Typescript SDK Version 1.x.x

ZohoCRM

--zoho.com/crm--

Table of contents

Overview

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

Environmental Setup

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

- The client app must have Node (version 12 and above).
- TypeScript SDK must be installed into the client app through npm.

Including the SDK in your project

You can include the SDK in your project by one of the following ways:

- Installing Node from nodejs.org (if not installed).
- Installing the TypeScript SDK:
    1. Navigate to the workspace of your client app.
    2. Run the following command:

```
npm install @zohocrm/typescript-sdk
```

The TypeScript SDK will be installed and a package named **@zohocrm/typescript-sdk** will be created in the local machine.

Another method to install the SDK:

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

> **Note**
> It is mandatory for the client to have **ZohoCRM.settings.fields.ALL** to access all the recoNoterd operations API. Otherwise, the system returns the **OAUTH-SCOPE-MISMATCH** error

Configuration

Before you get started with creating your TypeScript 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.

```
import {Levels,Logger} from "@zohocrm/typescript-sdk/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.
*/
let logger: Logger = Logger.getInstance(Levels.INFO, "/Users/user_name/Documents/ts_sdk_log.log");
```

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

```
import {UserSignature} from "@zohocrm/typescript-sdk/routes/user_signature"
//Create an UserSignature instance that takes user Email as parameter
let user: UserSignature = new UserSignature("abc@zoho.com");
```

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

```
import {USDataCenter} from "@zohocrm/typescript-sdk/routes/dc/us_data_center"
/*
* Configure the environment
* which is of the pattern Domain.Environment
 * Available Domains: USDataCenter, EUDataCenter, INDataCenter, CNDataCenter, AUDataCenter
 * Available Environments: PRODUCTION(), DEVELOPER(), SANDBOX()
 */
```

```
let environment: Environment = USDataCenter.PRODUCTION();
```

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

```
import { OAuthToken,TokenType } from "@zohocrm/typescript-sdk/models/authenticator/oauth_token"
    /*
     * Create a Token instance
     * 1 -> OAuth client id.
     * 2 -> OAuth client secret.
     * 3 -> REFRESH/GRANT token.
     * 4 -> token type.
     * 5 -> OAuth redirect URL.
    */
    let token: OAuthToken = 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.

```
import {DBStore} from "@zohocrm/typescript-sdk/models/authenticator/store/db_store"
import {FileStore} from "@zohocrm/typescript-sdk/models/authenticator/store/file_store"
/*
* DBStore takes the following parameters
* 1 -> DataBase host name. Default value "localhost"
 * 2 -> DataBase name. Default  value "zohooauth"
* 3 -> DataBase user name. Default value "root"
* 4 -> DataBase password. Default value ""
* 5 -> DataBase port number. Default value "3306"
*/

//let tokenstore: DBStore = new DBStore();

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

```
FileStore("/Users/userName/Documents/tssdk-tokens.txt")
```

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

```
import {SDKConfig} from "@zohocrm/typescript-sdk/routes/sdk_config";
import {SDKConfigBuilder} from "@zohocrm/typescript-sdk/routes/sdk_config_builder";
/*
     * 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.ts)
     *
     * 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: SDKConfig = new SDKConfigBuilder().setPickListValidation(false).setAutoRefreshFields(true).build();
```

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

```
let resourcePath: string = "/Users/user_name/Documents/typescript-app";
```

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

```
import { RequestProxy} from "@zohocrm/typescript-sdk/routes/request_proxy"
```

```
2
3       /*
4        * RequestProxy class takes the following parameters
5        * 1 -> Host
6        * 2 -> Port Number
7        * 3 -> User Name. Default null.
8        * 4 -> Password. Default null
9        */
10      let requestProxy: 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: UserSignature, token: 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: UserSignature, token: Token)** - invoked after fetching access and refresh tokens from Zoho.
- **deleteToken(token: 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;
3 Here is the code to create a DBStore object:
```

```
1 import {DBStore} from "@zohocrm/typescript-
  sdk/models/authenticator/store/db_store";
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 ""
8 * 5 -> DataBase port number. Default value "3306"
9 */
10
11 let tokenstore: DBStore = new DBStore();
12
13 let tokenstore: DBStore = new DBStore("hostName",
   "dataBaseName", "userName", "password", "portNumber");
```

File Persistence

In case of file persistence, you can set up persistence of 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:

```
1 import {FileStore} from "@zohocrm/typescript-
   sdk/models/authenticator/store/file_store";
2 /*
3 * FileStore takes the following parameter
4 * 1 -> Absolute file path of the file to persist tokens
5 */
6 let store: FileStore = new
   FileStore("/Users/username/Documents/ts_sdk_tokens.txt");
```

Custom Persistence

To use Custom Persistence, you must extend the TokenStore class

(@zohocrm/typescript-sdk/models/authenticator/store/token_store) and override the methods.

Here is the code:

```
import { TokenStore } from "@zohocrm/typescript-sdk/models/authenticator/store/token_store";

export class CustomStore implements TokenStore {

    constructor(){
    }

    /**
        *
        * @param {UserSignature} user A UserSignature class instance.
        * @param {Token} token A Token (@zohocrm/typescript-sdk/models/authenticator/oauth_token) class instance.
        * @returns A Token class instance representing the user token details.
        * @throws {SDKException} if any error occurs.
        */
   async getToken(user: UserSignature, token: Token): Promise<Token | undefined> {
        // Add code to get the token
        return undefined;
    }

    /**
        *
        * @param {UserSignature} user A UserSignature class instance.
        * @param {Token} token A Token (@zohocrm/typescript-sdk/models/authenticator/oauth_token) class instance.
        * @throws {SDKException} if any error occurs.
        */
    async saveToken(user: UserSignature, token: Token): Promise<void>{
        // Add code to save the token
```

```
28      }
29
30      /**
31          *
32          * @param {Token} token A Token (@zohocrm/typescript-
    sdk/models/authenticator/oauth_token) class instance.
33          * @throws {SDKException} if any error occurs.
34          */
35      async deleteToken(token: Token): Promise<void> {
36          // Add code to delete the token
37      }
38
39      /**
40      * @returns {Array} - An array of Token class instances
41      * @throws {SDKException}
42      */
43      async getTokens(): Promise<Token[]> {
44          //Add code to retrieve all the stored tokens.
45      }
46
47      /**
48      * @throws {SDKException}
49      */
50      deleteTokens(): void {
51          //Add code to delete all the stored tokens.
52      }
53  }
```

Register your Application

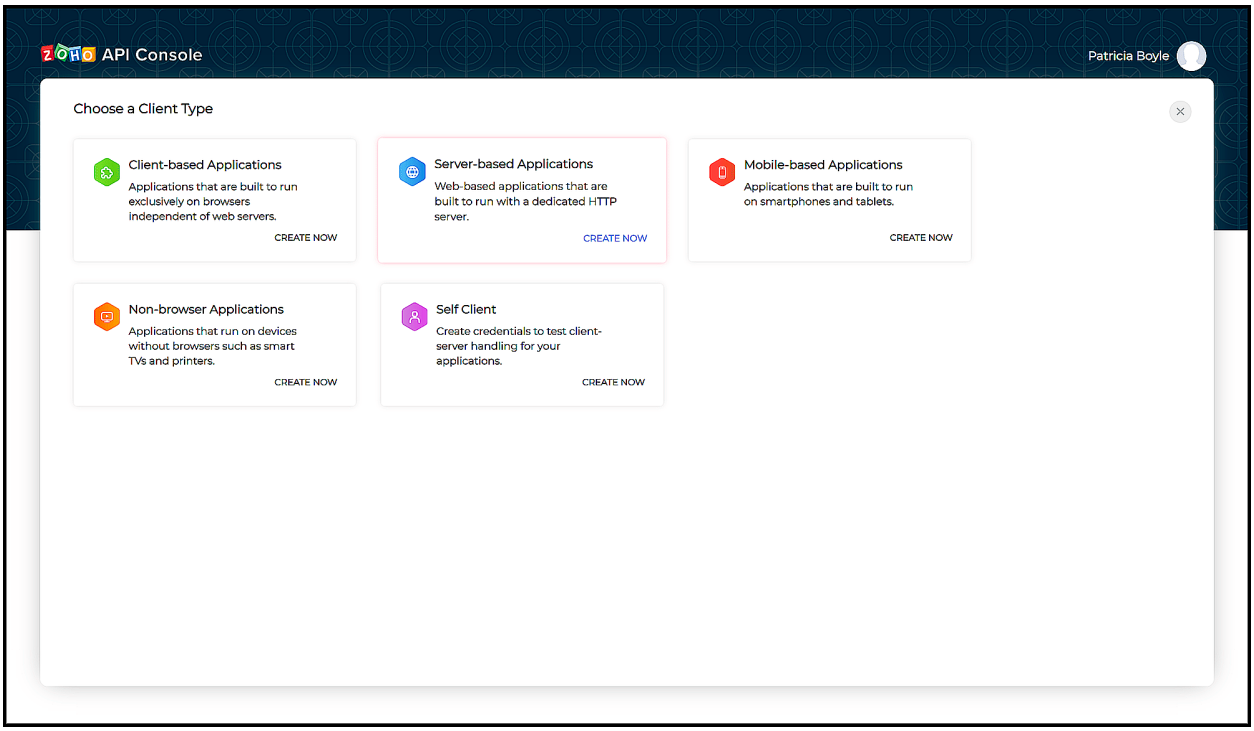
Before you get started with authorization and make any calls using the Zoho CRM APIs, you need to register your application with Zoho CRM.

To register,

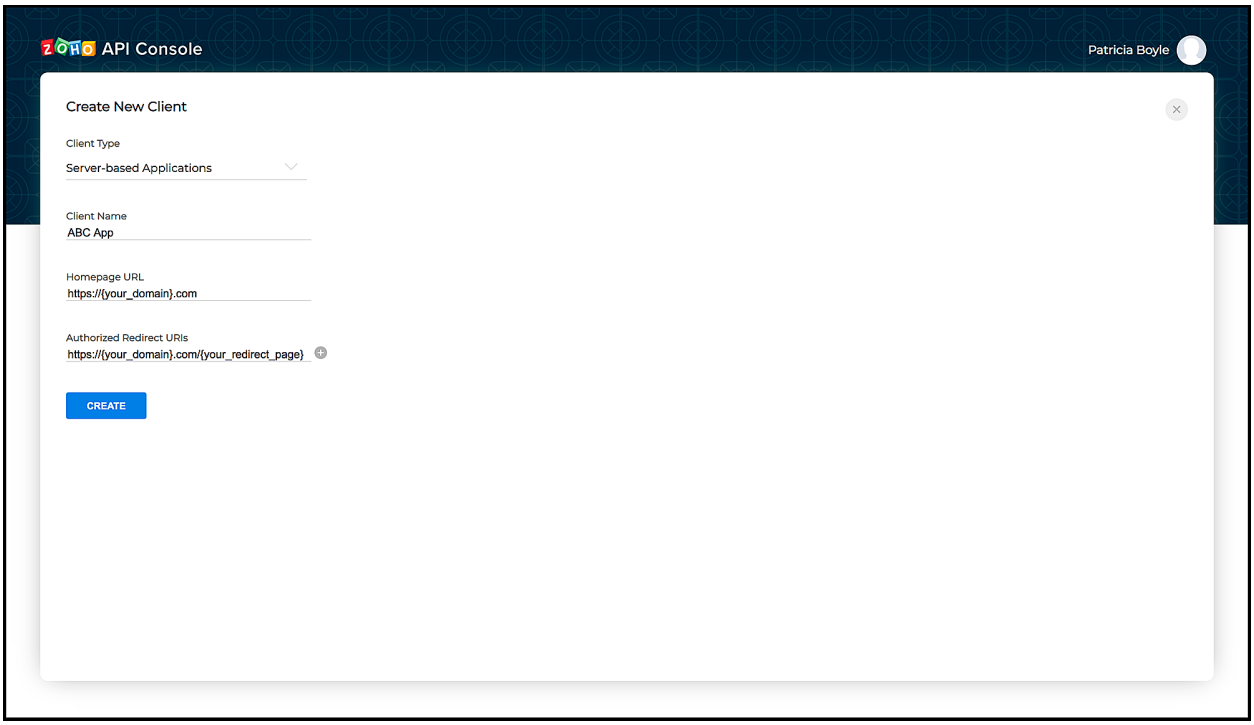
- Go to Zoho Developer Console.
- Choose a client type:

- **Client-based**: Applications that are built to run exclusively on browsers independent of web servers.
- **Server-based**: Web-based applications that are built to run with 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.

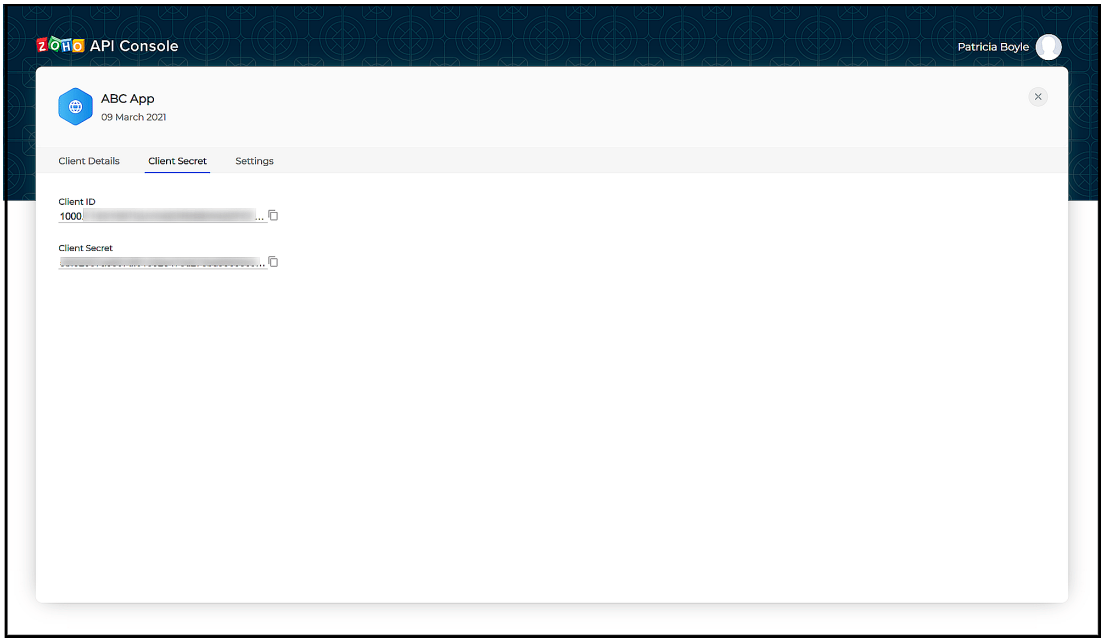
For more details, refer to OAuth Overview.



- 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.



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.

NoteNote

Initializing the Application

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

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. Provide “aaaserver.profile.READ” scope along with Zoho CRM scopes.
5. Copy the grant token that is displayed on the screen.

Note

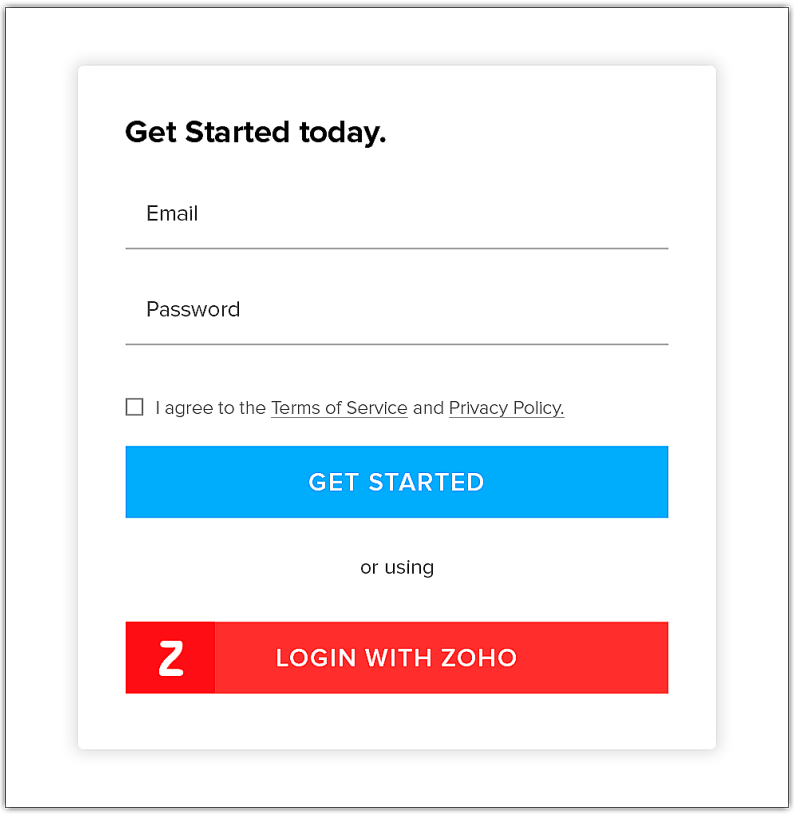
- The generated grant token is valid only for the stipulated time you chose while generating it. Hence, the access and refresh tokens should be generated within that time.
- The OAuth client registration and grant token generation must be done in the same Zoho account's (meaning - login) developer console.

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

```
import {UserSignature} from "@zohocrm/typescript-sdk/routes/user_signature"
import {SDKConfigBuilder} from "@zohocrm/typescript-sdk/routes/sdk_config_builder"
import {DBStore} from "@zohocrm/typescript-sdk/models/authenticator/store/db_store"
import {FileStore} from "@zohocrm/typescript-sdk/models/authenticator/store/file_store"
import {SDKConfig} from "@zohocrm/typescript-sdk/routes/sdk_config"
import {Levels,Logger} from "@zohocrm/typescript-sdk/routes/logger/logger"
import {Environment} from "@zohocrm/typescript-sdk/routes/dc/environment"
import {USDataCenter} from "@zohocrm/typescript-sdk/routes/dc/us_data_center"
import {OAuthToken,TokenType} from "@zohocrm/typescript-sdk/models/authenticator/oauth_token"
import {Initializer} from "@zohocrm/typescript-sdk/routes/initializer"
import {RequestProxy} from "@zohocrm/typescript-sdk/routes/request_proxy"

export class Initializer{

    public 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.
   */
        let logger: Logger = Logger.getInstance(Levels.INFO,
```

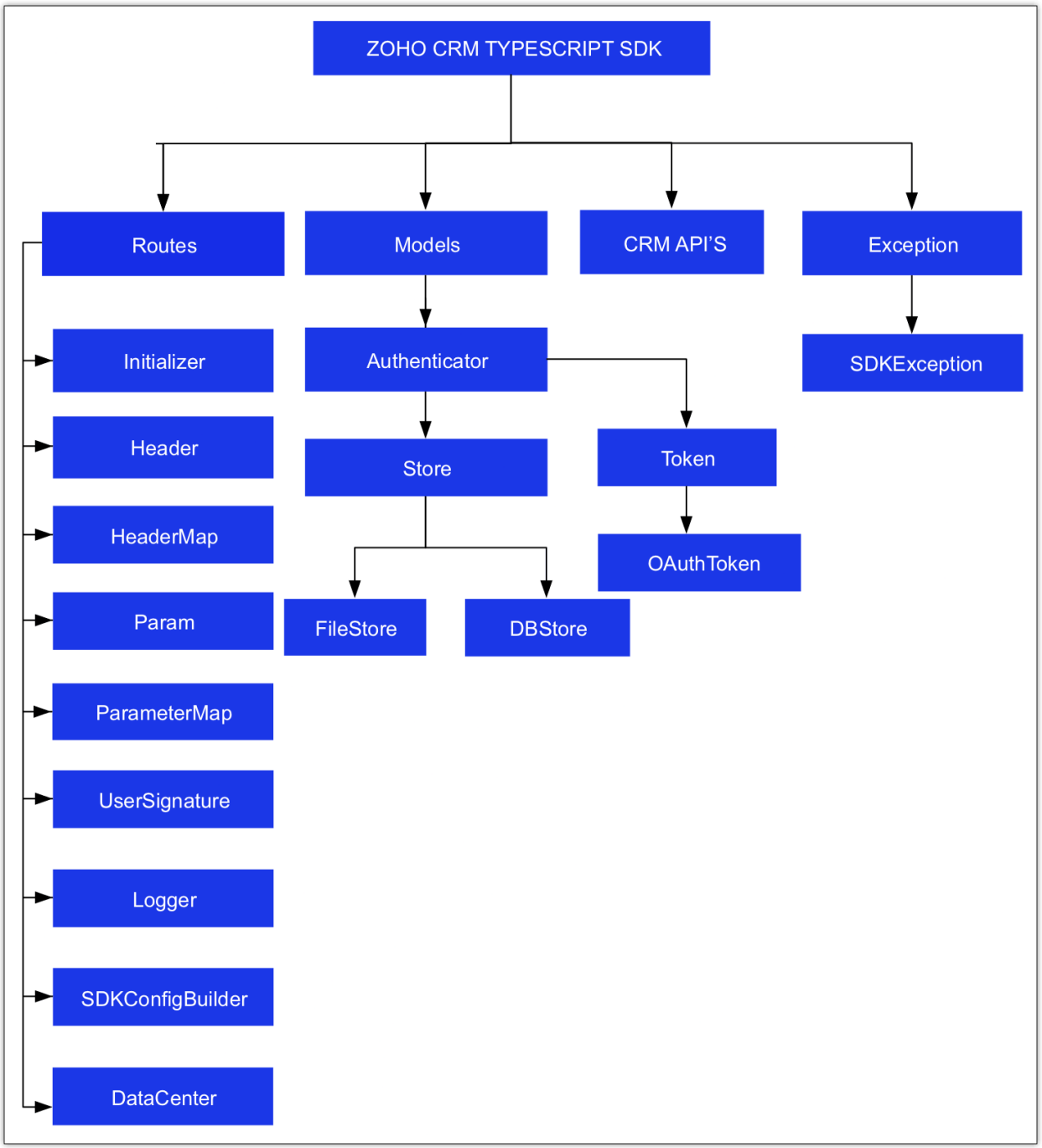
```
"/Users/user_name/Documents/ts_sdk_log.log");

        /*
         * Create an UserSignature instance that takes user Email as parameter
         */
        let user: UserSignature = 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: Environment = USDataCenter.PRODUCTION();

        /*
   * Create a Token instance
   * 1 -> OAuth client id.
   * 2 -> OAuth client secret.
   * 3 -> REFRESH/GRANT token.
   * 4 -> token type.
         * 5 -> OAuth redirect URL. Default value is null
   */
        let token: OAuthToken = new OAuthToken("clientId", "clientSecret", "REFRESH/ GRANT Token", TokenType.REFRESH/TokenType.GRANT, "redirectURL");

        /*
   * Create an instance of TokenStore.
   * 1 -> DataBase host name. Default "localhost"
   * 2 -> DataBase name. Default "zohooauth"
   * 3 -> DataBase user name. Default "root"
   * 4 -> DataBase password. Default ""
   * 5 -> DataBase port number. Default "3306"
   */
        // let tokenstore = new DBStore();

        let tokenstore: DBStore = new DBStore("hostName",
```

```
"dataBaseName", "userName", "password", "portNumber");

        /*
   * Create an instance of FileStore that takes absolute file path as parameter
   */
        // let tokenstore: FileStore = new FileStore("/Users/username/Documents/ts_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.ts)
  *
  * pickListValidation
  * if true - value for any picklist field will be validated with the available values.
  * if false - value for any picklist field will not be validated, resulting in creation of a new value.
 */
 let sdkConfig: 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: string = "/Users/user_name/Documents/tsssdk-application";

        /*
   * Create an instance of RequestProxy class that takes the following parameters
   * 1 -> Host
```

```
    * 2 -> Port Number
    * 3 -> User Name
    * 4 -> Password
    */
        let proxy: RequestProxy = new RequestProxy("proxyHost", 80);

        let proxy: RequestProxy = new RequestProxy("proxyHost", 80, "proxyUser", "password");

        /*
         * Call the static initialize method of Initializer class that takes the following arguments
         * 1 -> UserSignature instance
         * 2 -> Environment instance
         * 3 -> Token instance
         * 4 -> TokenStore instance
         * 5 -> SDKConfig instance
         * 6 -> resourcePath
         * 7 -> Logger instance. Default value is null
         * 8 -> RequestProxy instance. Default value is null
         */
        // The SDK can be initialized by any of the following methods
        await Initializer.initialize(user, environment, token, store, sdkConfig, resourcePath)

        await Initializer.initialize(user, environment, token, store, sdkConfig, resourcePath, logger, proxy);
    }
}
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 TypeScript SDK is depicted in the following image.



Multi-User Support

The TypeScript SDK 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

```
import {UserSignature} from "@zohocrm/typescript-sdk/routes/user_signature"
import {SDKConfigBuilder} from "@zohocrm/typescript-sdk/routes/sdk_config_builder"
import {DBStore} from "@zohocrm/typescript-sdk/models/authenticator/store/db_store"
import {FileStore} from "@zohocrm/typescript-sdk/models/authenticator/store/file_store"
import {SDKConfig} from "@zohocrm/typescript-sdk/routes/sdk_config"
import {Levels,Logger} from "@zohocrm/typescript-sdk/routes/logger/logger"
import {Environment} from "@zohocrm/typescript-sdk/routes/dc/environment"
import {USDataCenter} from "@zohocrm/typescript-sdk/routes/dc/us_data_center"
import {EUDataCenter} from "@zohocrm/typescript-
```

```
sdk/routes/dc/eu_data_center"
import { OAuthToken,TokenType } from "@zohocrm/typescript-sdk/models/authenticator/oauth_token"
import { Initializer} from "@zohocrm/typescript-sdk/routes/initializer"
import { RequestProxy} from "@zohocrm/typescript-sdk/routes/request_proxy"

import {RecordOperations, GetRecordsHeader, GetRecordsParam} from "@zohocrm/typescript-sdk/core/com/zoho/crm/api/record/record_operations";
import {ResponseWrapper} from  "@zohocrm/typescript-sdk/core/com/zoho/crm/api/record/response_wrapper";
import {ResponseHandler} from  "@zohocrm/typescript-sdk/core/com/zoho/crm/api/record/response_handler";
import {Record} from "@zohocrm/typescript-sdk/core/com/zoho/crm/api/record/record";
import {Tag} from "@zohocrm/typescript-sdk/core/com/zoho/crm/api/tags/tag";

import {APIResponse} from "@zohocrm/typescript-sdk/routes/controllers/api_response";
import { SDKException } from "@zohocrm/typescript-sdk/core/com/zoho/crm/api/exception/sdk_exception";
import {ParameterMap} from "@zohocrm/typescript-sdk/routes/parameter_map";
import {HeaderMap} from "@zohocrm/typescript-sdk/routes/header_map";

class SampleRecord{

    public 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.
```

```
33          */
34         let logger = Logger.getInstance(Levels.INFO,
   "/Users/user_name/Documents/ts_sdk_log.log");
35 
36         /*
37          * Create an UserSignature instance that takes user Email
   as parameter
38          */
39         let user1 = new UserSignature("abc@zoho.com");
40 
41         /*
42          * Configure the environment
43          * which is of the pattern Domain.Environment
44          * Available Domains: USDataCenter, EUDataCenter,
   INDataCenter, CNDataCenter, AUDataCenter
45          * Available Environments: PRODUCTION(), DEVELOPER(),
   SANDBOX()
46          */
47         let environment1: Environment =
   USDataCenter.PRODUCTION();
48 
49         /*
50          * Create a Token instance
51          * 1 -> OAuth client id.
52          * 2 -> OAuth client secret.
53          * 3 -> REFRESH/GRANT token.
54          * 4 -> token type.
55          * 5 -> OAuth redirect URL. Default value is null
56          */
57         let token1 = new OAuthToken("clientId1", "clientSecret1",
   "REFRESH/ GRANT Token", TokenType.REFRESH/TokenType.GRANT,
   "redirectURL");
58 
59         /*
60          * Create an instance of TokenStore.
61          * 1 -> DataBase host name. Default "localhost"
62          * 2 -> DataBase name. Default "zohooauth"
63          * 3 -> DataBase user name. Default "root"
64          * 4 -> DataBase password. Default ""
65          * 5 -> DataBase port number. Default "3306"
```

```
66          */
67         let store: DBStore = new DBStore();
68 
69         let store: DBStore = new DBStore("hostName",
   "dataBaseName", "userName", "password", "portNumber");
70 
71         /*
72          * Create an instance of FileStore that takes absolute
   file path as parameter
73          */
74         let store: FileStore = new
   FileStore("/Users/username/Documents/ts_sdk_tokens.txt");
75 
76         /*
77         * autoRefreshFields
78         * if true - all the modules' fields will be auto-
   refreshed in the background, every    hour.
79         * 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.ts)
80         *
81         * pickListValidation
82         * 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.
83         * True - the SDK validates the input. If the value does
   not exist in the pick list, the SDK throws an error.
84         * False - the SDK does not validate the input and makes
   the API request with the user's input to the pick list
85         */
86         let sdkConfig: SDKConfig = new
   SDKConfigBuilder().setPickListValidation(false).setAutoRefreshFie
   lds(true).build();
87 
88         /*
89          * The path containing the absolute directory path to
   store user specific JSON files containing module fields
   information.
90          */
```

```
91          let resourcePath: string =
    "/Users/user_name/Documents/ts-app";
92
93          /*
94           * Call the static initialize method of Initializer class
    that takes the following arguments
95           * 1 -> UserSignature instance
96           * 2 -> Environment instance
97           * 3 -> Token instance
98           * 4 -> TokenStore instance
99           * 5 -> SDKConfig instance
100              * 6 -> resourcePath
101              * 7 -> Logger instance. Default value is null
102              * 8 -> RequestProxy instance. Default value is null
103              */
104             await Initializer.initialize(user1, environment1,
    token1, store, sdkConfig, resourcePath, logger);
105
106             await SampleRecord.getRecords("Leads");
107
108             await Initializer.removeUserConfiguration(user1,
    environment1);
109
110             let user2: UserSignature = new
    UserSignature("abc2@zoho.eu");
111
112             let environment2: Environment =
    EUDataCenter.SANDBOX();
113
114             let token2: OAuthToken = new OAuthToken("clientId2",
    "clientSecret2", "REFRESH/ GRANT Token", TokenType.REFRESH,
    "redirectURL");
115
116             let requestProxy: RequestProxy = new
    RequestProxy("proxyHost", 80, "proxyUser", "password");
117
118             let sdkConfig2: SDKConfig = new
    SDKConfigBuilder().setPickListValidation(true).setAutoRefreshFiel
    ds(true).build();
119
```

```
            await Initializer.switchUser(user2, environment2, token2, sdkConfig2, requestProxy);

            await SampleRecord.getRecords("Leads");
        }

        static async getRecords(moduleAPIName: string){
            try {
                let moduleAPIName = "Leads";
                //Get instance of RecordOperations Class
                let recordOperations: RecordOperations = new RecordOperations();
                let paramInstance: ParameterMap = new ParameterMap();
                await paramInstance.add(GetRecordsParam.APPROVED, "both");
                let headerInstance: HeaderMap  = new HeaderMap();
                await headerInstance.add(GetRecordsHeader.IF_MODIFIED_SINCE, new Date("2020-01-01T00:00:00+05:30"));
                //Call getRecords method that takes paramInstance, headerInstance and moduleAPIName as parameters
                let response: APIResponse<ResponseHandler> = await recordOperations.getRecords(moduleAPIName, paramInstance, headerInstance);
                if(response != null){

                    //Get the status code from response
                    console.log("Status Code: " + response.getStatusCode());
                    if([204, 304].includes(response.getStatusCode())){
                        console.log(response.getStatusCode() == 204? "No Content" : "Not Modified");
                        return;
                    }
                    //Get the object from response
                    let responseObject: ResponseHandler = response.getObject();
```

```
146                     if(responseObject != null){
147                         //Check if expected ResponseWrapper
   instance is received
148                         if(responseObject instanceof
   ResponseWrapper){
149                             //Get the array of obtained Record
   instances
150                             let records: Record[] =
   responseObject.getData();
151                             for (let record of records) {
152 
153                                 //Get the ID of each Record
154                                 console.log("Record ID: " +
   record.getId());
155                                 //Get the createdBy User
   instance of each Record
156                                 let createdBy =
   record.getCreatedBy();
157                                 //Check if createdBy is not null
158                                 if(createdBy != null)
159                                 {
160                                     //Get the ID of the
   createdBy User
161                                     console.log("Record Created
   By User-ID: " + createdBy.getId());
162                                     //Get the name of the
   createdBy User
163                                     console.log("Record Created
   By User-Name: " + createdBy.getName());
164                                     //Get the Email of the
   createdBy User
165                                     console.log("Record Created
   By User-Email: " + createdBy.getEmail());
166                                 }
167                                 //Get the CreatedTime of each
   Record
168                                 console.log("Record CreatedTime:
   " + record.getCreatedTime());
169                                 //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: Tag[] =
   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());
                                        });
                                    }
                                    //To get particular field value
                                    console.log("Record Field Value:
```

```
    " + record.getKeyValue("Last_Name"));// FieldApiName
195
196                                 console.log("Record KeyValues: "
    );
197                                 let keyValues: Map<string,any> =
    record.getKeyValues();
198                                 let keyArray: string[] =
    Array.from(keyValues.keys());
199                                 for (let keyName of keyArray) {
200                                     let value: any =
    keyValues.get(keyName);
201                                     console.log(keyName + " : "
    + value);
202                                 }
203                             }
204                         }
205                     }
206                 }
207             } catch (error) {
208                 console.log(error);
209             }
210         }
211     }
212     SampleRecord.call();
213
```

1. The program execution starts from **call()**
2. The details of **user1** are given in the variables **user1, token1, environment1**.
3. Similarly, the details of another user **user2** are 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 records.

SDK Sample Code

```
1  import {UserSignature} from "@zohocrm/typescript-
```

```
   sdk/routes/user_signature"
2  import {SDKConfigBuilder} from "@zohocrm/typescript-
   sdk/routes/sdk_config_builder"
3  import {DBStore} from "@zohocrm/typescript-
   sdk/models/authenticator/store/db_store"
4  import {FileStore} from "@zohocrm/typescript-
   sdk/models/authenticator/store/file_store"
5  import {SDKConfig} from "@zohocrm/typescript-
   sdk/routes/sdk_config"
6  import {Levels,Logger} from "@zohocrm/typescript-
   sdk/routes/logger/logger"
7  import {Environment} from "@zohocrm/typescript-
   sdk/routes/dc/environment"
8  import {USDataCenter} from "@zohocrm/typescript-
   sdk/routes/dc/us_data_center"
9  import { OAuthToken,TokenType } from "@zohocrm/typescript-
   sdk/models/authenticator/oauth_token"
10 import { Initializer} from "@zohocrm/typescript-
   sdk/routes/initializer"
11
12 import {RecordOperations, GetRecordsHeader, GetRecordsParam} from
   "@zohocrm/typescript-
   sdk/core/com/zoho/crm/api/record/record_operations";
13 import {ParameterMap} from "@zohocrm/typescript-
   sdk/routes/parameter_map";
14 import {HeaderMap} from "@zohocrm/typescript-
   sdk/routes/header_map";
15 import {ResponseWrapper} from  "@zohocrm/typescript-
   sdk/core/com/zoho/crm/api/record/response_wrapper";
16 import {ResponseHandler} from  "@zohocrm/typescript-
   sdk/core/com/zoho/crm/api/record/response_handler";
17 import {Record} from "@zohocrm/typescript-
   sdk/core/com/zoho/crm/api/record/record";
18 import {Tag} from "@zohocrm/typescript-
   sdk/core/com/zoho/crm/api/tags/tag";
19
20 import {APIResponse} from "@zohocrm/typescript-
   sdk/routes/controllers/api_response";
21 class SampleRecord {
22
```

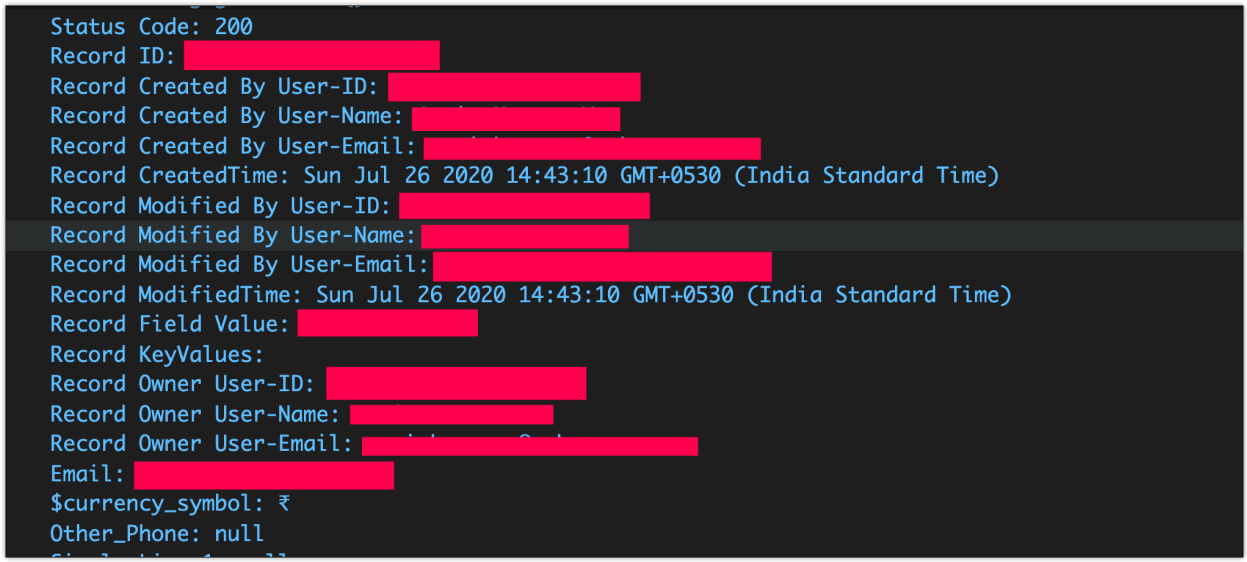
```
23      public static async getRecords(){
24
25          let user: UserSignature = new
    UserSignature("abc@zoho.com");
26          let myLogger: Logger = Logger.getInstance(Levels.INFO,
    "/Users/user_name/Documents/ts_sdk_log.log");
27          let dc: Environment = USDataCenter.PRODUCTION();
28          let sdkConfig: SDKConfig = new
    SDKConfigBuilder().setAutoRefreshFields(false).setPickListValidat
    ion(true).build();
29          // let store: DBStore = new DBStore(undefined, undefined,
    undefined, "abc");
30          let store: FileStore = new
    FileStore("/Users/username/Documents/ts_sdk_tokens.txt");
31          let oauth: OAuthToken = new OAuthToken("clientId",
    "clientSecret", "REFRESH/ GRANT Token",
    TokenType.REFRESH/TokenType.GRANT);
32          let path: string = "/Users/user_name/Documents/ts-app";
33          await Initializer.initialize(user, dc, oauth, store,
    sdkConfig, path, myLogger);
34
35          try {
36              let moduleAPIName = "Leads";
37              //Get instance of RecordOperations Class
38              let recordOperations: RecordOperations = new
    RecordOperations();
39              let paramInstance: ParameterMap = new ParameterMap();
40              await paramInstance.add(GetRecordsParam.APPROVED,
    "both");
41              let headerInstance: HeaderMap  = new HeaderMap();
42              await
    headerInstance.add(GetRecordsHeader.IF_MODIFIED_SINCE, new
    Date("2020-01-01T00:00:00+05:30"));
43              //Call getRecords method that takes paramInstance,
    headerInstance and moduleAPIName as parameters
44              let response: APIResponse<ResponseHandler> = await
    recordOperations.getRecords(moduleAPIName, paramInstance,
    headerInstance);
45              if(response != null){
46
```

```
47                 //Get the status code from response
48                 console.log("Status Code: " +
   response.getStatusCode());
49                 if([204,
   304].includes(response.getStatusCode())){
50                     console.log(response.getStatusCode() == 204?
   "No Content" : "Not Modified");
51                     return;
52                 }
53                 //Get the object from response
54                 let responseObject: ResponseHandler =
   response.getObject();
55                 if(responseObject != null){
56                     //Check if expected ResponseWrapper instance
   is received
57                     if(responseObject instanceof
   ResponseWrapper){
58                         //Get the array of obtained Record
   instances
59                         let records: Record[] =
   responseObject.getData();
60                         for (let record of records) {
61
62                             //Get the ID of each Record
63                             console.log("Record ID: " +
   record.getId());
64                             //Get the createdBy User instance of
   each Record
65                             let createdBy =
   record.getCreatedBy();
66                             //Check if createdBy is not null
67                             if(createdBy != null)
68                             {
69                                 //Get the ID of the createdBy
   User
70                                 console.log("Record Created By
   User-ID: " + createdBy.getId());
71                                 //Get the name of the createdBy
   User
72                                 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: Tag[] = 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());
                                     });
                                }
                                //To get particular field value
                                console.log("Record Field Value:
    " + record.getKeyValue("Last_Name"));// FieldApiName

                                console.log("Record KeyValues: "
    );
                                let keyValues: Map<string,any> =
    record.getKeyValues();
                                let keyArray: string[] =
    Array.from(keyValues.keys());
                                for (let keyName of keyArray) {
                                    let value: any =
    keyValues.get(keyName);
                                    console.log(keyName + " : "
    + value);
                                }
                            }
                        }
                    }
                } catch (error) {
                    console.log(error);
                }
            }
        }

    SampleRecord.getRecords();
```

Record Response



Sample Codes

All of Zoho CRM's APIs can be used through the TypeScript 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: string, recordID: bigint)	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.
uploadLinkAttachment	To upload a link as an attachment to a record

Blueprint Operations

Constructor	**Description**
BluePrintOperations(recordId: bigint, moduleAPIName: string)	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.
downloadResult	To download the result of the bulk write 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: string)	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 Meta Data Operations

Constructor	**Description**
FieldsOperations(module: string)	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: string)	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

Profiles Operations

Constructor	Description
ProfilesOperations(ifModifiedSince: Date)	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 Operations(COQL)

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: string)	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: string, recordId: bigint, moduleAPIName: string)	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: bigint, moduleAPIName: string)	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 ● Update the sharing permissions of a record granted to users as Read-Write, Read-only, or grant full access. ● Revoke access given to users to a shared record. ● 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

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 the **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**
    - **FileBodyWrapper**
    - **MassUpdateResponseWrapper**
    - **APIException**
- For File download responses
    - **FileBodyWrapper**
    - **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.

Release Notes

Current Version

1. **@zohocrm/typescript-sdk - VERSION 1.1.1**

Install command

```
npm install @zohocrm/typescript-sdk@1.1.1
```

Issue Fix

- Improved to fix an existing bug that caused "Cannot read property 'hasOwnProperty'" error due to improper file processing in Windows OS.

Previous Versions

2. **@zohocrm/typescript-sdk - VERSION 1.1.0**

Install command

```
npm install zcrmsdk@1.1.0
```

Notes

Supported External ID.

3. **@zohocrm/typescript-sdk - VERSION 1.0.2**

Install command

```
npm install zcrmsdk@1.0.2
```

Notes

Handled Date objects.

4. **@zohocrm/typescript-sdk - VERSION 1.0.1**

Install command

```
npm install @zohocrm/typescript-sdk@1.0.1
```

Notes

- 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.