DTC eKYC OpenAPI Documentation

Digital Treasures Center @ 2025

Update History

Date	Author	Comment
28 Oct 2025	Nick	E-KYC Version 1.0.0
28 Apr 2026	Jensen	get-verification-url add on reversesolicitation parameter
18 May 2026	Zhi Yan	webhook signature format

Table of Contents

Chapter 1. Introduction

1.1. Overview

The DTC eKYC OpenAPI is a RESTful JSON-based web service that allows third parties to integrate DTC eKYC solutions into their platforms.

This specification defines a set of interfaces and security standards.

NOTE	**The time-related values are based on the Singapore Time (SGT) Timezone by default.**

1.2. Specification Titles

- **Name**: This column specifies the name of JSON field in request or response.
- **Max Length**: This column specifies the content and maximum length of value of JSON field.
  - **a** - Alpha
  - **n** - Numeric
  - **s** - Special printable character
- **Type**: This column specifies the type of JSON field.
  - **String** - the value is in string type;
  - **Nested JSON** - the value is a JSON object.
- **Required**: This column specifies the requirement of JSON field
  - **M** - Mandatory. DTC system will validate the value of this field in request
  - **C** - Conditional. In some conditions, this field is Mandatory. The condition will be specified in the description.
  - **O** - Optional. This field is optional, which will not affect the result of the transaction.
- **Description**: This column describes the usage, conditions or values contained of this JSON field.

1.3. Security Standards

The OpenAPI provides different level of security standards.

HTTPS: First level of security which is compulsory for all connections to DTC Wallet OpenAPI. TLS 1.2.

Whitelist: IP whitelist is optional, you can set it up in the `API Key Management` Page.

OAuth 2.0: Request session management.

Signature: For security checking.

Chapter 2. Quick Start

2.1. Disclaimer

Use of these APIs is subject to dtcpay's review and approval. dtcpay reserves the right, at its sole discretion, to approve, restrict, suspend, or revoke access at any time.

2.2. Before Integration

1. Request for account registration in DTC Wallet platform
2. Login with username and password issued by DTC team, go to the `API Key Management` Page.
3. Click `+ Create` to open the dialog.
4. Key in the Name and IP Whitelist then submit.
5. The `API Key`, `API Secret` and `Sign Key` will be generated and pop-up in dialog. Be noted, These values are sensitive and present **one-time**, please keep them safely **before** close this dialog.

2.3. OAuth 2.0 Authentication

2.3.1. Usage

To use OAuth 2.0, please use Fetch Access Token first.

Add `Authorization: Bearer {access_token}` in the other API requests.

If the access token expired, you can re-fetch it or use the Fetch Access Token by Refresh Token API. Or you can get a new access token before it expired.

NOTE	Only one access token can be exists in the same time per client-id, we suggest cache the access token in your caching system, for example, Redis.

2.3.2. Fetch Access Token

Request sample

```
curl -X "POST" "https://{host}/openapi/auth/oauth2/token" \
     -H 'Content-Type: multipart/form-data; charset=utf-8;
boundary=__X_PAW_BOUNDARY__' \ ①
     -F "client_id=xxx" \  ②
     -F "client_secret=xxx" \ ③
     -F "grant_type=client_credentials" ④
```

① The `Content-Type` should be `multipart/form-data`.

② The `API Key` value fetched from Before Integration.

③ The `API Secret` value fetched from Before Integration.

④ Fixed value `client_credentials`.

NOTE	The `client_id` and `client_secret` can be obtained via the web portal at https://business.dtcpay.com/login. Navigate to the **API** menu and create a **Wallet API** to generate the credentials.

Response sample (formatted)

```
{
    "access_token": "CkppLkyiqEUKITGdtCZRUZBl", ①
    "expires_in": 21600, ②
    "refresh_token": "CpCAhcCRtLU4kPjs54omWW3A", ③
    "rt_expires_in": 43200, ④
    "token_type": "bearer" ⑤
}
```

① For API calls, keep safely in your system. Any leakage will cause potential security risk. Please contact DTC if you feel it's compromised.

② The Access Token expiry time, in seconds.

③ To fetch a new access token before these two tokens are expired.

④ The Refresh Token will expiry, in seconds.

⑤ Fixed value `bearer`.

NOTE	When Access Denied (errCode:00006) occurs, it may be caused by a client_id/client_secret mismatch. It can also happen if the IP address configured during API key creation does not match the requesting IP address (the source IP of the client calling the API).

2.3.3. Fetch Access Token by Refresh Token

Request sample

```
curl -X "POST" "https://{host}/openapi/auth/oauth2/token" \
     -H 'Content-Type: multipart/form-data; charset=utf-8;
boundary=__X_PAW_BOUNDARY__' \ ①
     -F "refresh_token=CpCAhcCRtLU4kPjs54omWW3A" \  ②
     -F "grant_type=refresh_token" ③
```

① The `Content-Type` should be `multipart/form-data`.

② The `refresh_token` value fetched from the previous step.

③ Fixed value `refresh_token`.

Response sample (formatted)

```
{
    "access_token": "CE2ZgMjcgcfOvzY0jcS7csYe", ①
    "expires_in": 21600,
```

```
    "refresh_token": "CdugTgyDDa05uURhK6n6GYiW", ②
    "rt_expires_in": 43200,
    "token_type": "bearer"
}
```

① A new Access Token will be generated.

② A new Refresh Token will be generated.

2.4. Request Signature

For key function API requests, the signature are mandatory in the HTTP Header for security reason. You can apply to all the API requests as well.

1. Components of signature before hash:
    - `HTTP Method`: In uppercase.
    - `Timestamp`: Timestamp number in millis, **Singapore timezone**. The request will be failed if the timestamp is before or after **2 minutes**.
    - `URL`: The request URL without scheme and hostname.
    - `Querystring`: The in-URL parameters without the prefix ?, un-encoded.
    - `Request Body`: For string-based body, empty if not set. Must **exactly same** as the actually request body, including the invisible characters.
2. You need to combine these values by sequence, for example:

```
GET1636360576641/openapi/test?test=234
POST1636360661729/openapi/test{"a":"124"}
```

3. Generate the signature string

After you get the string to sign in the 2nd step, then you can generate the signature string by using **HMAC-SHA512** method and **Base64** encoding with the `Sign Key` from Before Integration.

For example:

- `Sign Key`: `aR2822Y6XbehWMclnB0Y2NJK`
- `String to sign`: `POST1636360661729/openapi/test{"a":"124"}`

Then the signature string shoule be:

```
MxrYnCm9Q7JOAvOrISf8+T2kuTW1d/w0at8aaPaoiX08VWfun3XPokVlIx1TkHXdcitls09wzfUGtXQZq23xdg==
```

4. Append signature to HTTP headers

Beside the `Authorization` header every request needed, the following headers are also need to add when perform a request:

- `D-TIMESTAMP`: The timestamp value should be same as Timestamp component in signature.

- `D-SIGNATURE`: The signature string generated.

NOTE

The body used for signature calculation is read by the server directly from the raw HTTP request (`HttpServletRequest`, via `IOUtils.toByteArray(request.getReader(), UTF-8)`). Therefore, the body in the signature must be **exactly the same** as the request body actually sent by the client, including spaces, line breaks, tabs, and any invisible characters. Even the slightest difference (e.g., an extra space) will cause signature verification to fail. The correct practice is to use the exact same raw request body string for both sending the request and generating the signature.

2.4.1. Example: GET Request (Java)

```
import com.google.common.hash.Hashing;

import java.nio.charset.StandardCharsets;
import java.util.Base64;

public class SignatureTestGuava {

    public static void main(String[] args) {
        // Shared secret key (Sign Key) provided during integration
        String signKey = "4mt0MUecOtOEDI42WjN4BmM0";

        // Example request details
        String method = "GET";
        String uri = "/openapi/v1/user";
        String query = "";
        String body = "";
        String timestamp = String.valueOf(System.currentTimeMillis());

        // Step 1: Build the stringToSign (method + timestamp + uri + query + body)
        String stringToSign = method + timestamp + uri;
        if (!query.isEmpty()) {
            stringToSign += "?" + query;
        }
        stringToSign += body;

        // Step 2: Generate HMAC-SHA512 signature with Sign Key, then Base64 encode it
        String signature = Base64.getEncoder().encodeToString(
                Hashing.hmacSha512(signKey.getBytes(StandardCharsets.UTF_8))
                        .hashBytes(stringToSign.getBytes(StandardCharsets.UTF_8))
                        .asBytes()
        );

        System.out.println("Timestamp   : " + timestamp);
        System.out.println("StringToSign: " + stringToSign);
        System.out.println("Signature   : " + signature);
    }
}
```

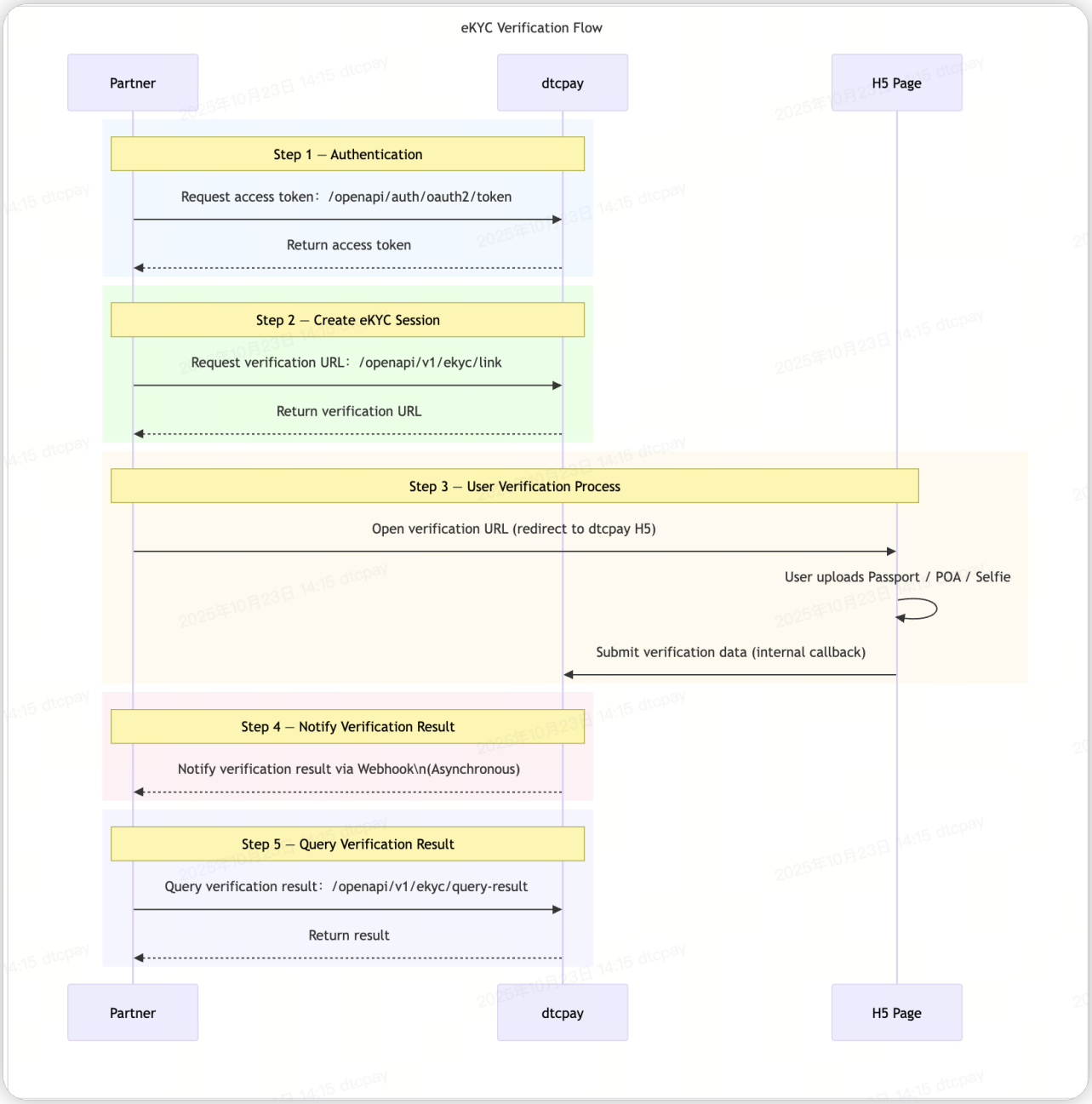
2.4.2. Example: POST Request (Java)

```
import com.google.common.hash.Hashing;

import java.nio.charset.StandardCharsets;
import java.util.Base64;

public class SignatureTestGuavaPOST {
    public static void main(String[] args) {
        // Shared secret key (Sign Key)
        String signKey = "4mt0MUecOtOEDI42WjN4BmM0";

        // Example request details
        String method = "POST";
        String uri = "/openapi/v1/otc/get-otc-rate";
        String query = "";
        String body = "{\"query\":{\"buyCurrency\":\"USD\",\"sellCurrency\":\"USDT\"
}}";

        // Current timestamp in milliseconds
        String timestamp = String.valueOf(System.currentTimeMillis());

        // Step 1: Build the stringToSign (method + timestamp + uri + query + body)
        String stringToSign = method + timestamp + uri;
        if (!query.isEmpty()) {
            stringToSign += "?" + query;
        }
        stringToSign += body;

        // Step 2: Generate HMAC-SHA512 signature with Sign Key, then Base64 encode it
        String signature = Base64.getEncoder().encodeToString(
                Hashing.hmacSha512(signKey.getBytes(StandardCharsets.UTF_8))
                        .hashBytes(stringToSign.getBytes(StandardCharsets.UTF_8))
                        .asBytes()
        );

        System.out.println("Timestamp   : " + timestamp);   // Value for D-TIMESTAMP
        System.out.println("StringToSign : " + stringToSign);// Raw string used for
signing
        System.out.println("Signature    : " + signature);   // Value for D-SIGNATURE

    }

}
```

Chapter 3. eKYC Process

3.1. Verification

This is used to obtain the eKYC verification URL.



Chapter 4. API

4.1. Get Verification Url

```
[POST] /openapi/v1/ekyc/get-verification-url
```

Request

Name	Length	Type	Required	Description	Remark
externalId	63	String	M	Unique applicant identifier as registered on your system.	
webHook	123	String	O	When the certification status changes, it will be notified via webhook.	
email	63	String	M	The applicant's email address	Example: user@example.com
mobile	63	String	M	The applicant's mobile number	Example: +60 123456789
countryOfResidence	3	String	M	Country code representing the applicant's country of residence.	Format: ISO 3166-1 alpha-3 (e.g. SGP, MYS, USA)
targetKycLevel	3	Integer	M	Target KYC level for the applicant.	Example: 100
reverseSolicitation	1	String	C	reverseSolicitation (Reverse Solicitation Flag) This field is required when the COR Country is not on the whitelist. It indicates whether the client falls under a "reverse solicitation" scenario. - T (True): Confirmed as reverse solicitation, meaning the client has proactively initiated and accepted the service. - F (False): Not a reverse solicitation case. Note: This field must be provided only when the COR Country is a non-whitelisted country/region. For whitelisted countries/regions, this field may be left blank or marked as not applicable.	Example: T

Response

Name	Description	Remark
url	The H5 page used for authentication	

Effective Request Body Sample

```
{
  "query": {
    "externalId": "demo_12345",
    "webHook": "https://partner.example.com/kyc/webhook",
    "email": "user@example.com",
    "mobile": "+60 123456789",
    "countryOfResidence": "SGP",
    "reverseSolicitation":"T",
    "targetKycLevel": 100 ①
  }
}
```

① Please refer to Enum List KycLevel for the meaning of each ID.

Response Sample

```
{
  "header":{
      "success": true,
      "result": {
        "url": "https://h5.example.com/openapi/ekyc-h5/?token=eyJhbGciOi..."
      }
  }
}
```

Invalid Request Body Sample

```
{
  "query": {
    "externalId": "demo_12345",
    "webHook": "https://partner.example.com/kyc/webhook",
    "email": "",
    "mobile": "+60 123456789",
    "countryOfResidence": "SGP",
    "targetKycLevel": 100 ①
  }
}
```

① Please refer to Enum List KycLevel for the meaning of each ID.

Response Sample

```
{
    "header": {
        "success": false,
        "errCode": "50002",
        "errMsg": "Please provide email in a correct format."
    }
```

```
}
```

Possible Error Codes

Code	Description
00010	Invalid parameters.
00025	Sorry, our services are currently unavailable in your country or region. We're continuously expanding, please check back for future updates.
01049	Your account is invalid. Please contact our support for further inquiries.
50001	Please provide mobile number in a correct format.
50002	Please provide email in a correct format.
50003	Please provide country of residence in a correct format.
50004	Please provide externalId in a correct format.
50005	Please provide targetKycLevel in a correct format.
50006	Mobile number exist, please choose a different mobile number.
50007	The applicant is currently undergoing verification.
50008	Email exist, please choose a different email.
50013	Reverse solicitation not declared by end user.
59999	Internal error. Please contact DTC.

4.2. Get Verification Result

```
[GET] /openapi/v1/ekyc/verification-result/{externalId}
```

Path Variables

No	Name	Type	Required	Description
1	externalId	String	M	Unique applicant identifier as registered on your system.

Response

Name	Description	Remark
externalId	Unique applicant identifier as registered on your system.	
dtcId	Unique applicant identifier as registered on DTC.	
clientStatus	The client status of the applicant.	
nationality	Nationality of the applicant.	Format: ISO 3166-1 alpha-3 (e.g. SGP, MYS, USA)

Name	Description	Remark
countryOfResidence	Country code representing the applicant's country of residence.	Format: ISO 3166-1 alpha-3 (e.g. SGP, MYS, USA)
verifyStatus	The KYC verify status of the applicant.	
currentKycLevel	Current KYC level for the applicant.	
passportVerifyStatus	The KYC review status of the applicant's passport.	
faceIdVerifyStatus	The KYC review status of the applicant's face ID.	
proofOfAddressVerifyStatus	The KYC review status of the applicant's proof of address.	

Response Sample

```
{
  "header": {
    "success": true
  },
  "result": {
    "externalId": "demo_12345",
    "dtcId": "214234123552",
    "clientStatus": 1, ①
    "nationality": "SGP",
    "countryOfResidence": "SGP",
    "verifyStatus": 1, ②
    "currentKycLevel": 100,
    "passportVerifyStatus": 3, ③
    "faceIdVerifyStatus": 3, ④
    "proofOfAddressVerifyStatus": 3 ⑤
  }
}
```

① Please refer to Enum List ClientStatus for the meaning of each ID.

② Please refer to Enum List VerifyStatus for the meaning of each ID.

③ Please refer to Enum List EKycFileVerifyStatus for the meaning of each ID.

④ Please refer to Enum List EKycFileVerifyStatus for the meaning of each ID.

⑤ Please refer to Enum List EKycFileVerifyStatus for the meaning of each ID.

Possible Error Codes

Code	Description
50010	The applicant information does not exist.
59999	Internal error. Please contact DTC.

4.3. Webhook

When the verification status changes, DTC will send a POST request to the webhook URL configured during API key creation (see Before Integration).

4.3.1. Signature Verification

The `signature` field in the webhook payload is generated using the same **HMAC-SHA512** method and **Base64** encoding as described in Request Signature, with the `Sign Key` from Before Integration.

The only difference is that the webhook signature does **not** include a timestamp.

4.3.2. Example: Webhook Signature Verification (Java)

```
import com.google.common.hash.Hashing;

import java.nio.charset.StandardCharsets;
import java.util.Base64;

public class WebhookSignatureVerification {
    public static void main(String[] args) {
        // Shared secret key (Sign Key)
        String signKey = "4mt0MUecOtOEDI42WjN4BmM0";

        // Webhook signature uses the same components as request signature,
        // but WITHOUT the timestamp.
        // Request signature: method + timestamp + uri + query + body
        // Webhook signature: method + uri + query + body

        String method = "POST";
        String uri = "/partner/webhook/endpoint";
        String query = "";
        String body = "{\"event\":\"KYC_VERIFICATION\",\"clientId\":852042402430001,
\"data\":{\"externalId\":\"demo_12345\",\"clientStatus\":1,\"nationality\":\"SGP\",\"c
ountryOfResidence\":\"SGP\",\"verifyStatus\":1,\"currentKycLevel\":100,\"passportVerif
yStatus\":3,\"faceIdVerifyStatus\":3,\"proofOfAddressVerifyStatus\":3}}";

        // The signature received in the webhook payload
        String receivedSignature =
"MxrYnCm9Q7JOAvOrISf8+T2kuTW1d/w0at8aaPaoiX08VWfun3XPokVlIx1TkHXdcitls09wzfUGtXQZq23xd
g==";

        // Step 1: Build the stringToSign (method + uri + query + body) — NO timestamp
        String stringToSign = method + uri;
        if (!query.isEmpty()) {
            stringToSign += "?" + query;
        }
        stringToSign += body;

        // Step 2: Generate HMAC-SHA512 signature with Sign Key, then Base64 encode it
```

```
        String expectedSignature = Base64.getEncoder().encodeToString(
                Hashing.hmacSha512(signKey.getBytes(StandardCharsets.UTF_8))
                        .hashBytes(stringToSign.getBytes(StandardCharsets.UTF_8))
                        .asBytes()
        );

        boolean isValid = expectedSignature.equals(receivedSignature);
        System.out.println("StringToSign : " + stringToSign);
        System.out.println("Expected     : " + expectedSignature);
        System.out.println("Received     : " + receivedSignature);
        System.out.println("Valid        : " + isValid);
    }
}
```

Webhook Request Sample

```
{
    "event":"KYC_VERIFICATION",
    "clientId":852042402430001,
    "signature":
"MxrYnCm9Q7JOAvOrISf8+T2kuTW1d/w0at8aaPaoiX08VWfun3XPokVlIx1TkHXdcitls09wzfUGtXQZq23xd
g==",
    "data":{
        "externalId": "demo_12345",
        "clientStatus": 1, ①
        "nationality": "SGP",
        "countryOfResidence": "SGP",
        "verifyStatus": 1, ②
        "currentKycLevel": 100, ③
        "passportVerifyStatus": 3, ④
        "faceIdVerifyStatus": 3, ⑤
        "proofOfAddressVerifyStatus": 3 ⑥
    }
}
```

① Please refer to Enum List ClientStatus for the meaning of each ID.

② Please refer to Enum List VerifyStatus for the meaning of each ID.

③ Please refer to Enum List KycLevel for the meaning of each ID.

④ Please refer to Enum List EKycFileVerifyStatus for the meaning of each ID.

⑤ Please refer to Enum List EKycFileVerifyStatus for the meaning of each ID.

⑥ Please refer to Enum List EKycFileVerifyStatus for the meaning of each ID.

Appendix A: Enum List

KycLevel

Name	ID	Descriptor
BASIC	100	Complete the Passport + PoA + Selfie review
STANDARD	200	Adding a questionnaire based on BASIC: Application Transaction Volume
ADVANCED	900	Based on the STANDARD foundation, we will enable the Stablecoin for the applicants.

VerifyStatus

Name	ID	Descriptor
Unverified	1	The applicant is not being verified.
Verifying	2	The applicant is under verifying.
Approved	3	The applicant has been approved.
Reject	4	The applicant's verification was rejected.

EKycFileVerifyStatus

Name	ID	Descriptor
UNVERIFIED	1	Document verification is not completed.
VERIFYING	2	Document verification is in progress.
VERIFY_SUCCESS	3	Document verification succeeded.
VERIFY_FAILURE	4	Document verification failed.

ClientStatus

Name	ID	Descriptor
SUSPENDED	0	Suspended
PENDING_KYC	1	Pending KYC
DORMANT	2	Dormant
REGISTERED	3	Self Registered
REJECTED	4	Rejected
OFF_BOARD	5	Off Board
REFERRER	8	Referrer
TERMINATED	9	Terminated
DEACTIVATED	10	Deactivated
RESTRICTED	11	Restricted
DELETED	12	Deleted
FROZEN	13	Frozen

Name	ID	Descriptor
DROP	14	Drop
ACTIVATED	99	Activated