# Node JS Version 1

Zoho CRM
--zoho.com/crm--

**Table of Contents**

# Overview

NodeJS SDK offers a way to create client NodeJS applications that can be integrated with Zoho CRM.

## Environmental Setup

NodeJS SDK is installable through npm. npm is a tool for dependency management in NodeJS. The SDK expects the following from the client app.

## Including the SDK in your project

You can include the SDK to your project using:

- Client app must have Node (version 9 and above)
- NodeJS SDK must be installed into client app through npm.
    - Install Node from nodejs.org (if not installed).
    - Install NodeJS SDK. Here's how you install the SDK:
        - Navigate to the workspace of your client app.
        - Run the command below:

```
i.   npm install zcrmsdk
```

The NodeJS SDK will be installed and a package named **zcrmsdk** will be created in the local machine.

Another method to install the SDK:

1. Add dependencies to the package.json of the node server with the latest version (recommended).
2. Run **npm install** in the directory which installs all the dependencies mentioned in package.json.

**Note**

- The **access and refresh tokens are environment-specific and domain-specific**. When you handle various environments and domains such as Production, Sandbox, or Developer and IN, CN, US, EU, or AU, respectively, you must use the access token and refresh token generated only in those respective environments and domains. The SDK throws an error, otherwise.
- For example, if you generate the tokens for your Sandbox environment in the CN domain, you must use only those tokens for that domain and environment. You cannot use the tokens generated for a different environment or a domain.
- For **Contact Roles and Records API**, you will need to provide the **ZohoCRM.settings.fields.ALL** scope along with the ZohoCRM.modules.ALL scope while generating the OAuthtoken. Otherwise, the system returns the **OAUTH-SCOPE-MISMATCH** error
- For **Related Records API**, the scopes required for generating OAuthtoken are **ZohoCRM.modules.ALL**, **ZohoCRM.settings.fields.ALL** and **ZohoCRM.settings.related_lists.ALL**. Otherwise, the system returns the **OAUTH-SCOPE-MISMATCH** error

## Configuration

Before you get started with creating your NodeJS application, you need to register your client and authenticate the app with Zoho.

Follow the below steps to configure the SDK.

1. Create an instance of the **Logger** Class to log exception and API information.

```
const {Logger, Levels}= require("zcrmsdk/routes/logger/logger");
    /*
    * Create an instance of Logger Class that takes two parameters
    * 1 -> Level of the log messages to be logged. Can be configured by typing Levels "." and choose any level from the list displayed.
    * 2 -> Absolute file path, where messages need to be
```

```
    logged.
6       */
7   let logger = Logger.getInstance(Levels.INFO,
    "/Users/user_name/Documents/nodejs_sdk_log.log");
```

2. Create an instance of **UserSignature** that identifies the current user.

```
1   const UserSignature = require(
    "zcrmsdk/routes/user_signature").UserSignature;
2   //Create an UserSignature instance that takes user Email as
    parameter
3   let user = new UserSignature("abc@zoho.com");
```

3. Configure the **API environment** which decides the domain and the URL to make API calls.

```
1   const USDataCenter = require(
    "zcrmsdk/routes/dc/us_data_center").USDataCenter;
2       /*
3        * Configure the environment
4        * which is of the pattern Domain.Environment
5        * Available Domains: USDataCenter, EUDataCenter,
    INDataCenter, CNDataCenter, AUDataCenter
6        * Available Environments: PRODUCTION(), DEVELOPER(),
    SANDBOX()
7       */
8   let environment = USDataCenter.PRODUCTION();
```

4. Create an instance of **OAuthToken** with the information that you get after registering your Zoho client.

```
1   const {OAuthToken, TokenType}= require(
    "zcrmsdk/models/authenticator/oauth_token");
2       /*
3        * Create a Token instance
4        * 1 -> OAuth client id.
5        * 2 -> OAuth client secret.
```

```
6       * 3 -> REFRESH/GRANT token.
7       * 4 -> token type.
8       * 5 -> OAuth redirect URL.
9      */
10 let token = new OAuthToken("clientId", "clientSecret",
   "REFRESH/ GRANT Token", TokenType.REFRESH/TokenType.GRANT,
   "redirectURL");
```

5. Create an instance of **TokenStore** to persist tokens used for authenticating all the requests.

```
1  const DBStore = require(
   "zcrmsdk/models/authenticator/store/db_store").DBStore;
2  const FileStore = require(
   "zcrmsdk/models/authenticator/store/file_store").FileStore;
3      /*
4      * DBStore takes the following parameters
5      * 1 -> DataBase host name. Default value "localhost"
6      * 2 -> DataBase name. Default  value "zohooauth"
7      * 3 -> DataBase user name. Default value "root"
8      * 4 -> DataBase password. Default value ""
9      * 5 -> DataBase port number. Default value "3306"
10     */
11 //let tokenstore = new DBStore();
12 let tokenstore = new DBStore("hostName", "dataBaseName",
   "userName", "password", "portNumber");
13 //let tokenstore = new
   FileStore("/Users/userName/Documents/nodejssdk-tokens.txt")
```

6. Create an instance of **SDKConfig** containing the SDK configuration.

```
1  const SDKConfigBuilder =
   require("zcrmsdk/routes/sdk_config_builder").MasterModel;
2      /*
```

```
     * autoRefreshFields
     * if true - all the modules' fields will be auto-refreshed in the background, every hour.
     * if false - the fields will not be auto-refreshed in the background. The user can manually delete the file(s) or refresh the fields using methods from ModuleFieldsHandler(utils/util/module_fields_handler.js)
     *
     * pickListValidation
     * A boolean field that validates user input for a pick list field and allows or disallows the addition of a new value to the list.
     * True - the SDK validates the input. If the value does not exist in the pick list, the SDK throws an error.
     * False - the SDK does not validate the input and makes the API request with the user's input to the pick list
     */
let sdkConfig = new SDKConfigBuilder().setPickListValidation(false).setAutoRefreshFields(true).build();
```

7. Set the absolute directory path to store user specific files containing module fields information in **resourcePath**

```
let resourcePath = "/Users/user_name/Documents/nodejs-app";
```

8. Create an instance of **RequestProxy** containing the proxy properties of the user.

```
const RequestProxy = require("zcrmsdk/routes/request_proxy").RequestProxy;
     /*
     * RequestProxy class takes the following parameters
     * 1 -> Host
     * 2 -> Port Number
     * 3 -> User Name. Default null.
```

```
7       * 4 -> Password. Default null
8       */
9  let requestProxy = new RequestProxy("proxyHost", 80,
   "proxyUser", "password");
```

9. Initialize the SDK and make API calls.

## Token Persistence

Token persistence refers to storing and utilizing the authentication tokens that are provided by Zoho. There are three ways provided by the SDK in which persistence can be utilized. They are DataBase Persistence, File Persistence and Custom Persistence.

### Implementing OAuth Persistence

Once the application is authorized, the OAuth access and refresh tokens can be used for subsequent user data requests to Zoho CRM. Hence, they need to be persisted by the client app.The persistence is achieved by writing an implementation of the inbuilt TokenStore Class, which has the following callback methods.

- **getToken(user, token)** - invoked before firing a request to fetch the saved tokens. This method should return an implementation of Token Class object for the library to process it.
- **saveToken(user, token)** - invoked after fetching access and refresh tokens from Zoho.
- **deleteToken(token)** - invoked before saving the latest tokens.
- **getTokens()** - The method to retrieve all the stored tokens.
- **deleteTokens()** - The method to delete all the stored tokens.

**Note**

- user is an instance of the UserSignature Class.
- token is an instance of the Token Class

## Database Persistence

If you want to use database persistence, you can use MySQL. The DB persistence mechanism is the default method.

- The database name should be **zohooauth**.
- There must be a table **oauthtokens** with columns
    - **id**(int(11))
    - **user_mail** (varchar(255))
    - **client_id** (varchar(255))
    - **refresh_token** (varchar(255))
    - **grant_token** (varchar(255))
    - **access_token** (varchar(255))
    - **expiry_time**(varchar(20))

## MySQL Query

```
1 create table oauthtoken(id int(11) not null auto_increment,
  user_mail varchar(255) not null, client_id varchar(255),
  refresh_token varchar(255), access_token varchar(255),
  grant_token varchar(255), expiry_time varchar(20), primary
  key (id));
2 alter table oauthtoken auto_increment = 1;
```

Here is the code to create a DBStore object:

```
1 const DBStore = require(
  "zcrmsdk/models/authenticator/store/db_store").DBStore;
2 /*
3 * DBStore takes the following parameters
4 * 1 -> DataBase host name. Default value "localhost"
5 * 2 -> DataBase name. Default  value "zohooauth"
6 * 3 -> DataBase user name. Default value "root"
7 * 4 -> DataBase password. Default value ""
```

```
 * 5 -> DataBase port number. Default value "3306"
 */

let tokenstore = new DBStore();

let tokenstore = new DBStore("hostName", "dataBaseName",
  "userName", "password", "portNumber");
```

## File Persistence

In case of file persistence, you can set up persistence the tokens in the local drive, and provide the absolute file path in the FileStore object. This file must contain the following:

- **user_mail**
- **client_id**
- **refresh_token**
- **access_token**
- **grant_token**
- **expiry_time**

Here is the code to create a FileStore object:

```
const FileStore = require(
  "zcrmsdk/models/authenticator/store/file_store").FileStore;
/*
* FileStore takes the following parameter
* 1 -> Absolute file path of the file to persist tokens
*/
let tokenstore = new
  FileStore("/Users/username/Documents/nodejs_sdk_tokens.txt"
  );
```

## Custom Persistence

To use Custom Persistence, you must extend the TokenStore class (zcrmsdk/models/authenticator/store/token_store) and override the methods.

Here is the code:

```
const TokenStore = require('zcrmsdk/models/authenticator/store/token_store').TokenStore;

class CustomStore extends TokenStore {

    constructor(){
        super();
    }

    /**
      *
      * @param {UserSignature} user A UserSignature class instance.
      * @param {Token} token A Token (zcrmsdk/models/authenticator/oauth_token) class instance.
      * @returns A Token class instance representing the user token details.
      * @throws {SDKException} if any error occurs.
      */
    getToken(user, token) {
        // Add code to get the token
        return null;
    }

    /**
      *
      * @param {UserSignature} user A UserSignature class instance.
      * @param {Token} token A Token (zcrmsdk/models/authenticator/oauth_token) class instance.
      * @throws {SDKException} if any error occurs.
      */
    saveToken(user, token) {
        // Add code to save the token
    }
```

```
30
31      /**
32        *
33        * @param {Token} token A Token
   (zcrmsdk/models/authenticator/oauth_token) class instance.
34        * @throws {SDKException} if any error occurs.
35        */
36      deleteToken(user, token) {
37          // Add code to delete the token
38      }
39
40      /**
41        * @returns {Array} - An array of Token class
   instances
42        * @throws {SDKException}
43        */
44      getTokens() {
45          //Add code to retrieve all the stored tokens.
46      }
47
48      /**
49        * @throws {SDKException}
50        */
51      deleteTokens() {
52          //Add code to delete all the stored tokens.
53      }
54  }
55  module.exports = {CustomStore}
```

## Register your Application

Before you get started with authorization and make any calls using the Zoho CRM APIs, you need to register your application with Zoho CRM.

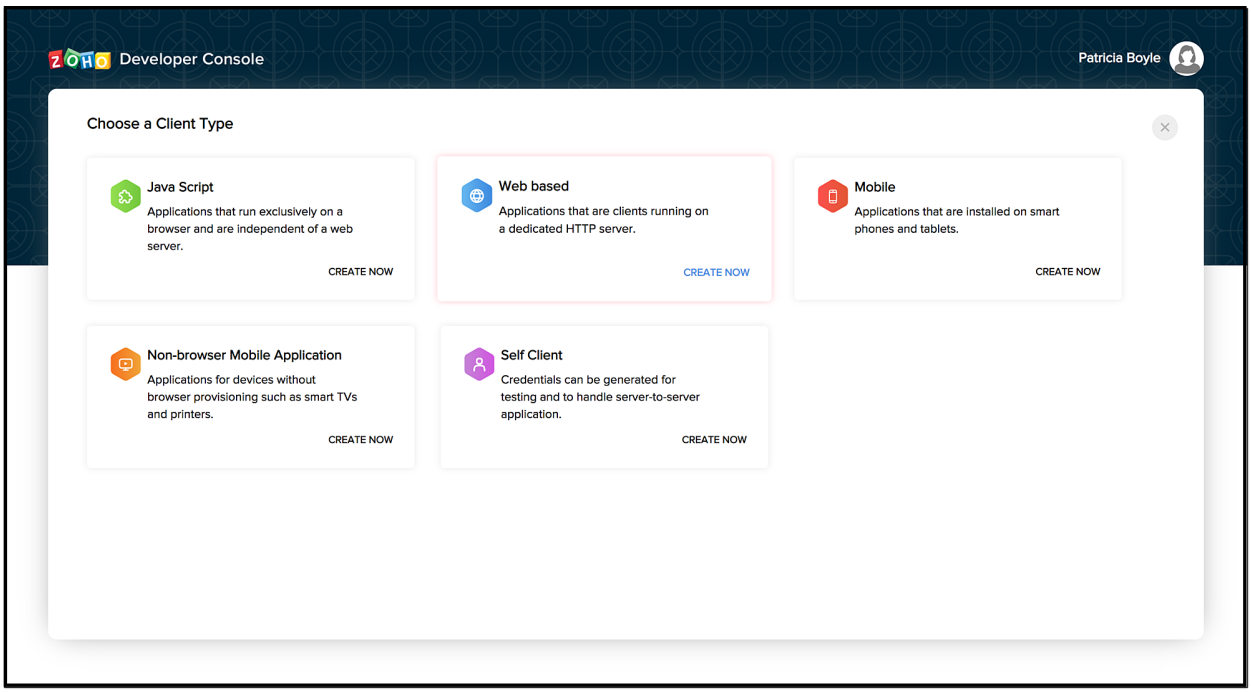
To register,

Zoho CRM

--zoho.com/crm--

- Go to Zoho Developer Console.
- Choose a client type:
    - Java Script: Applications that run exclusively on a browser and are independent of a web server.
    - Web Based: Applications that are clients running on a dedicated HTTP server.
    - Mobile: Applications that are installed on smart phones and tablets.
    - Non-browser Mobile Applications: Applications for devices without browser provisioning such as smart TVs and printers.
    - Self Client: Stand-alone applications that perform only back-end jobs (without any manual intervention) like data sync.

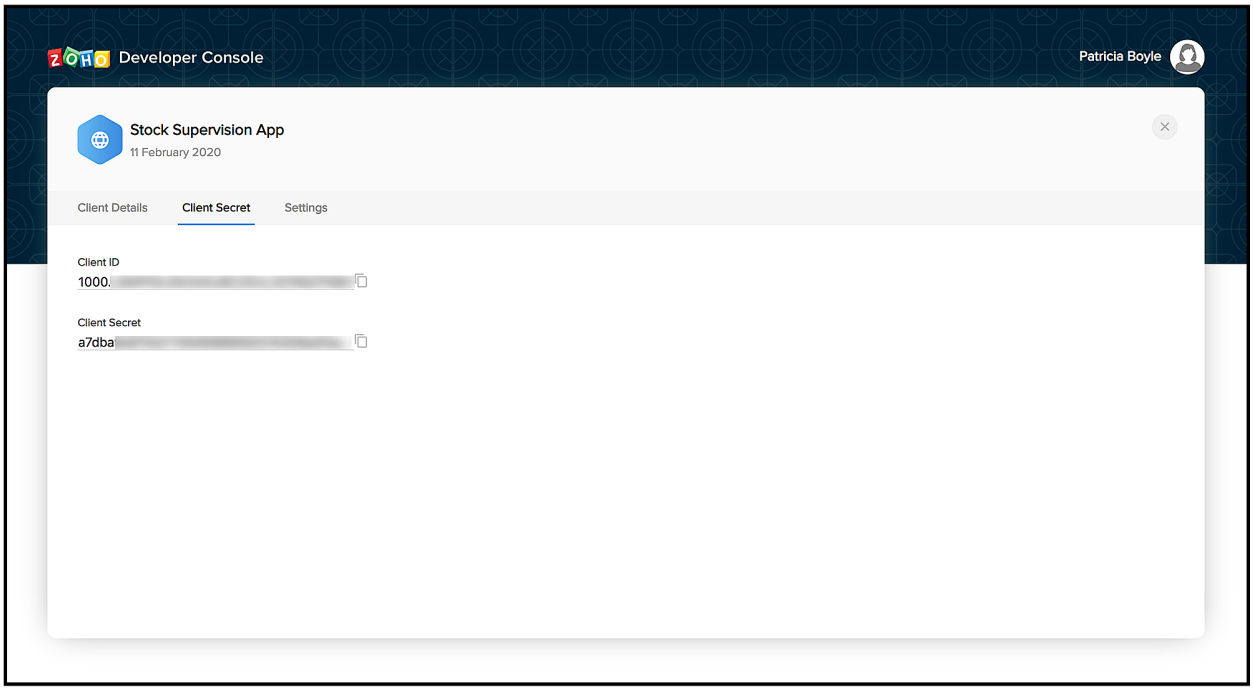
For more details, refer to OAuth Overview.




- Enter the following details:
    - **Client Name**: The name of your application you want to register with Zoho.
    - **Homepage URL**: The URL of your web page.
    - **Authorized Redirect URIs**: A valid URL of your application to which Zoho Accounts redirects you with a grant token(code) after successful

authentication.


- Click CREATE.
- You will receive the following credentials:
  - Client ID: The consumer key generated from the connected app.
  - Client Secret: The consumer secret generated from the connected app.



# Initializating the application

To access the CRM services through the SDK, you must first authenticate your client app.

> **Note**
> If you don't have a domain name and a redirect URL, you can use dummy values in their place and register your client.

## Generating the grant token

### *For a Single User*

The developer console has an option to generate grant token for a user directly. This option may be handy when your app is going to use only one CRM user's credentials for all its operations or for your development testing.

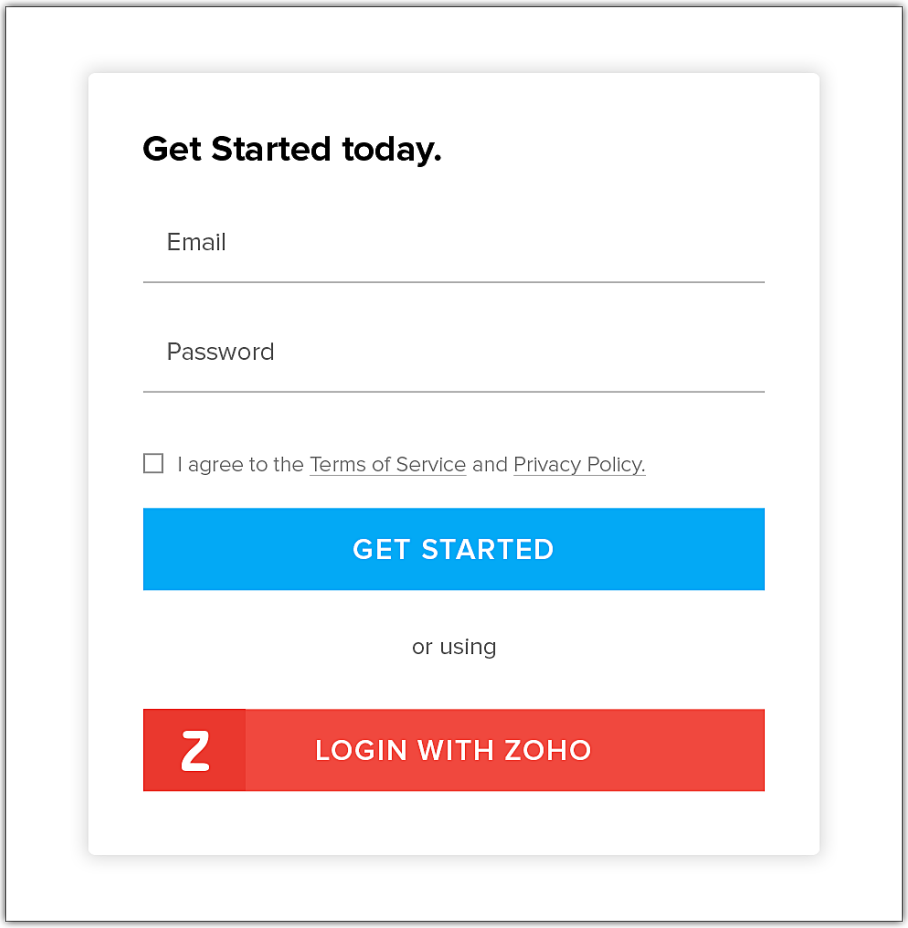
1. Login to your Zoho account.
2. Visit https://api-console.zoho.com
3. Click Self Client option of the client for which you wish to authorize.
4. Enter one or more (comma-separated) valid Zoho CRM scopes that you wish to authorize in the "Scope" field and choose the time of expiry.
5. Copy the grant token that is displayed on the screen.

> **Note**
> - The generated grant token is valid only for the stipulated time you chose while generating it. Hence, the access and refresh tokens should be generated within that time.
> - The OAuth client registration and grant token generation must be done in the same Zoho account's (meaning - login) developer console.

### *For Multiple Users*

For multiple users, it is the responsibility of your client app to generate the grant token from the users trying to login.

- Your Application's UI must have a "**Login with Zoho**" option to open the grant token URL of Zoho, which would prompt for the user's Zoho login credentials.




- Upon successful login of the user, the grant token will be sent as a param to your registered redirect URL.

**Note**

- The access and refresh tokens are environment-specific and domain-specific. When you handle various environments and domains such as Production, Sandbox, or Developer and IN, CN, US, EU, or AU, respectively, you must use the access token and refresh token generated only in those respective environments and domains. The SDK throws an error, otherwise.
- For example, if you generate the tokens for your Sandbox environment in the CN domain, you must use only those tokens for that domain and environment. You cannot use the tokens generated for a different environment or a domain.
- Initializing the SDK does not generate a token. A token is generated only when you make an API call.

## Initialization

```
const Initializer = require("zcrmsdk/routes/initializer").Initializer;
const {OAuthToken, TokenType} = require("zcrmsdk/models/authenticator/oauth_token");
const UserSignature = require("zcrmsdk/routes/user_signature").UserSignature;
const {Logger, Levels} = require("zcrmsdk/routes/logger/logger");
const USDataCenter = require("zcrmsdk/routes/dc/us_data_center").USDataCenter;
const DBStore = require("zcrmsdk/models/authenticator/store/db_store").DBStore;
const FileStore = require("zcrmsdk/models/authenticator/store/file_store").FileStore;
const RequestProxy = require("zcrmsdk/routes/request_proxy").RequestProxy;
const SDKConfigBuilder = require("zcrmsdk/routes/sdk_config_builder").MasterModel;


class Initializer{

    static async initialize(){

        /*
         * Create an instance of Logger Class that takes two parameters
         * 1 -> Level of the log messages to be logged. Can be configured by typing Levels "." and choose any level from the list displayed.
         * 2 -> Absolute file path, where messages need to be logged.
```

```
20          */
21         let logger = Logger.getInstance(Levels.INFO,
   "/Users/user_name/Documents/nodejs_sdk_log.log");
22
23         /*
24          * Create an UserSignature instance that takes user
   Email as parameter
25          */
26         let user = new UserSignature("abc@zoho.com");
27
28         /*
29          * Configure the environment
30          * which is of the pattern Domain.Environment
31          * Available Domains: USDataCenter, EUDataCenter,
   INDataCenter, CNDataCenter, AUDataCenter
32          * Available Environments: PRODUCTION(),
   DEVELOPER(), SANDBOX()
33          */
34         let environment = USDataCenter.PRODUCTION();
35
36         /*
37          * Create a Token instance
38          * 1 -> OAuth client id.
39          * 2 -> OAuth client secret.
40          * 3 -> REFRESH/GRANT token.
41          * 4 -> token type.
42          * 5 -> OAuth redirect URL. Default value is null
43          */
44         let token = new OAuthToken("clientId",
   "clientSecret", "REFRESH/ GRANT Token",
   TokenType.REFRESH/TokenType.GRANT, "redirectURL");
45
46        /*
47         * Create an instance of TokenStore.
48         * 1 -> DataBase host name. Default "localhost"
49         * 2 -> DataBase name. Default "zohooauth"
```

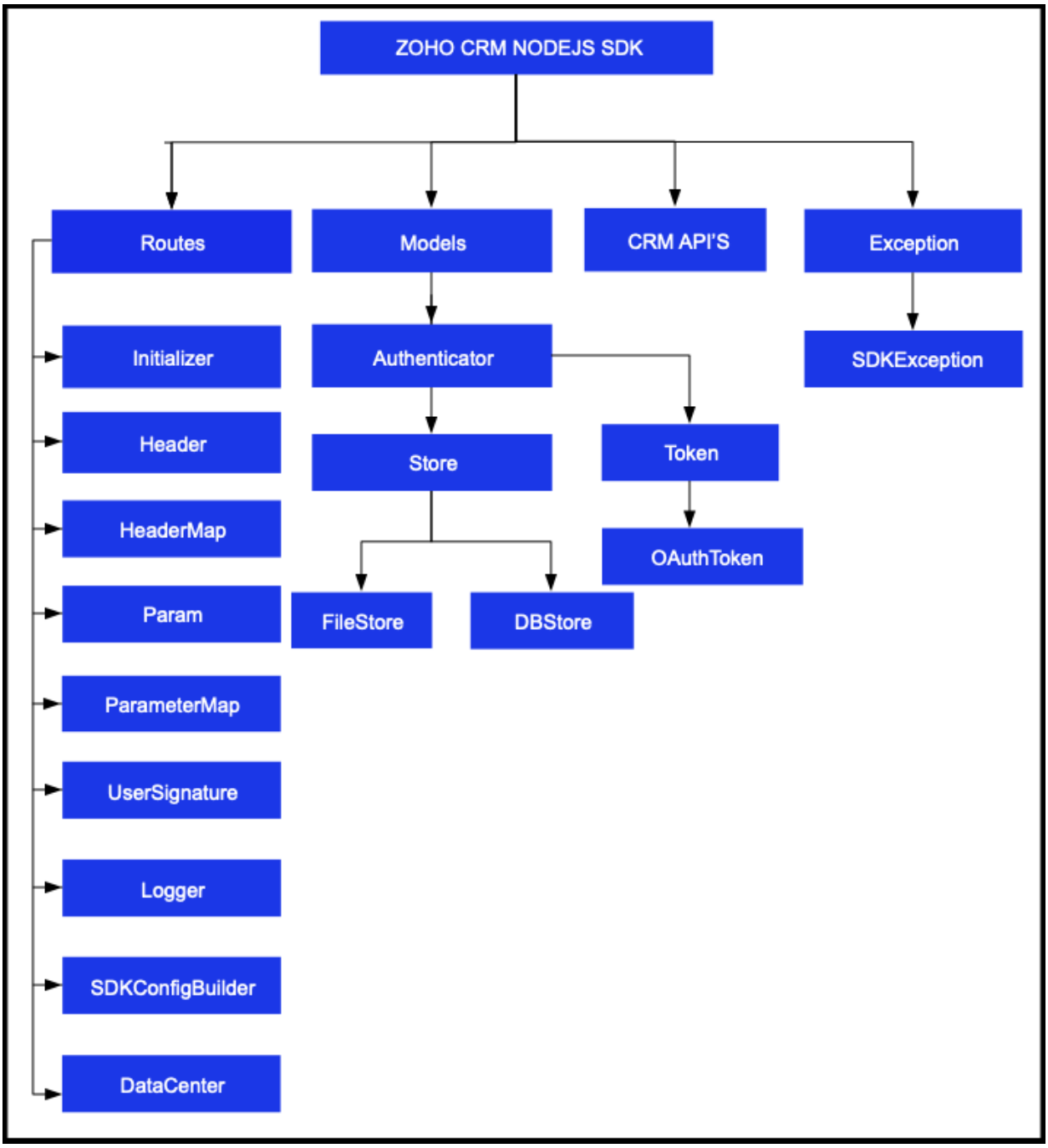
```
50          * 3 -> DataBase user name. Default "root"
51          * 4 -> DataBase password. Default ""
52          * 5 -> DataBase port number. Default "3306"
53          */
54          let tokenstore = new DBStore();
55
56          let tokenstore = new DBStore("hostName",
    "dataBaseName", "userName", "password", "portNumber");
57
58         /*
59          * Create an instance of FileStore that takes
    absolute file path as parameter
60          */
61          let tokenstore = new
    FileStore("/Users/username/Documents/nodejs_sdk_tokens.txt"
    );
62
63          /*
64           * autoRefreshFields
65           * true - all the modules' fields will be auto-
    refreshed in the background, every hour.
66           * false - the fields will not be auto-refreshed in
    the background. The user can manually delete the file(s) or
    refresh the fields using methods from
    ModuleFieldsHandler(utils/util/module_fields_handler.js)
67           *
68           * pickListValidation
69           * A boolean field that validates user input for a
    pick list field and allows or disallows the addition of a
    new value to the list.
70           * true - the SDK validates the input. If the value
    does not exist in the pick list, the SDK throws an error.
71           * false - the SDK does not validate the input and
    makes the API request with the user's input to the pick
    list
72           */
73          let sdkConfig = new
```

```
  SDKConfigBuilder().setPickListValidation(false).setAutoRefr
  eshFields(true).build();
74
75          /*
76           * The path containing the absolute directory path
  to store user specific JSON files containing module fields
  information.
77           */
78          let resourcePath =
  "/Users/user_name/Documents/nodejssdk-application";
79
80        /*
81         * Create an instance of RequestProxy class that
  takes the following parameters
82         * 1 -> Host
83         * 2 -> Port Number
84         * 3 -> User Name
85         * 4 -> Password
86         */
87          let proxy = new RequestProxy("proxyHost", 80);
88
89          let proxy = new RequestProxy("proxyHost", 80,
  "proxyUser", "password");
90
91          /*
92           * Call the static initialize method of Initializer
  class that takes the following arguments
93           * 1 -> UserSignature instance
94           * 2 -> Environment instance
95           * 3 -> Token instance
96           * 4 -> TokenStore instance
97           * 5 -> SDKConfig instance
98           * 6 -> resourcePath
99           * 7 -> Logger instance. Default value is null
100           * 8 -> RequestProxy instance. Default value is
  null
101           */
```

```
102         // The SDK can be initialized by any of the
  following methods
103         await Initializer.initialize(user, environment,
  token, tokenstore, sdkConfig, resourcePath)
104
105         await Initializer.initialize(user, environment,
  token, tokenstore, sdkConfig, resourcePath, logger, proxy);
106     }
107 }
108
109 Initializer.initialize()
```

## Class Hierarchy

All Zoho CRM entities are modeled as classes having members and methods applicable to that particular entity. The class hierarchy of various Zoho CRM entities in the Node JS SDK is depicted in the following image.




## Multi-User Support

The **NodeJS** SDK(from version 1.x.x) supports both single-user and multi-user app.

### Multi-user App

Multi-users functionality is achieved using the Initializer's static switchUser method.

```
//If proxy needs to be configured for the User
await Initializer.switchUser(user, environment, token, sdkConfig, requestProxy)
//Without proxy
await Initializer.switchUser(user, environment, token, sdkConfig)
```

Use the below code to remove a user's configuration from the SDK.

```
await Initializer.removeUserConfiguration(user, environment)
```

## Sample Multi-user code

```
const Initializer= require("zcrmsdk/routes/initializer").Initializer;
const {OAuthToken, TokenType}= require("zcrmsdk/models/authenticator/oauth_token");
const UserSignature = require("zcrmsdk/routes/user_signature").UserSignature;
const {Logger, Levels}= require("zcrmsdk/routes/logger/logger");
const USDataCenter = require("zcrmsdk/routes/dc/us_data_center").USDataCenter;
const EUDataCenter = require("zcrmsdk/routes/dc/eu_data_center").EUDataCenter;
const DBStore = require("zcrmsdk/models/authenticator/store/db_store").DBStore;
const FileStore = require("zcrmsdk/models/authenticator/store/file_store").FileStore;
const {RecordOperations, GetRecordsHeader, GetRecordsParam} = require("zcrmsdk/core/com/zoho/crm/api/record/record_operations");
const ParameterMap =
```

```
require("zcrmsdk/routes/parameter_map").ParameterMap;
const HeaderMap =
require("zcrmsdk/routes/header_map").HeaderMap;
const ResponseWrapper =
require("zcrmsdk/core/com/zoho/crm/api/record/response_wrap
per").ResponseWrapper;
const RequestProxy =
require("zcrmsdk/routes/request_proxy").RequestProxy;
const SDKConfigBuilder =
require("zcrmsdk/routes/sdk_config_builder").MasterModel;

class Record{

    static async call(){

        /*
         * Create an instance of Logger Class that takes
two parameters
         * 1 -> Level of the log messages to be logged. Can
be configured by typing Levels "." and choose any level
from the list displayed.
         * 2 -> Absolute file path, where messages need to
be logged.
         */
        let logger = Logger.getInstance(Levels.INFO,
"/Users/user_name/Documents/nodejs_sdk_log.log");

        /*
         * Create an UserSignature instance that takes user
Email as parameter
         */
        let user1 = new UserSignature("abc@zoho.com");

        /*
         * Configure the environment
         * which is of the pattern Domain.Environment
         * Available Domains: USDataCenter, EUDataCenter,
```

```
  INDataCenter, CNDataCenter, AUDataCenter
36          * Available Environments: PRODUCTION(),
  DEVELOPER(), SANDBOX()
37          */
38         let environment1 = USDataCenter.PRODUCTION();
39
40         /*
41          * Create a Token instance
42          * 1 -> OAuth client id.
43          * 2 -> OAuth client secret.
44          * 3 -> REFRESH/GRANT token.
45          * 4 -> token type.
46          * 5 -> OAuth redirect URL. Default value is null
47          */
48         let token1 = new OAuthToken("clientId1",
  "clientSecret1", "REFRESH/ GRANT Token",
  TokenType.REFRESH/TokenType.GRANT, "redirectURL");
49
50         /*
51          * Create an instance of TokenStore.
52          * 1 -> DataBase host name. Default "localhost"
53          * 2 -> DataBase name. Default "zohooauth"
54          * 3 -> DataBase user name. Default "root"
55          * 4 -> DataBase password. Default ""
56          * 5 -> DataBase port number. Default "3306"
57          */
58         let tokenstore = new DBStore();
59
60         let tokenstore = new DBStore("hostName",
  "dataBaseName", "userName", "password", "portNumber");
61
62         /*
63          * Create an instance of FileStore that takes
  absolute file path as parameter
64          */
65         let tokenstore = new
  FileStore("/Users/username/Documents/nodejs_sdk_tokens.txt"
```

```
);

        /*
        * autoRefreshFields
        * if true - all the modules' fields will be auto-refreshed in the background, every hour.
        * if false - the fields will not be auto-refreshed in the background. The user can manually delete the file(s) or refresh the fields using methods from ModuleFieldsHandler(utils/util/module_fields_handler.js)
        *
        * pickListValidation
        * A boolean field that validates user input for a pick list field and allows or disallows the addition of a new value to the list.
        * True - the SDK validates the input. If the value does not exist in the pick list, the SDK throws an error.
        * False - the SDK does not validate the input and makes the API request with the user's input to the pick list
        */
        let sdkConfig = new SDKConfigBuilder().setPickListValidation(false).setAutoRefreshFields(true).build();

        /*
         * The path containing the absolute directory path to store user specific JSON files containing module fields information.
         */
        let resourcePath = "/Users/user_name/Documents/nodejs-app";

        /*
         * Call the static initialize method of Initializer class that takes the following arguments
        * 1 -> UserSignature instance
```

```
87          * 2 -> Environment instance
88          * 3 -> Token instance
89          * 4 -> TokenStore instance
90          * 5 -> SDKConfig instance
91          * 6 -> resourcePath
92          * 7 -> Logger instance. Default value is null
93          * 8 -> RequestProxy instance. Default value is
  null
94          */
95         await Initializer.initialize(user1, environment1,
  token1, tokenstore, sdkConfig, resourcePath, logger);
96
97         await Record.getRecords("Leads");
98
99         await Initializer.removeUserConfiguration(user1,
  environment1);
100
101         let user2 = new UserSignature("abc2@zoho.eu");
102
103         let environment2 = EUDataCenter.SANDBOX();
104
105         let token2 = new OAuthToken("clientId2",
  "clientSecret2", "REFRESH/ GRANT Token",
  TokenType.REFRESH/TokenType.GRANT, "redirectURL");
106
107         let requestProxy = new RequestProxy("proxyHost",
  80, "proxyUser", "password");
108
109         let sdkConfig2 = new
  SDKConfigBuilder().setPickListValidation(true).setAutoRefre
  shFields(true).build();
110
111         await Initializer.switchUser(user2, environment2,
  token2, sdkConfig2, requestProxy);
112
113         await Record.getRecords("Leads");
```

```
114     }
115
116     static async getRecords(moduleAPIName){
117         try {
118             //Get instance of RecordOperations Class
119             let recordOperations = new RecordOperations();
120             let paramInstance = new ParameterMap();
121             await
  paramInstance.add(GetRecordsParam.APPROVED, "both");
122             let headerInstance = new HeaderMap();
123             await
  headerInstance.add(GetRecordsHeader.IF_MODIFIED_SINCE, new
  Date("2020-01-01T00:00:00+05:30"));
124             //Call getRecords method that takes
  paramInstance, headerInstance and moduleAPIName as
  parameters
125             let response = await
  recordOperations.getRecords(moduleAPIName, paramInstance,
  headerInstance);
126             if(response != null){
127
128                 //Get the status code from response
129                 console.log("Status Code: " +
  response.statusCode);
130                 if([204,
  304].includes(response.statusCode)){
131                     console.log(response.statusCode ==
  204? "No Content" : "Not Modified");
132                     return;
133                 }
134                 //Get the object from response
135                 let responseObject = response.object;
136                 if(responseObject != null){
137                     //Check if expected ResponseWrapper
  instance is received
138                     if(responseObject instanceof
  ResponseWrapper){
```

```
139                         //Get the array of obtained Record
  instances
140                         let records =
  responseObject.getData();
141                         for (let index = 0; index
  <records.length; index++) {
142                             let record = records[index];
143                             //Get the ID of each Record
144                             console.log("Record ID: " +
  record.getId());
145                             //Get the createdBy User
  instance of each Record
146                             let createdBy =
  record.getCreatedBy();
147                             //Check if createdBy is not
  null
148                             if(createdBy != null) {
149                                 //Get the ID of the
  createdBy User
150                                 console.log("Record
  Created By User-ID: " + createdBy.getId());
151
152                                 //Get the name of the
  createdBy User
153                                 console.log("Record
  Created By User-Name: " + createdBy.getName());
154
155                                 //Get the Email of the
  createdBy User
156                                 console.log("Record
  Created By User-Email: " + createdBy.getEmail());
157                             }
158
159                             //Get the CreatedTime of each
  Record
160                             console.log("Record
  CreatedTime: " + record.getCreatedTime());
```

```
161
162                             //Get the modifiedBy User
  instance of each Record
163                             let modifiedBy =
  record.getModifiedBy();
164
165                             //Check if modifiedBy is not
  null
166                             if(modifiedBy != null){
167
168                                 //Get the ID of the
  modifiedBy User
169                                 console.log("Record
  Modified By User-ID: " + modifiedBy.getId());
170
171                                 //Get the name of the
  modifiedBy User
172                                 console.log("Record
  Modified By User-Name: " + modifiedBy.getName());
173
174                                 //Get the Email of the
  modifiedBy User
175                                 console.log("Record
  Modified By User-Email: " + modifiedBy.getEmail());
176                             }
177
178                             //Get the ModifiedTime of each
  Record
179                             console.log("Record
  ModifiedTime: " + record.getModifiedTime());
180
181                             //Get the list of Tag instance
  each Record
182                             let tags = record.getTag();
183
184                             //Check if tags is not null
```

```
185                                 if(tags != null){
186
187                                     tags.forEach(tag => {
188                                         //Get the Name of each
    Tag
189                                         console.log("Record
    Tag Name: " + tag.getName());
190                                         //Get the Id of each
    Tag
191                                         console.log("Record
    Tag ID: " + tag.getId());
192                                     });
193                                 }
194
195                                 //Get all the values
196                                 let keyValues =
    record.getKeyValues();
197
198                                 let keyArray =
    Array.from(keyValues.keys());
199
200                                 for (let keyIndex = 0;
    keyIndex <keyArray.length; keyIndex++) {
201                                     const keyName =
    keyArray[keyIndex];
202
203                                     let value =
    keyValues.get(keyName);
204
205                                     console.log(keyName + " :
    " + value);
206                                 }
207                             }
208                         }
209                     }
210                 }
```

```
        } catch (error) {
            console.log(error);
        }
    }
}
Record.call();
```

1. The program execution starts from **call()**
2. The details of **user1** is given in the variables **user1, token1, environment1**.
3. Similarly, the details of another user **user2** is given in the variables **user2, token2, environment2**
4. Then, the **switchUser()** function is used to switch between the **User 1** and **User 2** as required.
5. Based on the latest switched user, the **Record.getRecords(moduleAPIName)** will fetch record.

## SDK Sample Code

```
const Initializer= require("zcrmsdk/routes/initializer").Initializer;
const {OAuthToken, TokenType}= require("zcrmsdk/models/authenticator/oauth_token");
const UserSignature = require("zcrmsdk/routes/user_signature").UserSignature;
const {Logger, Levels}= require( "zcrmsdk/routes/logger/logger");
const USDataCenter = require("zcrmsdk/routes/dc/us_data_center").USDataCenter;
const DBStore = require("zcrmsdk/models/authenticator/store/db_store").DBStore;
const FileStore = require("zcrmsdk/models/authenticator/store/file_store").FileStore;
const {RecordOperations, GetRecordsHeader, GetRecordsParam} = require("zcrmsdk/core/com/zoho/crm/api/record/record_operations");
const ParameterMap = require("zcrmsdk/routes/parameter_map").ParameterMap;
const HeaderMap = require("zcrmsdk/routes/header_map").HeaderMap;
const ResponseWrapper =
```

```
require("zcrmsdk/core/com/zoho/crm/api/record/response_wrapper").ResponseWrapper;
const SDKConfigBuilder = require("zcrmsdk/routes/sdk_config_builder").MasterModel;

class Record{

    static async getRecords(){

        /*
         * Create an instance of Logger Class that takes two parameters
         * 1 -> Level of the log messages to be logged. Can be configured by typing Levels "." and choose any level from the list displayed.
         * 2 -> Absolute file path, where messages need to be logged.
         */
        let logger = Logger.getInstance(Levels.INFO, "/Users/user_name/Documents/nodejs_sdk_log.log");

        /*
         * Create an UserSignature instance that takes user Email as parameter
         */
        let user = new UserSignature("abc@zoho.com");

        /*
         * Configure the environment
         * which is of the pattern Domain.Environment
         * Available Domains: USDataCenter, EUDataCenter, INDataCenter, CNDataCenter, AUDataCenter
         * Available Environments: PRODUCTION(), DEVELOPER(), SANDBOX()
         */
        let environment = USDataCenter.PRODUCTION();

        /*
         * Create a Token instance
         * 1 -> OAuth client id.
```

```
41          * 2 -> OAuth client secret.
42          * 3 -> REFRESH/GRANT token.
43          * 4 -> token type.
44          * 4 -> OAuth redirect URL. Default value is null
45          */
46         let token = new OAuthToken("clientId", "clientSecret",
   "REFRESH/ GRANT Token", TokenType.REFRESH/TokenType.GRANT);
47 
48         /*
49          * Create an instance of TokenStore.
50          * 1 -> DataBase host name. Default "localhost"
51          * 2 -> DataBase name. Default "zohooauth"
52          * 3 -> DataBase user name. Default "root"
53          * 4 -> DataBase password. Default ""
54          * 5 -> DataBase port number. Default "3306"
55          */
56         let tokenstore = new DBStore();
57 
58         let tokenstore = new DBStore("hostName", "dataBaseName",
   "userName", "password", "portNumber");
59 
60         /*
61          * Create an instance of FileStore that takes absolute
   file path as parameter
62          */
63         let tokenstore = new
   FileStore("/Users/username/Documents/nodejs_sdk_tokens.txt");
64 
65         /*
66         * autoRefreshFields
67         * if true - all the modules' fields will be auto-
   refreshed in the background, every hour.
68         * if false - the fields will not be auto-refreshed in the
   background. The user can manually delete the file(s) or refresh
   the fields using methods from
   ModuleFieldsHandler(utils/util/module_fields_handler.js)
69         *
70         * pickListValidation
71         * A boolean field that validates user input for a pick
   list field and allows or disallows the addition of a new value to
```

```
the list.
72          * True - the SDK validates the input. If the value does
not exist in the pick list, the SDK throws an error.
73          * False - the SDK does not validate the input and makes
the API request with the user's input to the pick list
74          */
75          let sdkConfig = new
SDKConfigBuilder().setPickListValidation(false).setAutoRefreshFie
lds(true).build();
76
77          /*
78           * The path containing the absolute directory path to
store user specific JSON files containing module fields
information.
79           */
80          let resourcePath = "/Users/user_name/Documents/nodejs-
app";
81
82          /*
83           * Call the static initialize method of Initializer class
that takes the following arguments
84           * 1 -> UserSignature instance
85           * 2 -> Environment instance
86           * 3 -> Token instance
87           * 4 -> TokenStore instance
88           * 5 -> sdkConfig instance
89           * 6 -> resourcePath
90           * 7 -> Logger instance
91           */
92          await Initializer.initialize(user, environment, token,
tokenstore, sdkConfig, resourcePath, logger);
93
94          try {
95              let moduleAPIName = "Leads";
96              //Get instance of RecordOperations Class
97              let recordOperations = new RecordOperations();
98              let paramInstance = new ParameterMap();
99              await paramInstance.add(GetRecordsParam.APPROVED,
"both");
100              let headerInstance = new HeaderMap();
101              await
```

```
    headerInstance.add(GetRecordsHeader.IF_MODIFIED_SINCE, new
    Date("2020-01-01T00:00:00+05:30"));
102             //Call getRecords method that takes paramInstance,
    headerInstance and moduleAPIName as parameters
103             let response = await
    recordOperations.getRecords(moduleAPIName, paramInstance,
    headerInstance);
104             if(response != null){
105
106                 //Get the status code from response
107                 console.log("Status Code: " +
    response.statusCode);
108                 if([204, 304].includes(response.statusCode)){
109                     console.log(response.statusCode == 204? "No
    Content" : "Not Modified");
110                     return;
111                 }
112                 //Get the object from response
113                 let responseObject = response.object;
114                 if(responseObject != null){
115                     //Check if expected ResponseWrapper instance
    is received
116                     if(responseObject instanceof
    ResponseWrapper){
117                         //Get the array of obtained Record
    instances
118                         let records = responseObject.getData();
119                         for (let index = 0; index <
    records.length; index++) {
120                             let record = records[index];
121                             //Get the ID of each Record
122                             console.log("Record ID: " +
    record.getId());
123                             //Get the createdBy User instance of
    each Record
124                             let createdBy =
    record.getCreatedBy();
125                             //Check if createdBy is not null
126                             if(createdBy != null) {
127                                 //Get the ID of the createdBy
    User
```

```
                                    console.log("Record Created By User-ID: " + createdBy.getId());
                                    //Get the name of the createdBy User
                                    console.log("Record Created By User-Name: " + createdBy.getName());
                                    //Get the Email of the createdBy User
                                    console.log("Record Created By User-Email: " + createdBy.getEmail());
                                }
                                //Get the CreatedTime of each Record
                                console.log("Record CreatedTime: " + record.getCreatedTime());
                                //Get the modifiedBy User instance of each Record
                                let modifiedBy = record.getModifiedBy();
                                //Check if modifiedBy is not null
                                if(modifiedBy != null){
                                    //Get the ID of the modifiedBy User
                                    console.log("Record Modified By User-ID: " + modifiedBy.getId());
                                    //Get the name of the modifiedBy User
                                    console.log("Record Modified By User-Name: " + modifiedBy.getName());
                                    //Get the Email of the modifiedBy User
                                    console.log("Record Modified By User-Email: " + modifiedBy.getEmail());
                                }
                                //Get the ModifiedTime of each Record
                                console.log("Record ModifiedTime: " + record.getModifiedTime());
                                //Get the list of Tag instance each Record
                                let tags = record.getTag();
```

```
                                //Check if tags is not null
                                if(tags != null) {
                                    tags.forEach(tag => {
                                        //Get the Name of each Tag
                                        console.log("Record Tag
   Name: " + tag.getName());
                                        //Get the Id of each Tag
                                        console.log("Record Tag ID:
   " + tag.getId());
                                    });
                                }
                                //Get all the values
                                let keyValues =
   record.getKeyValues();
                                let keyArray =
   Array.from(keyValues.keys());
                                for (let keyIndex = 0; keyIndex
   <keyArray.length; keyIndex++) {
                                    const keyName =
   keyArray[keyIndex];
                                    let value =
   keyValues.get(keyName);

                                    console.log(keyName + " : " +
   value);
                                }
                            }
                        }
                    }
                } catch (error) {
                    console.log(error);
                }
            }
 }

 Record.getRecords();
```

**Record Response**


## Responses and Exceptions

All SDK methods return an instance of the **APIResponse** class.

After a successful API request, the **getObject()** method returns an instance of the **ResponseWrapper** (for GET) or the **ActionWrapper** (for POST, PUT, DELETE).

Whenever the API returns an error response, the **getObject**() returns an instance of **APIException** class.

**ResponseWrapper** (for GET requests) and **ActionWrapper** (for **POST, PUT, DELETE** requests) are the expected objects for Zoho CRM APIs' responses.

However, some specific operations have different expected objects, such as the following:

- Operations involving records in Tags
  -RecordActionWrapper
- For getting Record Count for a specific Tag operation
  -CountWrapper
- For operations involving BaseCurrency
  -BaseCurrencyActionWrapper

- For Lead convert operation
  -ConvertActionWrapper
- For retrieving Deleted records operation
  -DeletedRecordsWrapper
- For Record image download operation
  -FileBodyWrapper
- For MassUpdate record operations
  -MassUpdateActionWrapper
  -MassUpdateResponseWrapper

## For GET Requests

The getObject() returns an instance of one of the following classes, based on the return type.

For application/json responses

- **ResponseWrapper**
- **CountWrapper**
- **DeletedRecordsWrapper**
- **MassUpdateResponseWrapper**
- **APIException**

## For POST, PUT, DELETE Requests

The getObject() returns an instance of one of the following classes

- **ActionWrapper**
- **RecordActionWrapper**
- **BaseCurrencyActionWrapper**
- **MassUpdateActionWrapper**
- **ConvertActionWrapper**
- **APIException**

These wrapper classes may contain one or an array of instances of the following

classes, depending on the response.

- **SuccessResponse** Class, if the request was successful.
- **APIException** Class, if the request was erroneous.

For example, when you insert two records, and one of them was inserted successfully while the other one failed, the ActionWrapper will contain one instance each of the SuccessResponse and APIException classes.

All other exceptions such as SDK anomalies and other unexpected behaviours are thrown under the **SDKException** class.

## Sample Codes

All of Zoho CRM's APIs can be used through the Node JS SDK, to enable your custom application to perform data sync to the best degree. Here are the sample codes for all the API methods available in our SDK.

**Attachment Operations**

| **Constructor** | **Description** |
|---|---|
| AttachmentsOperations(moduleAPIName, recordId) | Creates an AttachmentsOperations class instance with the moduleAPIName and recordId. |

| **Method** | **Description** |
|---|---|
| getAttachments | To fetch the list of attachments of a record. |
| uploadAttachments | To upload attachments to a record. |

| | |
|---|---|
| deleteAttachments | To delete the attachments that were added to a record. |
| deleteAttachment | To delete an attachment that was added to a record. |
| downloadAttachment | To download an attachment that was uploaded to a record. |
| uploadLinkAttachments | To upload a link as an attachment to a record. |

**Blueprint Operations**

| **Constructor** | **Description** |
|---|---|
| BluePrintOperations(recordId, moduleAPIName) | Creates a BluePrintOperations class instance with the recordId and moduleAPIName. |

| **Method** | **Description** |
|---|---|
| getBlueprint | To get the next available transitions for that record, fields available for each transition, current value of each field, and validation(if any). |
| updateBlueprint | To update a single transition at a time. |

**Bulk Read Operations**

| Method | Description |
| --- | --- |
| createBulkReadJob | To schedule a bulk read job to export records that match the criteria. |
| getBulkReadJobDetails | To know the status of the bulk read job scheduled previously. |
| downloadResult | To download the result of the bulk read job. The response contains a zip file. Extract it to get the CSV or ICS file depending on the "file_type" you specified while creating the bulk read job |

**Bulk Write Operations**

| Method | Description |
| --- | --- |
| uploadFile | To upload a CSV file in ZIP format. The response contains the "file_id". Use this ID while making the bulk write request. |
| createBulkWriteJob | To create a bulk write job to insert, update, or upsert records. The response contains the "job_id". Use this ID while getting the status of the scheduled bulk write job. |
| getBulkWriteJobDetails | To know the status of the bulk write job scheduled previously. |
| | To download the result of the bulk write |

| | |
|---|---|
| downloadResult | job. The response contains a zip file. Extract it to get the CSV or ICS file depending on the "file_type" you specified while creating the write job |

**Contact Roles Operations**

| **Method** | **Description** |
|---|---|
| getContactRoles | To get the list of all contact roles. |
| createContactRoles | To create contact roles. |
| updateContactRoles | To update contact roles. |
| deleteContactRoles | To delete contact roles. |
| getContactRole | To get specific contact role. |
| updateContactRole | To update specific contact role. |
| deleteContactRole | To delete specific contact role |

**Currencies Operations**

| **Method** | **Description** |
|---|---|

| | |
|---|---|
| getCurrencies | To get the list of all currencies available for your org. |
| addCurrencies | To add new currencies to your org. |
| updateCurrencies | To update the currencies' details of your org. |
| enableMultipleCurrencies | To enable multiple currencies for your org. |
| updateBaseCurrency | To update the base currency details of your org. |
| getCurrency | To get the details of specific currency. |
| updateCurrency | To update the details of specific currency. |

**Custom View Operations**

| **Constructor** | **Description** |
|---|---|
| CustomViewsOperations(module) | Creates a CustomViewsOperations class instance with the moduleAPIName. |

| **Method** | **Description** |
|---|---|
| getCustomViews | To get the list of all custom views in a module. |

| | |
|---|---|
| getCustomView | To get the details of specific custom view in a module. |

**Fields Metadata Operations**

| **Constructor** | **Description** |
|---|---|
| FieldsOperations(module) | Creates a FieldsOperations class instance with the module |

| **Method** | **Description** |
|---|---|
| getFields | To get the meta details of all fields in a module. |
| getField | To get the meta details of specific field in a module. |

**Files Operations**

| **Method** | **Description** |
|---|---|
| uploadFiles | To upload files and get their encrypted |

| | |
|---|---|
| | IDs. |
| getFile | To get the uploaded file through its encrypted ID |

**Layouts Operations**

| **Constructor** | **Description** |
|---|---|
| LayoutsOperations(module) | Creates a LayoutsOperations class instance with the moduleAPIName. |

| **Method** | **Description** |
|---|---|
| getLayouts | To get the details of all the layouts in a module. |
| getLayout | To get the details (metadata) of a specific layout in a module |

**Modules Operations**

| **Method** | **Description** |
|---|---|
| getModules | To get the details of all the modules. |
| getModule | To get the details (metadata) of a specific module. |
| updateModuleByAPIName | To update the details of a module by its |

| | |
|---|---|
| | API name. |
| updateModuleById | To update the details of a module by its ID |

**Notes Operations**

| **Method** | **Description** |
|---|---|
| getNotes | To get the list of notes of a record. |
| createNotes | To add new notes to a record. |
| updateNotes | To update the details of the notes of a record. |
| deleteNotes | To delete the notes of a record. |
| getNote | To get the details of a specific note. |
| updateNote | To update the details of an existing note. |
| deleteNote | To delete a specific note |

**Notification Operations**

| Method | Description |
|---|---|
| enableNotifications | To enable instant notifications of actions performed on a module. |
| getNotificationDetails | To get the details of the notifications enabled by the user. |
| updateNotifications | To update the details of the notifications enabled by a user. All the provided details would be persisted and rest of the details would be removed. |
| updateNotification | To update only specific details of a specific notification enabled by the user. All the provided details would be persisted and rest of the details will not be removed. |
| disableNotifications | To stop all the instant notifications enabled by the user for a channel. |
| disableNotification | To disable notifications for the specified events in a channel. |

**Organization Operations**

| Method | Description |
|---|---|
| getOrganization | To get the details of your organization. |

| | |
|---|---|
| uploadOrganizationPhoto | To upload a photo of your organization |

**Profile Operations**

| **Constructor** | **Description** |
|---|---|
| ProfilesOperations(ifModifiedSince) | Creates a ProfilesOperations class instance with the value of the If-Modified-Since header. |

| **Method** | **Description** |
|---|---|
| getProfiles | To get the list of profiles available for your organization. |
| getProfile | To get the details of a specific profile. |

**Query(COQL Operations)**

| **Method** | **Description** |
|---|---|
| getRecords | To get the records from a module through a COQL query |

**Records Operations**

| Method | Description |
| --- | --- |
| getRecord | To get a specific record from a module. |
| updateRecord | To update a specific record in a module. |
| deleteRecord | To delete a specific record from a module. |
| getRecords | To get all records from a module. |
| createRecords | To insert records in a module. |
| updateRecords | To update records in a module. |
| deleteRecords | To delete records from a module. |
| upsertRecords | To insert/update records in a module. |
| getDeletedRecords | To get the deleted records from a module. |
| searchRecords | To search for records in a module that match certain criteria, email, phone number, or a word. |
| convertLead | To convert records(Leads to Contacts/Deals). |
| getPhoto | To get the photo of a record. |
| uploadPhoto | To upload a photo to a record. |

| | |
|---|---|
| deletePhoto | To delete the photo of a record. |
| massUpdateRecords | To update the same field for multiple records in a module. |
| getMassUpdateStatus | To get the status of the mass update job scheduled previously |

**Related List Operations**

| **Constructor** | **Description** |
|---|---|
| RelatedListsOperations(module) | Creates a RelatedListsOperations class instance with the moduleAPIName |

| **Method** | **Description** |
|---|---|
| getRelatedLists | To get the details of all the related lists of a module. |
| getRelatedList | To get the details of a specific related list of a module |

**Related Records Operations**

| Constructor | Description |
| --- | --- |
| RelatedRecordsOperations(relatedListAPIName, recordId, moduleAPIName) | Creates a RelatedRecordsOperations class instance with the relatedListAPIName, recordId, and moduleAPIName |

| Method | Description |
| --- | --- |
| getRelatedRecords | To get list of records from the related list of a module. |
| updateRelatedRecords | To update the association/relation between the records. |
| delinkRecords | To delete the association between the records. |
| getRelatedRecord | To get the records from a specific related list of a module. |
| updateRelatedRecord | To update the details of a specific record of a related list in a module. |
| delink Record | To delete a specific record from the related list of a module |

**Role Operations**

| Method | Description |
| --- | --- |
| getRoles | To get the list of all roles available in your organization. |
| getRole | To get the details of a specific role |

**Shared Records Operations**

| Constructor | Description |
| --- | --- |
| ShareRecordsOperations( recordId, moduleAPIName) | Creates a ShareRecordsOperations class instance with the recordId and moduleAPIName |

| Method | Description |
| --- | --- |
| getSharedRecordDetails | To get the details of a record shared with other users. |
| shareRecord | To share a record with other users in the |

| | |
|---|---|
| | organization. |
| updateSharePermissions | To<br>• Update the sharing permissions of a record granted to users as Read-Write, Read-only, or grant full access.<br>• Revoke access given to users to a shared record.<br>• Update the access permission to the related lists of the record that was shared with the user. |
| revokeSharedRecord | To revoke access to a shared record |

**Tags Operations**

| **Method** | **Description** |
|---|---|
| getTags | To get the list of all tags in your organization. |
| createTags | To create tags. |
| updateTags | To update multiple tags. |
| updateTag | To update a specific tag. |

| | |
|---|---|
| deleteTag | To delete a specific tag from the module. |
| mergeTags | To merge two tags. |
| addTagsToRecord | To add tags to a specific record. |
| removeTagsFromRecord | To remove tags from a record. |
| addTagsToMultipleRecords | To add tags to multiple records. |
| removeTagsFromMultipleRecords | To remove tags from multiple records. |
| getRecordCountForTag | To get the record count for a tag |

**Taxes Operations**

| **Method** | **Description** |
|---|---|
| getTaxes | To get the taxes of your organization. |
| createTaxes | To add taxes to your organization. |
| updateTaxes | To update the existing taxes of your organization. |
| deleteTaxes | To delete multiple taxes from your organization. |
| getTax | To get the details of a specific tax. |

| | |
|---|---|
| deleteTax | To delete a specific tax from your organization |

**Territory Operations**

| Method | Description |
|---|---|
| getTerritories | To get the list of all territories. |
| getTerritory | To get the details of a specific territory |

**Users Operations**

| Method | Description |
|---|---|
| getUsers | To get the list of users in your organization. |
| createUser | To add a user to your organization. |
| updateUsers | To update the existing users of your organization. |
| getUser | To get the details of a specific user. |
| updateUser | To update the details of a specific user. |

| | |
|---|---|
| deleteUser | To delete a specific user from your organization |

**Variable Groups Operations**

| Method | Description |
|---|---|
| getVariableGroups | To get the list of all variable groups available for your organization. |
| getVariableGroupById | To get the details of a variable group by its ID. |
| getVariableGroupByAPIName | To get the details of a specific variable group by its API name |

**Variable Operations**

| Method | Description |
|---|---|
| getVariables | To get the list of variables available for your organization. |
| createVariables | To add new variables to your organization. |
| updateVariables | To update the details of variables. |
| deleteVariables | To delete multiple variables. |
| getVariableById | To get the details of a specific variable by |

| | |
|---|---|
| | its unique ID. |
| updateVariableById | To update the details of a specific variable by its unique ID. |
| deleteVariable | To delete a specific variable. |
| getVariableForAPIName | To get the details of a variable by its API name. |
| updateVariableByAPIName | To update the details of a variable by its API name |

# Release Notes

## Current Version:

### 1. ZCRMSDK -VERSION 1.1.0

- Install command

```
npm install zcrmsdk@1.1.0
```

**Notes**
Supported External ID.

## Previous Versions

### 1. ZCRMSDK -VERSION 1.0.4

- Install command

```
npm install zcrmsdk@1.0.4
```

**Notes**
Supported Carry-over tags.

### 2. ZCRMSDK -VERSION 1.0.3

- Install command

```
npm install zcrmsdk@1.0.3
```

**Notes**
Resolved cyclic import.
Handled null values in the request body.

### 3. ZCRMSDK -VERSION 1.0.2

- Install command

```
npm install zcrmsdk@1.0.2
```

**Notes**
Handled different field types.

### 4. ZCRMSDK -VERSION 1.0.1

- Install command

```
npm install zcrmsdk@1.0.1
```

**Notes**

- Handled file paths across multiple platforms.

**5. ZCRMSDK -VERSION 1.0.0**

- Install command

```
npm install zcrmsdk@1.0.0
```

**Notes**

- This is a new major version of the SDK which was developed with the following objectives:
    - Improve the capabilities of the SDK
    - Incorporate customer feedback
    - Upgrade our dependencies
    - Improve performance
    - Adopt the latest Node standards

**Enhancements**

- The SDK is highly structured to ensure easy access to all the components.
- Each CRM entity is represented by a package, and each package contains an Operations Class that incorporates methods to perform all possible operations over that entity.
- **SDKException** - A wrapper class to wrap all exceptions such as SDK anomalies and other unexpected behaviors.
- **StreamWrapper** - A wrapper class for File operations.
- **APIResponse** - A common response instance for all the SDK method calls.