "GaugeMaximum": "",
      "MeasurementUnit": <measurement unit>,
      "compPort": "",
      "parameterName": <parameter name>,
      "CoefficientB": "",
      "analyzerId": <analyzer id>
      “customparameters” :{}
    },
.
.
    "SensorN": {
      "monitoringType": <Monitoring Type>,
      "monitoringId": <Monitoring Id>,
      "ChannelNo": "0",
      "GaugeMinimum": "",
      "CoefficientA": "1",
      "parameterId": <parameter id>,
      "GaugeMaximum": "",
      "MeasurementUnit": <measurement unit>,
      "compPort": "",
      "parameterName": <parameter name>,
      "CoefficientB": "0",
      "analyzerId": <analyzer id>,
      “customparameters” :{}
    }
}
```


Response for the Request will be

Success status

```
{
   "status": "Success",
   "configUpdateStatus": "Received Site configuration successfully"
}
```


Failure status


```
{
   "status": "Failed",
   "configUpdateStatus": "Failed to receive Site Configuration. Please retry"
}
```


Configuration Update Acknowledgement

Whenever the site client software has received the configuration from the Central Server Software and successfully update the site configuration, the client software should provide acknowledgement to the Central Server to ensure that server doesn't request for configuration update again.

This method gives the status of calibration.
http://ipaddress:port/MPCBServer/completedConfig

Path: **completedConfig**

Method: **POST**

Parameter: The site id and monitoring id will be passed as the parameter.

Returns: The response json contains, success in case of success or failure message in case of failure.

Request body:


```
{
   "siteId": <site-id>, "monitoringid":
   <monitor-id>, "ConfigUpdated":
   "True"
}
```


Response to Configuration acknowledgement received by client software

Success Response

```
{
   "status": "Success",
   "calibrationUpdateStatus": "Server and Site Configuration Synchronized"
}
```


Failure response


```
{
   "status": "Failed",
   "calibrationUpdateStatus": "Failed to update Configuration status"
}
```


Remote Calibration Service

This method download the configuration required for calibration.

http://ipaddress:port/MPCBServer/getCalibrationConfig

Path: getCalibrationConfig

Method: POST

Parameter: The site id, monitoring id, CalibrationType will be passed as the parameter. CalibrationType will be "scheduled" when a schedule is submitted to client or "immediate" if an immediate request for calibration is required.

Returns: The response json contains the configuration required for calibration

Request body:


```
{
   "siteId": <site-id>, "monitoringid":
   <monitor-id>,
    "CalibrationType": "Scheduled" or "Immediate"
}
```


Response provided by the Server will have the following fields

If any analyser maker needs any additional fields for performing, remote calibration, this can be discussed with MPCB Online Monitoring team at onlinecems.support@mpcb.gov.in and can use "customparameters" tag in the json

The configuration details has the sequence for calibrations, the required parameters for calibrations and the schedule for the calibrations.

RESPONSE

```
{
   "status": "Success", "calibration": {
      "calibratorName": <calibrator-name>,
      "sequence": [
         {
            "function": <function name>, "duration_secs":
            <duration in seconds>, "gas": <gas>,
            "value": "0",
            "delay": <delay in minutes>,
            "sequenceName": <sequence name>,
            "duration": <duration in minute>, "type":
            <type of calibration>, "unit": <unit of
            gas>
         } .....................

      ],
      "siteName": <site name>,
      "monitoringType": <monitoring type>,
      "frequency": <frequency>, "analyzerId":
      <analyser id>, "parameterId": <parameter
      id>,
      "remoteCalibrationId": <remote calibration id>, "parameterName":
      "SO2",
      "cycleUnit": "1", "total_duration":
      <total duration>, "frequencyDay":
      <day>,
      "siteId": <site id>, "startTime": {
         "date": <date>,
         "time": <time>
      },
      "executeImmediate": "True",
      "day": <day>,
      "cycle": <cycle>,
      "frequencyTime": <frequency time>,
      "calibratorId": <calibration id>,
      "monitoringUnit": <monitoring unit>, "value":
      "",
      "channelNumber": <channel number>,
      "analyzerType": <analyser type>, "endTime": {
         "date": <date>,
         "time": <time>
      },
      "remoteCalibrationName": <remote calibration name>,
      "analyzerName": <analyser name>

   },
   "serverCalibrationLastUpdatedTime": <serverCalibrationLastUpdatedTime>,
   "siteCalibrationLastUpdatedTime": <siteCalibrationLastUpdatedTime>,
   "lastCalibratedOn": <lastCalibratedOn>,
   "siteId": <siteid>
}
```

Failure

```
{
    "status": "Failed. Calibration configuration not available"
}
```

Calibration Update Acknowledgement

Whenever the site client software has received the calibration sequence and has scheduled the calibration on the analyser, the calibration acknowledgement has to be provided to MPCB Central Server to ensure that server doesn't request for calibration configuration again.

This method gives the status of calibration.
http://ipaddress:port/MPCBServer/updateCalibrationConfig

Path: **updateCalibrationConfig**

Method: **POST**

Parameter: The site id and monitoring id will be passed as the parameter.

Returns: The response json contains, success in case of success or failure message in case of failure.

Request body:

```
{
  "siteId": <site-id>, "monitoringid":
  <monitor-id>,
  "CalibrationType": "Scheduled" or "Immediate"
}
```

Response to Calibration Update Received by Client Software

Success Response

```
{
  "status": "Success",
  "calibrationUpdateStatus": "Server and Site Calibration Synchronized"
}
```

Failure response

```
{
   "status": "Failed",
   "calibrationUpdateStatus": "Failed to update calibration configuration
status"
}
```

Fetch Diagnostic Information From Client

This method will be invoked by the client to upload the current diagnostic information in the analyser to the Central Server Software when the DiagnosticUpdateFlag is set to "True"

http://ipaddress:port/MPCBServer/uploadDiagnisticsInfo

Path: uploadDiagnosticInfo

Method: POST

Parameter: The diagnostic information of the Site in the json format

Returns: The response Json contains, success in case of success or failure message in case of failure. The diagnostics Json will be an array of key value pairs with the corresponding category associated with the key.

Request body:

```
{
   "Command": "DiagnosticFetch",
   "SiteDetails": {
      "siteName": <SiteName>,
      "siteLabel": <SiteLabel>,
      "siteConfigLastUpdatedTime": <SiteConfigUpdatedLastTime>,
      "siteId": <site id>,
      "monitoringId": <monitoring id>,
         "customparameters" :{}
   },
   "CollectorDetails":[ { "CollectorType": <>,
      "CollectorName": <>,
      "ConfiguredChannels": <>,
      "PollingStep": <polling step>,
      "ChecksumStatusBit": <checksum bit>,
      "Address": <address>,
      "HeartBeat": <heartbeat>,
      "DataFormatBits": "00",
      "Port": <port>,
      "CommunicationTimeOut": <communication bit>
      "customparameters" :{}
  }],
   "diagnosticJson": [{"analyserId":<analyser-id>,"parameterName":"",
diagnostics":[{"key":<key>, "value":<value>,"category":<category>}]}]

}
```

Response for the Request will be

Success status


```
{
   "status": "Success",
   "diagnosticUpdateStatus": "Received Site diagnostics successfully"
}
```


Failure status


```
{
   "status": "Failed",
   "diagnosticUpdateStatus": "Failed to receive Site diagnostics. Please retry"
}
```

Central Server Software Open API

.

Central Server Software Open API

Central Server Software Open API

Central Server Software Open API

Central Server Software Open API

Central Server Software Open API