# CGBAS PRO OPEN API DOCUMENT

## Ⅰ. Context

This document is designed to provide a unified interface invocation and interaction specification for API access.

## Ⅱ. Open API Invocation Instructions

### 1. Preparation

Please contact the CGBAS PRO system administrator to create AK/SK key pair before invoking the API.

Open API is not allowed to access to non-business modules such as Settings, Help, Notice, etc. with the same permissions as the API-bound user.

This system provides AK/SK authentication method, which is to use AK/SK to sign the request and add the signature information to the message header at the time of the request to pass the authentication.

- AK(Access Key): Access Key. Unique identifier associated with the secret key; The access key ID is used with the secret key to encrypt the request.
- SK(Secret  Key): Secret  Key. The key used in conjunction with the access key to cryptographically sign the request, which identifies the sender and prevents the request from being modified. When using AK/SK authentication, you can sign the request using AK/SK based on the signature algorithm, refer **Signature Description**。

### 2. Interface Overview

(1) All API interfaces are HTTP interfaces.

(2) The return data from all APIs is in JSON format.

(That is, the Content-Type response header of the API is fixed: application/json;charset=UTF-8)

(3) The encoding used for API interface request and response data is UTF-8.

### 3. Constructing Request

#### 3.1 Request URL

Request URL consists of the following parts: {Protocol}://{Endpoint}/{Resource-path}?{Query-params}

URL parameter description:

| Parameter | Description |
| --- | --- |
| Protocol | Indicates the protocol used to transmit requests: HTTP/HTTPS is supported, and the use of HTTPS is recommended for extranet access. |
| Endpoint | Specifies the server domain name or IP address hosting the REST service endpoint. |
| Resource-path | Resource path, i.e. API access path. Get from the URI module of the specific API. |
| Query-params | Query parameters are optional and are not available in every API. The query parameter needs to be preceded by a "?", in the form of "parameter name = parameter value". |

```
Ex: http://127.0.0.1:8080/openapi/stream/stations
```

## 3.2 Request Method

HTTP request methods (also known as operation or verb), It tells the service what type of operation you are requesting. This system uses the following request methods:

| Method | Description |
| --- | --- |
| GET | Request the server to return the specified resource. |
| PUT | Request the server to update the specified resource. |
| POST | Request the server to add a new resource or perform special operations. |
| DELETE | Request the server to delete a specified resource, such as an object. |

## 3.3 Request Header

Attach request header fields, such as those required by the specified URI and HTTP methods. For example, the request header "Content-Type" defines the type of message body, requesting authentication information, etc.

The detailed public request header field is as follows:

| Field | Description | Required | Example |
| --- | --- | --- | --- |
| Content-Type | The type (format) of the message body. Users are recommended to use the default value "application/json", other values will be specified in the specific interface. | YES | application/json |
| X-Nonce | A random string of characters, either as a pure word or as a combination of a word and a number. The greater the randomness, the better, and the X-Nonce is **different for each request**. | YES | weweiuon332hhe |
| X-Access-Key | Access Key ID. a unique identifier associated with secret key; the Access Key ID is used in conjunction with the secret key to cryptographically sign requests. | YES | demoKey |
| X-Sign-Method | Summary algorithm for signatures with an optional value of: HmacSHA1 , HmacSHA256 | NO | Recommended HmacSHA256, default use HmacSHA256, **to be consistent with the algorithm at the time of the sign** |
| X-Timestamp | Current UNIX timestamp, the number of **milliseconds** from 0:0:0 on January 1, 1970 to the present (meaning within 10 minutes before and after the instant the request was initiated). | YES | 1698591687000 |
| Sign | Signature parameter, which has **to be re-created for each request** and is not reusable. (See signature description) | YES | 1465ff84777488d9614c62 |
| Accept-Language | Specify the interface return language, currently only supports Chinese and English. Support only specify country, input unsupported options will return English. | NO | Specify Chinese -> zh-CN or zh,<br>specify English -> en-US or en. |

## 4. Signature Description

**The steps to create the signature parameter sign at API entry are as follows:**

(1) Assembled signature content: consists of "Request Method", "URL Resource Path", "Request Header Parameters" 3 parts, using space to separate.

- Request Method: HTTP request method, ex: GET, POST etc. Capitalization required.
- URL Resource Path: "Resource-path" section of the HTTP request URL (without query parameters), Starting with "/", for example: /openapi/stream/stations
- Request Header String: All parameters in the request header that **start with "X-"** are sorted by parameter name **to lowercase** and by ASCII. Splicing the sorted parameter key-value pairs with &: i.e., splicing: key1=val1&key2=val2&...

```
Ex: String headerStr = "GET /openapi/stream/stations x-access-key=123456&x-nonce=1&x-sign-method=HmacSHA1&x-timestamp=1698592692000";
```

(2) Prepare the HMAC hash key for the next step, i.e., use **SK** as the hash key directly in the API, assuming that the HMAC hash key obtained is **shaKey**.

HMAC-SHA (i.e., hmac-sha1/hmac-sha256) hashing is performed on **the string obtained in the previous step** using **shaKey** to obtain a byte array, which is expressed in pseudo-code, as follows

```
Ex: shaResultBytes = HMAC-SHA(headerStr, shaKey)
```

(3) The shaResultBytes obtained from the above logic is converted to a hexadecimal string, i.e., sign.

## 5. Calculate sign example (Java)

```java
import javax.crypto.Mac;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.GeneralSecurityException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Test {
    private static final char[] hex = "0123456789abcdef".toCharArray();
    private static final String ACCESS_KEY = "vt34w8bRcxYWLaYB";
    private static final String SECRET_KEY = "T1w3pVR1pOumFInN";
    private static final String SIGN_METHOD = "HmacSHA256";
    private static final String GET = "GET";
    private static final String STATION_URI = "/openapi/stream/stations";

    public static void main(String[] args) throws IOException {
        Map<String, String> headers = new HashMap<>();

        headers.put("X-Nonce", "1");
```

```
        headers.put("X-Access-Key", ACCESS_KEY);
        headers.put("X-Sign-Method", SIGN_METHOD);
        headers.put("X-Timestamp", String.valueOf(System.currentTimeMillis()));

        String s = calcSign(SECRET_KEY, SIGN_METHOD, GET, STATION_URI, headers);
        System.out.println("Counting the results of the sign: " + s);
    }

    /**
     * Calculating sign
     *
     * @param secretKey  accessKey is vt34w8bRcxYWLaYB, secretKey corresponding
is T1w3pVR1pOumFInN
     * @param signMethod Using HmacSHA256 encryption
     * @param method     Request Method
     * @param uri        Request URI
     * @param xHeaders   Request Headers
     * @return Result of calculating sign
     */
    private static String calcSign(String secretKey, String signMethod, String
method, String uri, Map<String, String> xHeaders) throws IOException {
        StringBuilder builder = new StringBuilder();
        builder.append(method).append(" ").append(uri).append(" ");
        List<Map.Entry<String, String>> headerEntries = new ArrayList<>
(xHeaders.entrySet());
        headerEntries.sort(Map.Entry.comparingByKey());
        for (Map.Entry<String, String> entry : headerEntries) {

 builder.append(entry.getKey().toLowerCase()).append("=").append(entry.getValue(
)).append("&");
        }
        builder.setLength(builder.length() - 1);
        String signData = builder.toString();
        byte[] shaResultBytes = encrypt(signMethod, signData, secretKey);
        return toHexString(shaResultBytes);
    }

    private static byte[] encrypt(String signMethod, String data, String secret)
throws IOException {
        try {
            SecretKey secretKey = new
SecretKeySpec(secret.getBytes(StandardCharsets.UTF_8), signMethod);
            Mac mac = Mac.getInstance(secretKey.getAlgorithm());
            mac.init(secretKey);
            return mac.doFinal(data.getBytes(StandardCharsets.UTF_8));
        } catch (GeneralSecurityException e) {
            throw new IOException(e.toString());
        }
    }

    private static String toHexString(byte[] bytes) {
        if (null == bytes) {
            return null;
        }

        StringBuilder sb = new StringBuilder(bytes.length << 1);

        for (byte aByte : bytes) {
```

```
            sb.append(hex[(aByte & 0xf0) >> 4])
                    .append(hex[(aByte & 0x0f)]);
        }

        return sb.toString();
    }
}
```

# 6. Response Instructions

## 6.1 Public Response Structure

```
{
    "code": "SUCCESS", // Interface request status code: SUCCESS, CGBAS0000XXX indicates a failure, see the table below for details
    "msg": null, // Error message when an interface request fails
    "data": {}   // Data returned
}
```

## 6.2 Public Response Code

| Code | Msg |
| --- | --- |
| SUCCESS | Request Success |
| CGBAS00000101 | Request expired |
| CGBAS00000102 | Request parameter is missing |
| CGBAS00000103 | Request duplicated, check x-nonce |
| CGBAS00000104 | Mismatch of counting results |
| CGBAS00000105 | Number of requests exceeded |
| CGBAS00000106 | API Key not exist |
| CGBAS00000999 | Other errors |