API

Open API for Emission, Effluent, and Ambient Air Quality Monitoring

API Version : 1.0

Published Date: 14th Aug 2017

1. Introduction

Karnataka Pollution Control Board has established the Central Server Software for acquisition of data from Industries from the Online Monitoring of Emission, Effluent discharges and Ambient Air Quality Monitoring. This document outlines, the Open API published by Central Server Software for data integration directly from the Industries.

Key Requirements to be adhered to by Industry are

a. Data should be transmitted directly from the Analyzer using tamper proof datalogger without any intermediate PC or Server.
b. Data should be transmitted to the Karnataka Pollution Control Board over the Open API published.
c. Digital data output in the form of RS232/RS485/LAN etc from the analyser is preferred over 4-20mA based signals.

2. Procedure for Online Connectivity of CEMS, CEQMS and AQQMS at KSPCB

The following are the steps required to upload the data to KSPCB Central Server for Online Monitoring.

Step 1 : Visit the KSPCB website and read all the documents with respect to Uploading/Transmission of Continuous Monitoring data of Emission (CEMS) Effluent (EQMS) and Ambient (AAQMS) monitoring system to KSPCB at the following link

http://kspcb.kar.nic.in/onlinemonitoring.html

Step 2: Register the Industry details

In-case of any doubts or clarification required while filling the Industry details please email us attl.kspcb@nic.in.

Step 3 :KSPCB will provide you the connectivity credentials and other details including the security keys for establishing connectivity. Use this credentials and work with **any of the Open API Client Software** to upload the data to KSPCB Central Server. KSPCB does not endorse any client, Brand, Model of the sensors. It is the responsibility of the Industry to make sure that the chosen Sensors and Software are compliant with CPCB guidelines.

Step 4: Ensure the data is uploading continuously to KSPCB Central Server and send email to KSPCBtl.kspcb@nic.inabout connectivity establishment and completion of this activity.

3. Open API Overview

The open API will be based on secured https REST based protocol. During the initial period, http protocol will be enabled. However, it is expected that all interfaces are developed on https from a security perspective.

The fundamental principle for enabling REST based Open API is to ensure that multiple vendor software are able to integrated with the Central Server. However, it is noted, that strict action will be taken against any vendor transmitting data from their server instead of directly transmitting from the Industry site.

All the APIs are authenticated and hence ensure non-authenticated request are not sent to the Central Server. Any approved software client complying with the specified open API can upload the data to the Central Server.

4. Data Operations supported by the Open API

The following operations should be support by the client submitting data over Open API to KSPCB

- **Real Time Data Upload for CEMS, CEQMS and CAAQMS**
  - This API will be used to upload live data (with 1-minute frequency) to Central Server directly from the Analyser.

- **Delayed Data Upload (data during internet connectivity failure) for CEMS, CEQMS and CAAQMS**
  - This API should be used to upload live data that is acquired when the internet connectivity with Central Server is not available. In this case, live data is stored locally in the data logger in an encrypted form and

transmitted when internet connectivity is restored using the delayed upload API.

- **Analyzer Configuration Upload for CEMS, CEQMS and CAAQMS**
  - Once in every hour, the analyser configuration should be uploaded to Central Server. This should include all the configurations that can be changed by the operator of the Analyzer. All configuration with respect to range of the analyser, calibration linearization coefficients etc should be transmitted to Central Server every 1 hr.

- **Remote Analyzer Calibration API**
  - The remote calibration API enables regulator to trigger/initiate remote calibration on the Analyzer and view the details of calibration in real time. Th expectation is that regulator should be able to trigger the calibration on demand or should be able to see all calibration done periodically by the operator.

- **Analyzer Diagnostics API**
  - Any Error or diagnostic information reported at the analyser should be transmitted using the Analyzer diagnostic API. It is expected that every 1 hour the analyser diagnostics are captured and transmitted to Central Server for validation perspective.

5. Key Concepts related to Open API

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

- KSPCB ID: KSPCB ID is the unique ID assigned to each industry by KSPCB.

- Site ID: Unique Site ID identifying the specific industry. Industry is identified based on the Consent to Operate (CTO) provided by KSPCB. Each CTO is associated to a unique Site ID. So if a plant has multiple CTO, each of them has to be transmitted using a separate SiteID and the registration should also be done separately.

- Monitoring ID: Each Site has multiple monitoring stations where the analysers are installed. Each monitoring station will be assigned a unique monitoring ID within the Site. A site can have multiple monitoring station

- Analyser ID: Each analyser make and model will be assigned a unique Analyser ID
  This id is assigned by KSPCB based on the analyser make and model provided during registration of the industry.

- Parameter ID: Each monitored parameter will have a common unified ID across all industries. The parameter ID is unique to each of the parameters measured and will be provided by KSPCB.

6. Client Side Software Requirement

Each client software implementing the API should also comply with the below requirements.

- Client Software running inside Data Logger should ensure tamper free secured data is transmitted directly from the analyser.
- Each site client software has to collect the data from the analyser based on the poll frequency defined. Ideal frequency for data sampling from analyser is 10 second. Data transmission to the Central Server should be at 1 minute frequency. The raw data should be transmitted to the server along with the data quality code and captured timestamp.
- In-case of any linearization applied in the Client Side Software that should be transmitted as well.
- The transmitted data should be encrypted zipped data in ISO-7168 or CSV format.
- All API request data transfer should be using a REST Service over HTTPS protocol.
- The client site software should wait for successful upload and also read subsequent instructions (Remote calibration, Configuration upload, Diagnostics information etc.) from the Central Server Software
- On receiving instructions on Remote calibrations or Configuration upload, the site client software should invoke the Remote Calibration or Configuration upload services to upload/download the corresponding configurations.
- In-case of any communication failure or any delayed data transmitted beyond a period 15 minutes, site software should store the data the 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 client should transmit data captured directly from the analyser from the site location. Any data transmission from different location will be rejected by the server.
- All the requests from the client to the server should be authenticated requests only. Any unauthenticated requests will be discarded or not processed.

API

Basic Organization of API

http://<ipaddress:port>/KSPCB/...

The IP Address will be shared with the email sent to Industry once the registration is completed.

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 Upload	This is for uploading the configuration to the server.	/uploadConfig	POST
	When the ConfigurationUploadFlag is set to "True" in the response of the Realtime Upload or Delayed Upload, client needs to provide the current configurations set at the Analyser.	/uploadConfig	POST
Calibration Trigger	When the RemoteCalibrationUpdateFlag is set to "True", the client software should using this URL for downloading the configuration required for calibration. The remote calibration data and sequence should be updated to the Calibrator locally	/InitiateRemoteCalibration	POST
Calibration Trigger 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	/RemotCalibrationInitiated	POST
Diagnostic Upload service	When the DiagnosticUpdateFlag is set to "True", the client software should using this URL for uploading the diagnostic information including any internal state of the analyser as per the analyser make and model.	/uploadDiagnosticInfo	POST

API

Authentication Mechanism for the API

Authentication Mechanism details will be send directly to Industry or Software integration vendors once they have registered with KSPCB.

Key Details Required

- The data transmitted for real time upload and delayed upload will follow the ISO-7168 format. ISO-7168 format is adopted to ensure that the API is Open format with respect to data exchange.
- Similar to the data format, the authentication and encryption mechanism is standard open format. Some of the parameters like padding, block size, Initialization Vectors, Key Size etc, has been customized to provide robust security from a long-term perspective. Periodically, these parameters will be changed as we perceive security threats. so the client software should be developed with flexibility to ensure that encryption scheme can be altered with minimal configuration changes.

Steps to be done

Data Upload

- The data upload follows an ISO-7168 format zip file.
- The zip file upload to the server will be multipart/form-data format. The data should be sent in zip format. The uploaded zip file will have two files, namely 1. Data File, 2. Metadata File.
- The Metadata file will specify the file formats (ISO-7168) etc.
- The data file should comply with the same.
  1. ISO-7168 Data should have encrypted using the AES Algorithm using Data Encryption Key (Site Private key)
  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. The details of the Header are mentioned separately below.

Header request should follow the RSA encryption principles and will have the following content.

A. Timestamp
B. site_key→ Site ID or Unique Site Key
C. Authorization Data (RSA) → The authorization data consist of the following fields in order separated by the delimiter. The authorization data will be encrypted using the KSPCB Server RSA Public Key provided
   - site_key→ Unique Site Key or Site ID provided by the KSPCB for each site for authentication
   - software_version_id→ Software version set by the Central Server
   - time_stamp_data→ Timestamp when the data was encrypted

D Signature (RSA)

The signature will be encrypting the authorization data using the Digital Signature of the Industry to ensure that the data is sent by the industry. In-case digital signature is not available industry can use the Site-specific RSA Key generated by KSPCB.

Data Upload

The section below describes the request response for each of the API Request.

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 CSV file format.

The zip file upload to the server will be multipart/form-data format. The data should be sent in 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 using AES Encryption**. 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) etc. and the data file should comply with the same. This gives flexibility to support different file formats based on the analyser or client software capability.

However, all files has 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

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”**

http://ipaddress:port/KSPCBServerIP/realtimeUpload OR
http://ipaddress:port/KSPCBServerIP/delayedUpload

Path:realtimeUpload or 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 2 minutes. Anything delayed should be uploaded to delayedUpload URL

API

If the upload is success, the following response will be obtained.


```
{
  "status": "Success",
  "serverConfigLastUpdatedTime": "<time>",
  "ConfigurationDownloadFlag": "<Flag>",
  "ConfigurationUploadFlag": "<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",
  "ConfigurationUploadFlag": "False",
  "RemoteCalibrationUpdateFlag ": "True",
  "DiagnosticUpdateFlag": "False",
  "statusMessage": "file uploaded successfully. "
}
```


If the upload is a failure the following response will be obtained.


```
{
  "status": "Failed",
  "statusMessage": "No files were uploaded."
}
```

API

Configuration Upload to Server

Upload 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/KSPCBServer/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"

}

Remote Calibration Service

This method download the configuration required for calibration.

http://ipaddress:port/KSPCBServer/initiateRemoteCalibration

Path: **InitiateRemoteCalibration**

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 glens team at glens@knowledgelens.com **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"
}
```

API

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 KSPCB server to ensure that server doesn't request for calibration configuration again.

This method gives the status of calibration.
http://ipaddress:port/KSPCBServer/RemoteCalibrationInitiated

Path: RemoteCalibrationInitiated

Method: POST

Parameter: The site id, monitoring id, parameter 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/KSPCBServer/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 pair with the corresponding category associated to 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"

}