JAVA SDK

Version 3

Zoho CRM

--zoho.com/crm--

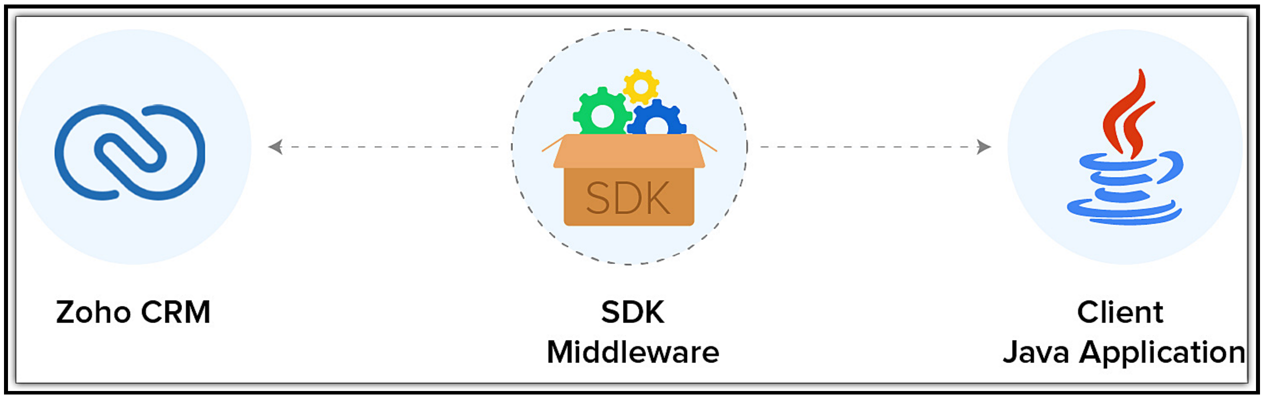
Table of Contents

Overview

Java SDK offers a way to create client java applications that can be integrated with Zoho CRM. This SDK makes the access and use of necessary CRM APIs easy. In other words, it serves as a wrapper for the REST APIs, making it easier to use the services of Zoho CRM.

A point to note would be that the developer of the client application should create programming code elements along with configuration-related properties files, interface implementations, instances or objects. Authentication to access Zoho CRM APIs is through OAuth authentication mechanism. Invariably, HTTP requests and responses are taken care of by the SDK.

A sample of how an SDK acts a middleware or interface between Zoho CRM and a client Java application.



Java SDK allows you to

1. Exchange data between Zoho CRM and the client application where the CRM entities are modeled as classes.
2. Declare and define CRM API equivalents as simple functions in your Java application.
3. Push data into Zoho CRM by accessing appropriate APIs of the CRM Service.

Note:
For the sake of better explanation, we have used Eclipse to describe how to get started on using the SDK.

Environmental Setup

Java SDK requires Java (version 7 and above) to be setup in your development environment.

Including the SDK in your project

Java SDK is available through Maven distribution. You can include the SDK to your project using:

- Maven
- Gradle
- Downloadable JARs (by Zoho)

Maven Distribution

Maven is a build automation tool used primarily for Java projects. Maven addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies.

If you are using Maven to build your project, we already have the dependencies set up.

You just need to include the following in your **pom.xml** file, which will get created once your Java project is created using Maven.

```xml
<repositories>
<repository>
<id>java-sdk</id>
<url>https://maven.zohodl.com</url>
</repository>
</repositories>

<dependencies>
<dependency>
<groupId>com.zoho.crm</groupId>
<artifactId>java-sdk</artifactId>
<version>3.1.0</version>
</dependency>
```

```
</dependencies>
```

Gradle

```
repositories{
    maven { url "https://maven.zohodl.com" }
}
dependencies{
    implementation 'com.zoho.crm:java-sdk:3.1.0'
}
```

Downloadable JAR File
Download SDK

This version downloads simply the SDK without dependent jars. In this case, the following jars are to be made available by adding them in the referenced libraries of your java application. The jars can be downloaded from here.

The list of dependency JARs that you need are:

- commons-lang3-3.9.jar
- httpclient-4.4.1
- httpcore-4.4.4
- httpmime-4.5.3
- json-20170516
- commons-logging-1.1.3
- mysql-connector-java-5.1.47-bin.jar
- opencsv-5.0.jar

Note:

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

- 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.

Using Java SDK through Maven

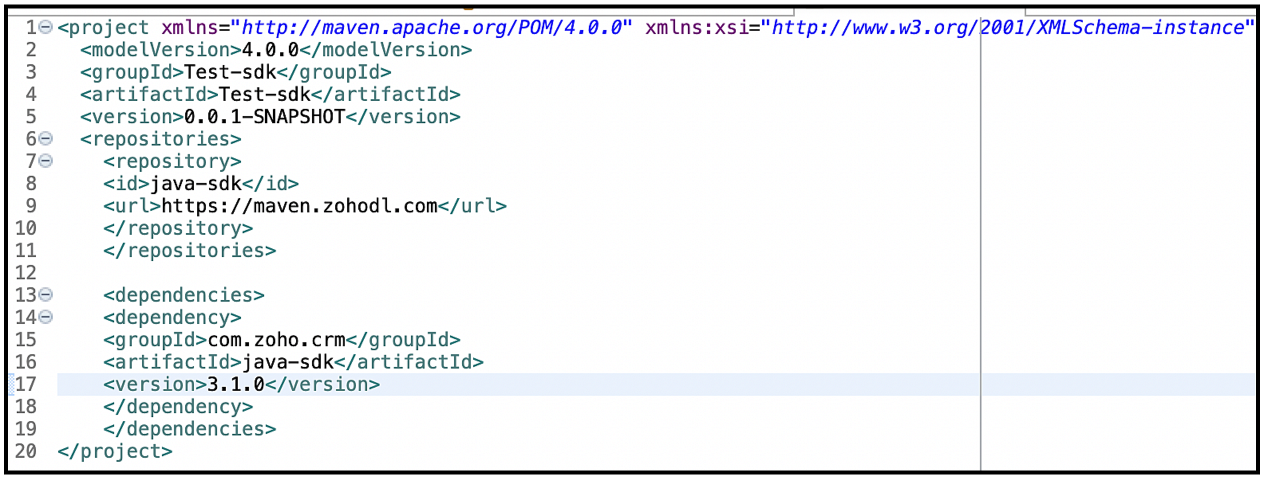
Follow the below steps to use the Java ZCRSMDK through Maven.
1. Create a Maven project.

2. Place the below code in your **pom.xml** file of your Maven project.

```
<repositories>
<repository>
<id>java-sdk</id>
<url>https://maven.zohodl.com</url>
</repository>
</repositories>

<dependencies>
<dependency>
<groupId>com.zoho.crm</groupId>
<artifactId>java-sdk</artifactId>
```

```
<version>3.1.0</version>
</dependency>
</dependencies>
```

The below image shows how your pom.xml should look like.



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <modelVersion>4.0.0</modelVersion>
  <groupId>Test-sdk</groupId>
  <artifactId>Test-sdk</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <repositories>
    <repository>
    <id>java-sdk</id>
    <url>https://maven.zohodl.com</url>
    </repository>
    </repositories>

    <dependencies>
    <dependency>
    <groupId>com.zoho.crm</groupId>
    <artifactId>java-sdk</artifactId>
    <version>3.1.0</version>
    </dependency>
    </dependencies>
</project>
```

3. Update the maven project. Under project explorer, right click the project name, select **Maven > update** project. The jar will be downloaded in the maven dependencies.

4. Inside your source code, import the appropriate files from the SDK.

5. Generate the grant token and initialize the SDK.

6. You can now access the functionalities of the SDK. Refer to sample codes to make various API calls through the SDK.

Register your application

All the Zoho CRM APIs are authenticated with OAuth2 standards, so it is mandatory to register and authenticate your client app with Zoho.

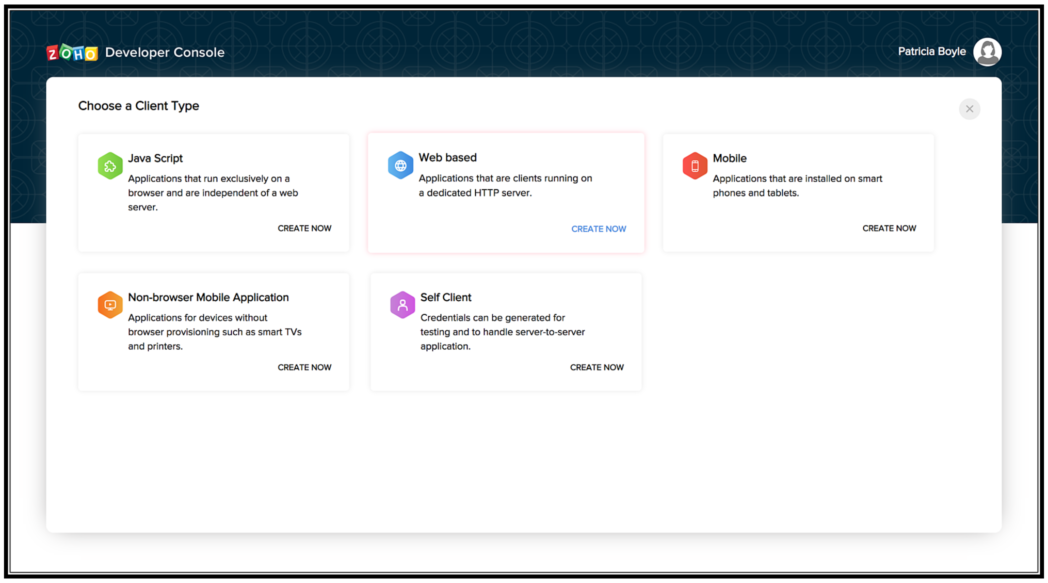
To register,

1. Go to Zoho Developer Console.

2. Choose a client type:

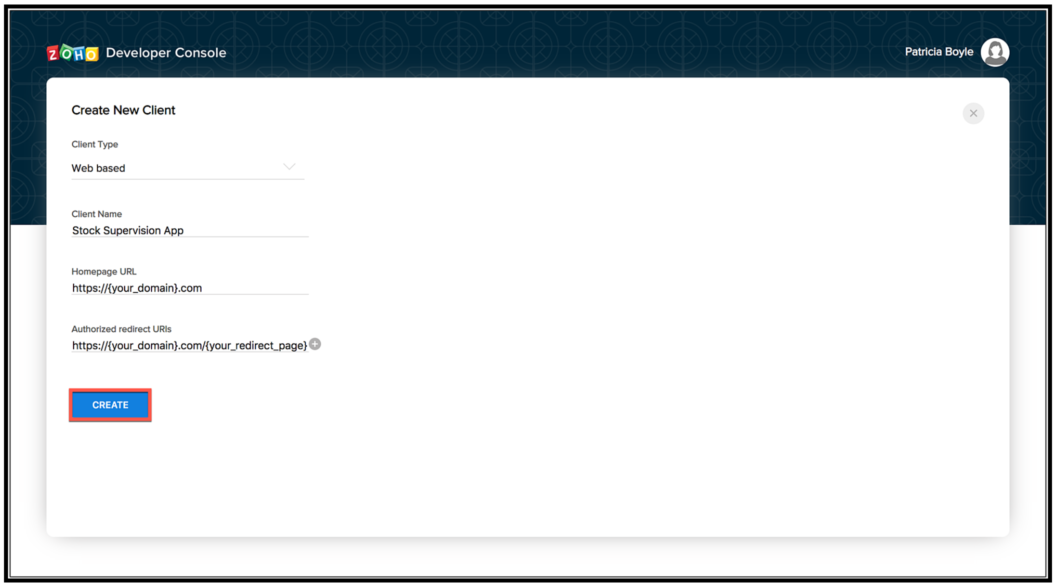
- **Java Script**: Applications that run exclusively on a browser and are independent of a web server.
- **Web Based**: Applications that are clients running on a dedicated HTTP server.
- **Mobile**: Applications that are installed on smart phones and tablets.
- **Non-browser Mobile Applications**: Applications 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]


ZOHO Developer Console
Patricia Boyle
Choose a Client Type
Java Script
Applications that run exclusively on a browser and are independent of a web server.
CREATE NOW
Web based
Applications that are clients running on a dedicated HTTP server.
CREATE NOW
Mobile
Applications that are installed on smart phones and tablets.
CREATE NOW
Non-browser Mobile Application
Applications for devices without browser provisioning such as smart TVs and printers.
CREATE NOW
Self Client
Credentials can be generated for testing and to handle server-to-server application.
CREATE NOW

3. 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.



Create New Client

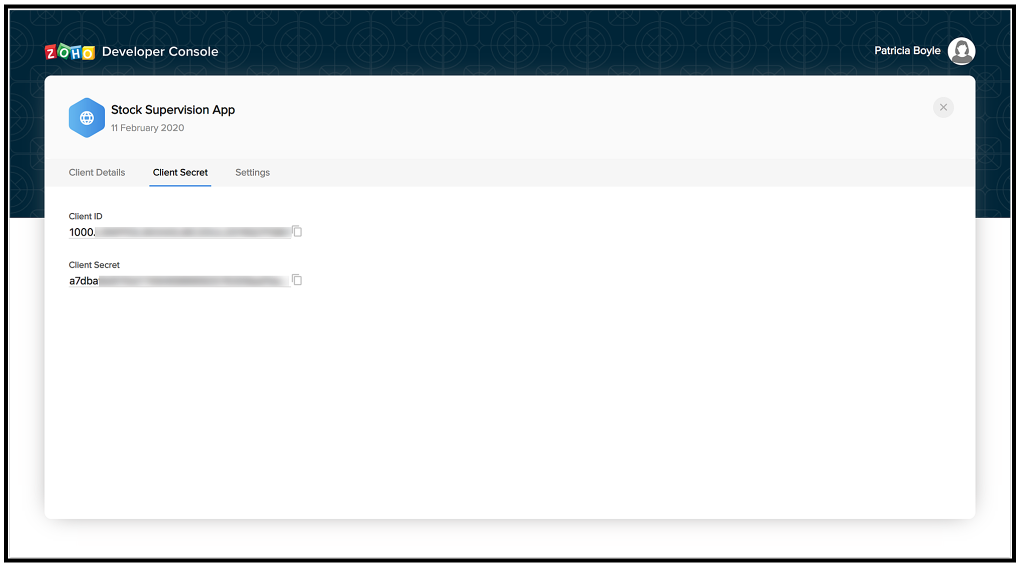
Client Type
Web based

Client Name
Stock Supervision App

Homepage URL
https://{your_domain}.com

Authorized redirect URIs
https://{your_domain}.com/{your_redirect_page}

4. 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.



Stock Supervision App
11 February 2020

Client ID
1000.[redacted]

Client Secret
a7dba[redacted]

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.

Configuration

Before you get started with creating your Java 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.

```java
/*
   * 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.
   */
  Logger logger = Logger.getInstance(Levels.INFO, "absolute_file_path");
```

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

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

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

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

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

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

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

```
/*
   * 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"
*/
//TokenStore tokenstore = new DBStore();

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

//TokenStore tokenstore = new
FileStore("absolute_file_path");

//TokenStore tokenStore = new CustomStore();
```

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

```
/*
 * 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(com.zoho.crm.api.util.ModuleFieldsHandler)
 *
 * 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
*/
SDKConfig sdkConfig = new SDKConfig.Builder().setAutoRefreshFields(false).setPickListValidation(true).build();
```

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

```
String resourcePath = "/Users/user_name/Documents";
```

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

```
RequestProxy requestProxy = new RequestProxy("proxyHost", proxyPort, "proxyUser", "password", "userDomain");
```

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.

Implementing OAuth Persistence

Once the application is authorized, 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** interface, which has the following callback methods.

- **getToken(UserSignature user, Token token)** - invoked before firing a request to fetch the saved tokens. This method should return implementation Token interface object for the library to process it.
- **saveToken(UserSignature user, 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.

There are three ways provided by the SDK in which you can achieve persistence. They are:

1. 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))
  - -- access_token (varchar(255))

– refresh_token (varchar(255))
– grant_token (varchar(255))
– expiry_time (varchar (20))

Here is the **MySQL** query:

```
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));
alter table oauthtoken auto_increment = 1;
```

Here is the code to create a **DBStore** object:

```
/*
* 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"
*/
TokenStore tokenstore = new DBStore();
//TokenStore interface
TokenStore tokenstore = new DBStore("hostName", "dataBaseName", "userName", "password", "portNumber");
```

2. 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:

```
//Parameter containing the absolute file path to store tokens
TokenStore tokenstore = new FileStore("/Users/username/Documents);
```

3. Custom Persistence

To use **Custom Persistence**, you must implement the **TokenStore** interface **(com.zoho.api.authenticator.store.TokenStore)** and override the methods.

Here is the code:

```
package user.store;

import com.zoho.api.authenticator.Token;
import com.zoho.api.exception.SDKException;
import com.zoho.crm.api.UserSignature;
import com.zoho.api.authenticator.store.TokenStore;
public class CustomStore implements TokenStore
{
 /**
  * @param user A UserSignature class instance.
  * @param token A Token (com.zoho.api.authenticator.OAuthToken) class instance.
  * @return A Token class instance representing the user token details.
  * @throws SDKException if any problem occurs.
  */
 @Override
 public Token getToken(UserSignature user, Token token) throws SDKException
 {
 // Add code to get the token
```

```
return null;
}

/**
 * @param user A UserSignature class instance.
 * @param token A Token
(com.zoho.api.authenticator.OAuthToken) class instance.
 * @throws SDKException if any problem occurs.
 */
@Override
public void saveToken(UserSignature user, Token token)
throws SDKException
{
// Add code to save the token
}

/**
 * @param token A Token
(com.zoho.api.authenticator.OAuthToken) class instance.
 * @throws SDKException if any problem occurs.
 */
@Override
public void deleteToken(Token token) throws SDKException
{
// Add code to delete the token
}

@Override
public List<Token> getTokens() throws SDKException
{
 // Add code to get the all stored tokens
 }

@Override
public void deleteTokens() throws SDKException
{
 // Add code to delete the all stored token
}
}
```

Initialization

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 the User's 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.

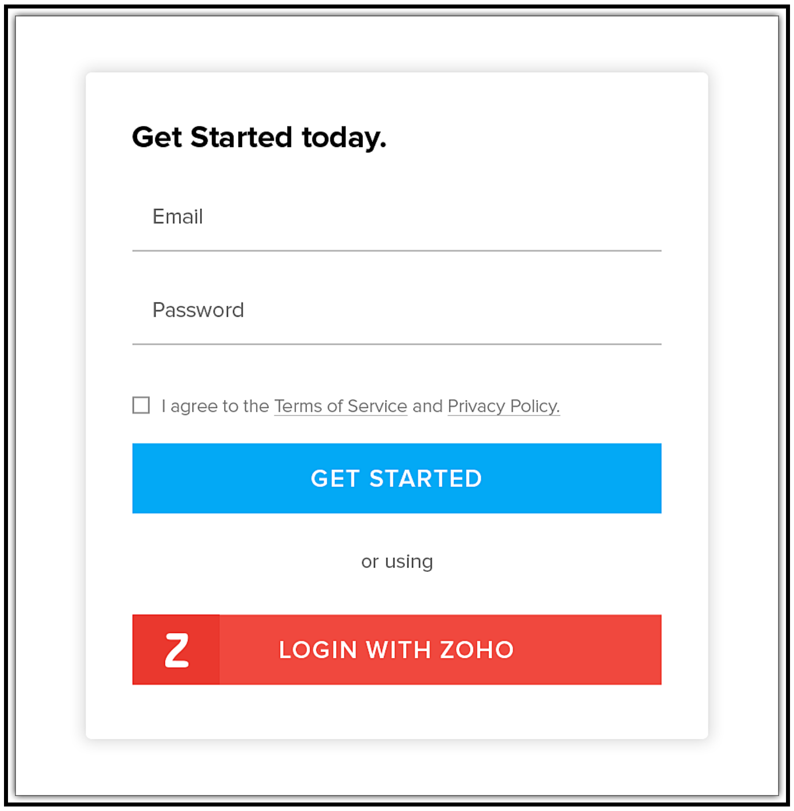
Note:

- 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.

Initialization

You must pass the following details to the SDK and initialize it before you can make API calls.

1. UserSignature - The email ID of the user that is making the API calls. The tokens are also specific to this user.
2. Environment - The environment such as **Production, Developer, or Sandbox** from which you want to make API calls. This instance also takes the domain (data center) in which the tokens are generated. The format is **USDataCenter.PRODUCTION, EUDataCenter.SANDBOX** and so on.
3. Token - The grant or refresh token. The token must be specific to the user that makes the call, and specific to the org and the environment the token was generated in. Besides the token, the token instance also takes the **client ID, client secret, and the redirect URI** as its parameters.
4. Tokenstore - The token persistence method. The possible methods are DB persistence and file persistence. For file persistence, you must specify the absolute file path to the file where you want to store the tokens. For DB persistence, you must specify the **host, database name, user name, password** and the **port** at which the server runs.
5. Logger - To log the messages. You can choose the level of logging of messages through "**Logger.Levels**", and provide the absolute file path to the file where you want the SDK to write the messages in.
6. SDKConfig - The class that contains the values of autoRefresh and pickListValidation fields.
7. resourcePath - The absolute directory path to store user-specific files containing information about the fields of a module.
8. RequestProxy - An instance containing the proxy details of the request.

Note

- From version 3.x.x, initialization of the SDK happens through the Initializer class. This class contains instances of the current user, environment, token, tokenstore, and logger.
- Initializing the SDK does not generate a token. A token is generated only when you make an API call.

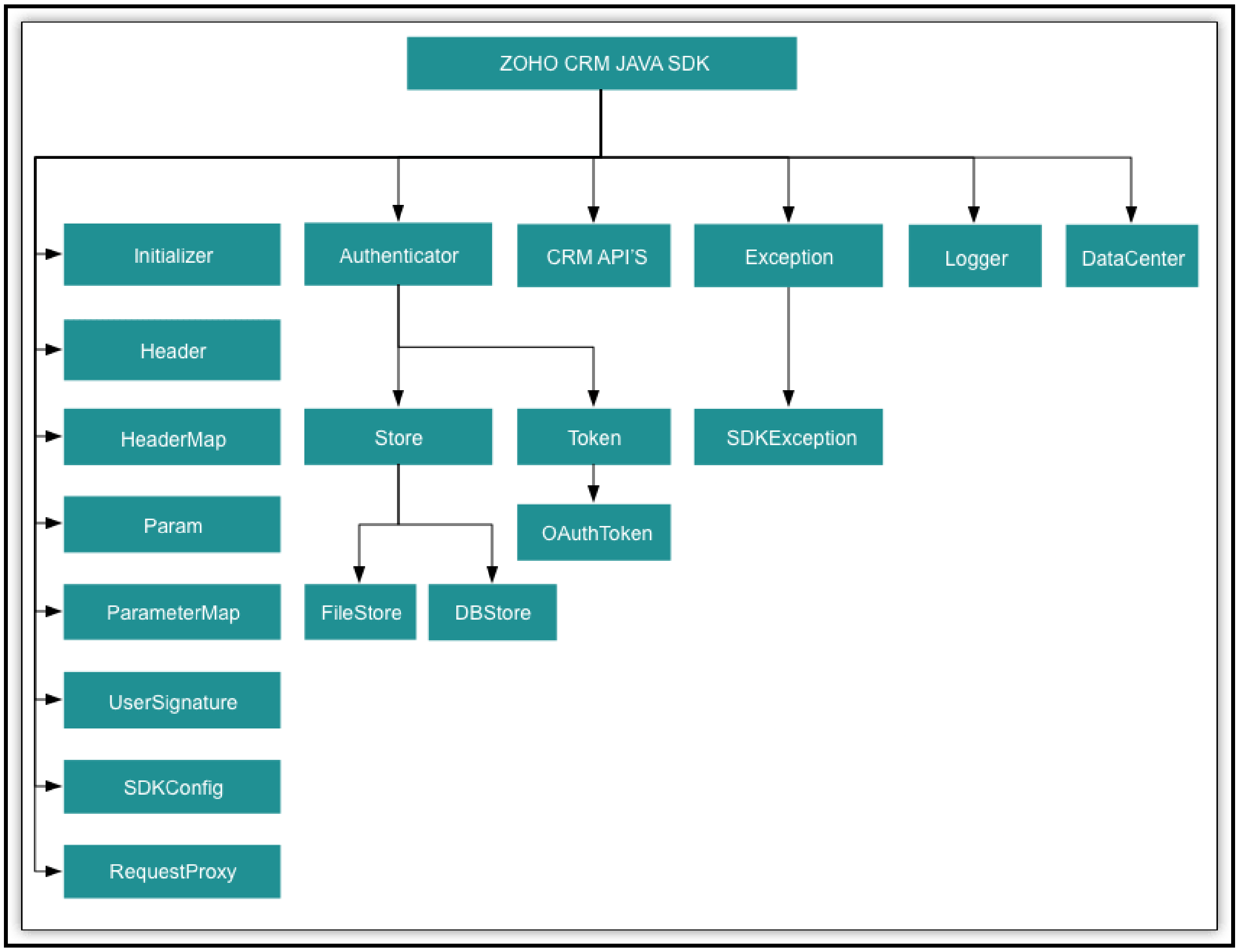
```java
package com.zoho.crm.sample.initializer;

import com.zoho.api.authenticator.OAuthToken;
import com.zoho.api.authenticator.Token;
import com.zoho.api.authenticator.OAuthToken.TokenType;
import com.zoho.api.authenticator.store.DBStore;
import com.zoho.api.authenticator.store.TokenStore;

import com.zoho.crm.api.Initializer;
import com.zoho.crm.api.RequestProxy;
import com.zoho.crm.api.UserSignature;
import com.zoho.crm.api.dc.DataCenter.Environment;
import com.zoho.crm.api.dc.USDataCenter;
import com.zoho.api.logger.Logger;
import com.zoho.api.logger.Logger.Levels;
import com.zoho.crm.api.SDKConfig;


public class Initialize
{
    public static void main(String[] args) throws Exception
    {
        initialize();
    }
    public static void initialize() throws Exception
    {
        /*
        * 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.
        */
        Logger logger = Logger.getInstance(Levels.INFO, "/Users/user_name/Documents/java_sdk_log.log");

        //Create an UserSignature instance that takes user Email as parameter
        UserSignature 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
        */
        Environment environment = USDataCenter.PRODUCTION;

        /*
        * Create a Token instance
        * 1 -> OAuth client id.
        * 2 -> OAuth client secret.
        * 3 -> REFRESH/GRANT token.
        * 4 -> Token type(REFRESH/GRANT).
        * 5 -> OAuth redirect URL.
        */
        Token token = new OAuthToken("clientId", "clientSecret", "REFRESH/GRANT token", TokenType.REFRESH/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"
        */
        //TokenStore tokenstore = new DBStore();
        TokenStore tokenstore = new DBStore("hostName", "dataBaseName", "userName", "password", "portNumber");

        //TokenStore tokenstore = new FileStore("absolute_file_path");

        /*
         * 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(com.zoho.crm.api.util.ModuleFieldsHandler)
         *
         * 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.
         */
        SDKConfig sdkConfig = new SDKConfig.Builder().setAutoRefreshFields(false).setPickListValidation(true).build();

        String resourcePath = "/Users/user_name/Documents/javasdk-application";

        /**
         * Create an instance of RequestProxy class that takes the following parameters
         * 1 -> Host
         * 2 -> Port Number
         * 3 -> User Name
         * 4 -> Password
         * 5 -> User Domain
         */
        // RequestProxy requestProxy = new RequestProxy("proxyHost", "proxyPort", "proxyUser", "password");

        RequestProxy requestProxy = new RequestProxy("proxyHost", "proxyPort", "proxyUser", "password", "userDomain");

        /*
        * The 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 -A String
         * 7 -> Logger instance
         * 8 -> RequestProxy instance
         */

        // The following are the available initialize methods

        Initializer.initialize(user, environment, token, tokenstore, sdkConfig, resourcePath);

        Initializer.initialize(user, environment, token, tokenstore, sdkConfig, resourcePath, logger);

        Initializer.initialize(user, environment, token, tokenstore, sdkConfig, resourcePath, requestProxy);

        Initializer.initialize(user, environment, token, tokenstore, sdkConfig, resourcePath, logger, requestProxy);
    }
}
```

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 Java SDK is depicted in the following image



Responses and Exceptions

APIResponse<ResponseHandler> and **APIResponse<ActionHandler>** are the wrapper objects for Zoho CRM APIs' responses. All API calling methods would return one of these two objects.

Use the **getObject()** method to obtain the response handler interface depending on the type of request **(GET, POST,PUT,DELETE).**

The following are the wrappers with their handlers for the respective APIs:

- Tag API's record and count tag operation:

-**APIResponse<RecordActionHandler>**
-**APIResponse<CountHandler>**

- BaseCurrency operation:
  -**APIResponse<BaseCurrencyActionHandler>**

- Lead convert operation:
  -**APIResponse<ConvertActionHandler>**

- Deleted record operation:
  -**APIResponse<DeletedRecordsHandler>**

- Download record's photo operation:
  -**APIResponse<DownloadHandler>**

- MassUpdate record operation:
  -**APIResponse<MassUpdateActionHandler>**
  -**APIResponse<MassUpdateResponseHandler>**

For GET Requests

- **APIResponse.getObject()** returns the response handler interface.
- This ResponseHandler interface encompasses the **ResponseWrapper class** (for **application/json** responses), **file body wrapper class** (for file download responses), and the **APIException class**.
- This CountHandler interface encompasses the **CountWrapper class** (for **application/json** responses) and the **APIException class**.
- This DeletedRecordsHandler interface encompasses the **DeletedRecordsWrapper** class (for **application/json** responses) and the **APIException class**.
- This DownloadHandler interface encompasses the **FileBodyWrapper class** (for file download responses responses) and the **APIException class**.
- This MassUpdateResponseHandler interface encompasses the **MassUpdateResponseWrapper class** (for **application/json** responses) and the **APIException class**.

For POST, PUT, DELETE Requests

- **APIResponse.getObject()** returns the ActionHandler interface.
- The ActionHandler interface encompasses the **ActionWrapper class** (for **application/json** responses) and the **APIException class**.
- The ActionResponse interface encompasses the **SuccessResponse class** (for **application/json** responses) and the **APIException class.**
- The RecordActionHandler interface encompasses the **RecordActionWrapper** class (for **application/json** responses), and the **APIException class**.
- The **BaseCurrencyActionHandler** interface encompasses the BaseCurrencyActionWrapper class (for **application/json** responses), and the **APIException class**.
- The MassUpdateActionHandler interface encompasses the **MassUpdateActionWrapper class** (for **application/json** responses), and the **APIException class**.
- The ConvertActionHandler interface encompasses the **ConvertActionWrapper** class (for **application/json** responses), and the **APIException class**.
- If the root key of the response is not "**data**" (errors such as **Internal Server Error**), then the ActionResponse interface with either the SuccessResponse class or APIException class is returned.

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

Sample Codes

All of **Zoho CRM's APIs** can be used through the **Java 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(String moduleAPIName,	Creates an AttachmentsOperations class instance

String recordId)	with the moduleAPIName and recordId.

Method	**Return Type**	**Description**
getAttachments	APIResponse<ResponseHandler>	To fetch the list of attachments of a record.
uploadAttachments	APIResponse<ActionHandler>	To upload attachments to a record.
deleteAttachments	APIResponse<ActionHandler>	To delete the attachments that were added to a record.
deleteAttachment	APIResponse<ActionHandler>	To delete an attachment that was added to a record.
downloadAttachment	APIResponse<ResponseHandler>	To download an attachment that was uploaded to a record.
uploadLinkAttachments	APIResponse<ActionHandler>	To upload a link as an attachment to a record.

Blueprint Operations

Constructor	**Description**

BluePrintOperations(String recordId, String moduleAPIName)	Creates a BluePrintOperations class instance with the recordId and moduleAPIName.

Method	Return Type	Description
getBlueprint	APIResponse<ResponseHandler>	To get the next available transitions for that record, fields available for each transition, current value of each field, and validation(if any).
updateBlueprint	APIResponse<ActionResponse>	To update a single transition at a time.

Bulk Read Operations

Method	Return Type	Description
createBulkReadJob	APIResponse<ActionHandler>	To schedule a bulk read job to export records that match the criteria.
getBulkReadJobDetails	APIResponse<ResponseHandler>	To know the status of the bulk read job scheduled previously.

downloadResult	APIResponse<ResponseHandler>	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	Return Type	Description
uploadFile	APIResponse<ActionResponse>	To upload a CSV file in ZIP format. The response contains the "file_id". Use this ID while making the bulk write request.
createBulkWriteJob	APIResponse<ActionResponse>	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.
getBulkReadJobDetails	APIResponse<ResponseWrapper>	To know the status of the bulk read job scheduled previously.
	APIResponse<ResponseHa	To download the result of

downloadResult	ndler>	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

Contact Roles Operations

Method	Return Type	Description
getContactRoles	APIResponse<ResponseHandler>	To get the list of all contact roles.
createContactRoles	APIResponse<ActionHandler>	To create contact roles.
updateContactRoles	APIResponse<ActionHandler>	To update contact roles.
deleteContactRoles	APIResponse<ActionHandler>	To delete contact roles.
getContactRole	APIResponse<ResponseHandler>	To get specific contact role.
updateContactRole	APIResponse<ActionHandler>	To update specific contact role.
	APIResponse<ActionHandl	To delete specific contact

deleteContactRole	er>	role.

Currencies Operations

Method	Return Type	Description
getCurrencies	APIResponse<ResponseHandler>	To get the list of all currencies available for your org.
addCurrencies	APIResponse<ActionHandler>	To add new currencies to your org.
updateCurrencies	APIResponse<ActionHandler>	To update the currencies' details of your org.
enableMultipleCurrencies	APIResponse<BaseCurrencyAction Handler>	To enable multiple currencies for your org.
updateBaseCurrency	APIResponse<BaseCurrencyAction Handler>	To update the base currency details of your org.
getCurrency	APIResponse<ResponseHandler>	To get the details of specific currency.
updateCurrency	APIResponse<ActionHandler>	To update the details of specific currency.

Custom View Operations

Constructor	Description
CustomViewsOperations(String module)	Creates a CustomViewsOperations class instance with the moduleAPIName.

Method	Return Type	Description
getCustomViews	APIResponse<ResponseHandler>	To get the list of all custom views in a module.
getCustomView	APIResponse<ResponseHandler>	To get the details of specific custom view in a module.

Fields Metadata Operations

Constructor	Description
FieldsOperations(String module)	Creates a FieldsOperations class instance with the module.

Method	Return Type	Description
getFields	APIResponse<ResponseHandler>	To get the meta details of all fields in a module.

getField	APIResponse<ResponseHandler>	To get the meta details of specific field in a module.

Files Operations

Method	Return Type	Description
uploadFiles	APIResponse<ActionHandler>	To upload files and get their encrypted IDs.
getFile	APIResponse<ResponseHandler>	To get the uploaded file through its encrypted ID.

Layouts Operations

Constructor	Description
LayoutsOperations(String module)	Creates a LayoutsOperations class instance with the moduleAPIName.

Method	Return Type	Description
getLayouts	APIResponse<ResponseHandler>	To get the details of all the layouts in a module.
getLayout	APIResponse<ResponseHandler>	To get the details (metadata) of a specific layout in a module.

Modules Operations

Method	Return Type	Description
getModules	APIResponse<ResponseHandler>	To get the details of all the modules.
getModule	APIResponse<ResponseHandler>	To get the details (metadata) of a specific module.
updateModuleByAPIName	APIResponse<ActionHandler>	To update the details of a module by its module API name.
updateModuleById	APIResponse<ActionHandler>	To update the details of a module by its ID.

Notes Operations

Method	Return Type	Description
getNotes	APIResponse<ResponseHandler>	To get the list of notes of a record.
createNotes	APIResponse<ActionHandler>	To add new notes to a record.
updateNotes	APIResponse<ActionHandl	To update the details of the

	er>	notes of a record.
deleteNotes	APIResponse<ActionHandler>	To delete the notes of a record.
getNote	APIResponse<ResponseHandler>	To get the details of a specific note.
updateNote	APIResponse<ActionHandler>	To update the details of an existing note.
deleteNote	APIResponse<ActionHandler>	To delete a specific note.

Notification Operations

Method	Return Type	Description
enableNotifications	APIResponse<ActionHandler>	To enable instant notifications of actions performed on a module.
getNotificationDetails	APIResponse<ResponseHandler>	To get the details of the notifications enabled by the user.
updateNotifications	APIResponse<ActionHandler>	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	APIResponse<ActionHandler>	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	APIResponse<ActionHandler>	To stop all the instant notifications enabled by the user for a channel.
disableNotification	APIResponse<ActionHandler>	To disable notifications for the specified events in a channel.

Organization Operations

Method	**Return Type**	**Description**
getOrganization	APIResponse<ResponseHandler>	To get the details of your organization.
uploadOrganizationPhoto	APIResponse<ActionHandler>	To upload a photo of your organization.

Profile Operations

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

Method	Return Type	Description
getProfiles	APIResponse<ResponseHandler>	To get the list of profiles available for your organization.
getProfile	APIResponse<ResponseHandler>	To get the details of a specific profile.

Query (COQL) Operation

Method	Return Type	Description
getRecords	APIResponse<ResponseHandler>	To get the records from a module through a COQL query.

Records Operations

Method	Return Type	Description

getRecord	APIResponse<ResponseHandler>	To get a specific record from a module.
updateRecord	APIResponse<ActionHandler>	To update a specific record in a module.
deleteRecord	APIResponse<ActionHandler>	To delete a specific record from a module.
getRecords	APIResponse<ResponseHandler>	To get all records from a module.
createRecords	APIResponse<ActioneHandler>	To insert records in a module.
updateRecords	APIResponse<ActionHandler>	To update records in a module.
deleteRecords	APIResponse<ActionHandler>	To delete records from a module.
upsertRecords	APIResponse<ActionHandler>	To insert/update records in a module.
getDeletedRecords	APIResponse<DeletedRecords Handler>	To get the deleted records from a module.
searchRecords	APIResponse<ActionHandler>	To search for records in a module that match certain criteria, email, phone

		number, or a word.
convertLead	APIResponse<ConvertActionHandler>	To convert records(Leads to Contacts/Deals).
getPhoto	APIResponse<DownloadHandler>	To get the photo of a record.
uploadPhoto	APIResponse<FileHandler>	To upload a photo to a record.
deletePhoto	APIResponse<FileHandler>	To delete the photo of a record.
massUpdateRecords	APIResponse<MassUpdate Action Handler>	To update the same field for multiple records in a module.
getMassUpdateStatus	APIResponse<MassUpdate Action Handler>	To get the status of the mass update job scheduled previously.

Related List Operations

Constructor	Description
RelatedListsOperations(String module)	Creates a RelatedListsOperations class instance with the moduleAPIName.

Method	Return Type	Description
getRelatedLists	APIResponse<ResponseHandler>	To get the details of all the related lists of a module.
getRelatedList	APIResponse<ResponseHandler>	To get the details of a specific related list of a module.

Related Records Operations

Constructor	Description
RelatedRecordsOperations(String relatedListAPIName, Long recordId, String moduleAPIName)	Creates a RelatedRecordsOperations class instance with the relatedListAPIName, recordId, and moduleAPIName.

Method	Return Type	Description
getRelatedRecords	APIResponse<ResponseHandler>	To get list of records from the related list of a module.
updateRelatedRecords	APIResponse<ActionHandler>	To update the association/relation between the records.
delinkRecords	APIResponse<ActionHandler>	To delete the association between the records.

getRelatedRecord	APIResponse<ResponseHandler>	To get the records from a specific related list of a module.
updateRelatedRecord	APIResponse<ActionHandler>	To update the details of a specific record of a related list in a module.
delinkRecord	APIResponse<ActionHandler>	To delete a specific record from the related list of a module.

Role Operations

Method	Return Type	Description
getRoles	APIResponse<ResponseHandler>	To get the list of all roles available in your organization.
getRole	APIResponse<ResponseHandler>	To get the details of a specific role.

Shared Records Operations

Constructor	Description
ShareRecordsOperations(Long recordId, String moduleAPIName)	Creates a ShareRecordsOperations class instance with the recordId and

	moduleAPIName.

Method	Return Type	Description
getSharedRecordDetails	APIResponse<ResponseHandler>	To get the details of a record shared with other users.
shareRecord	APIResponse<ActionHandler>	To share a record with other users in the organization.
updateSharePermissions	APIResponse<ActionHandler>	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	APIResponse<DeleteActionHandler>	To revoke access to a shared record.

Tags Operations

Method	Return Type	Description
getTags	APIResponse<ResponseHandler>	To get the list of all tags in your organization.
createTags	APIResponse<ActionHandler>	To create tags.
updateTags	APIResponse<ActionHandler>	To update multiple tags.
updateTag	APIResponse<ActionHandler>	To update a specific tag.
deleteTag	APIResponse<ActionHandler>	To delete a specific tag from the module.
mergeTags	APIResponse<ActionHandler>	To merge two tags.
addTagsToRecord	APIResponse<RecordActionHandler>	To add tags to a specific record.

removeTagsFromRecord	APIResponse<RecordActionHandler>	To remove tags from a record.
addTagsToMultipleRecords	APIResponse<RecordActionHandler>	To add tags to multiple records.
removeTagsFromMultipleRecords	APIResponse<RecordActionHandler>	To remove tags from multiple records.
getRecordCountForTag	APIResponse<RecordActionHandler>	To get the record count for a tag.

Taxes Operations

Method	Return Type	Description
getTaxes	APIResponse<ResponseHandler>	To get the taxes of your organization.
createTaxes	APIResponse<ActionHandler>	To add taxes to your organization.
updateTaxes	APIResponse<ActionHandler>	To update the existing taxes of your organization.
deleteTaxes	APIResponse<ActionHandler>	To delete multiple taxes from your organization.
	APIResponse<ResponseHa	To get the details of a

getTax	ndler>	specific tax.
deleteTax	APIResponse<ActionHandler>	To delete a specific tax from your organization.

Territory Operations

Method	Return Type	Description
getTerritories	APIResponse<ResponseHandler>	To get the list of all territories.
getTerritory	APIResponse<ResponseHandler>	To get the details of a specific territory.

Users Operations

Method	Return Type	Description
getUsers	APIResponse<ResponseHandler>	To get the list of users in your organization.
createUser	APIResponse<ActionHandler>	To add a user to your organization.
updateUsers	APIResponse<ActionHandler>	To update the existing users of your organization.

getUser	APIResponse<ResponseHandler>	To get the details of a specific user.
updateUser	APIResponse<ActionHandler>	To update the details of a specific user.
deleteUser	APIResponse<ActionHandler>	To delete a specific user from your organization.

Variable Groups Operations

Method	Return Type	Description
getVariableGroups	APIResponse<ResponseHandler>	To get the list of all variable groups available for your organization.
getVariableGroupById	APIResponse<ResponseHandler>	To get the details of a variable group by its group ID.
getVariableGroupByAPIName	APIResponse<ResponseHandler>	To get the details of a specific variable group by its API name.

Variables Operations

Method	Return Type	Description

getVariables	APIResponse<ResponseHandler>	To get the list of variables available for your organization.
createVariables	APIResponse<ActionHandler>	To add new variables to your organization.
updateVariables	APIResponse<ActionHandler>	To update the details of variables.
deleteVariables	APIResponse<ActionHandler>	To delete multiple variables.
getVariableById	APIResponse<ResponseHandler>	To get the details of a specific variable by its unique ID.
updateVariableById	APIResponse<ActionHandler>	To update the details of a specific variable by its unique ID.
deleteVariable	APIResponse<ActionHandler>	To delete a specific variable.
getVariableForAPIName	APIResponse<ResponseHandler>	To get the details of a variable by its API name.
updateVariableByAPIName	APIResponse<ActionHandler>	To update the details of a variable by its API name.

Threading in the Java SDK

Threads in a Java program help you achieve parallelism. By using multiple threads, you can make a Java program run faster and do multiple things at the same time.

The Java SDK (from version 3.x.x) supports both single-threading and multi-threading irrespective of a single user or a multi user app.

Refer to the below code snippets that use multi-threading for a single-user and multi-user app.

Multi-threading in a Multi-user App

- The program execution starts from main().
- The details of "user1" are given in the variables user1, token1, environment1.
- Similarly, the details of another user "user2" are given in the variables user2, token2, environment2.
- For each user, an instance of MultiThread class is created.
- When start() is called which in-turn invokes run(), the details of user1 are passed to the switchUser function through the MultiThread object. Therefore, this creates a thread for user1.
- Similarly, When start() is invoked again, the details of user2 are passed to the switchUser function through the MultiThread object. Therefore, this creates a thread for user2.

```java
package com.zoho.crm.sample.threading.multiuser;

import com.zoho.api.authenticator.OAuthToken;
import com.zoho.api.authenticator.Token;
import com.zoho.api.authenticator.OAuthToken.TokenType;
import com.zoho.api.authenticator.store.DBStore;
import com.zoho.api.authenticator.store.TokenStore;
import com.zoho.api.logger.Logger;
import com.zoho.crm.api.Initializer;
import com.zoho.crm.api.RequestProxy;
import com.zoho.crm.api.SDKConfig;
import com.zoho.crm.api.UserSignature;
import com.zoho.crm.api.dc.USDataCenter;
import com.zoho.crm.api.dc.DataCenter.Environment;
import com.zoho.crm.api.exception.SDKException;
import com.zoho.crm.api.record.RecordOperations;
import com.zoho.crm.api.util.APIResponse;

public class MultiThread extends Thread {
    Environment environment;

    UserSignature user;

    Token token;

    String moduleAPIName;

    RequestProxy userProxy;

    SDKConfig sdkConfig;

    public MultiThread(UserSignature user, Environment environment, Token token, String moduleAPIName, SDKConfig config, RequestProxy proxy) {
        this.environment = environment;

        this.user = user;

        this.token = token;

        this.moduleAPIName = moduleAPIName;

        this.sdkConfig = config;

        this.userProxy = proxy;
    }

    public void run() {
        try {
            Initializer.switchUser(user, environment, token,
sdkConfig, userProxy);


System.out.println(Initializer.getInitializer().getUser().getEmail());

            RecordOperations cro = new RecordOperations();

            @SuppressWarnings("rawtypes")
            APIResponse getResponse =
cro.getRecords(this.moduleAPIName, null, null);

            System.out.println(getResponse.getObject());

        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    public static void main(String[] args) throws SDKException {

        Logger loggerInstance =
Logger.getInstance(Logger.Levels.ALL, "/Users/


abc-1234/Documents/AutomateSDK/java/GitLab/sdk.log");

        Environment env = USDataCenter.PRODUCTION;

        UserSignature user1 = new UserSignature("abc@xyz.com");

        TokenStore tokenstore = new DBStore(null, null, null,
"abc@1234", null);

        Token token1 = new OAuthToken("1000xxx0e16",
"1000xxxe83", TokenType.REFRESH, "https://www.zoho.com");

        String resourcePath =
"/Users/username/Documents/zohocrm-javasdk-sample-application";
```

```

        SDKConfig user1Config = new SDKConfig.Builder().setAutoRefreshFields(false).setPickListValidation(true).build();

        Initializer.initialize(user1, env, token1, tokenstore, user1Config, resourcePath, loggerInstance);

        MultiThread multiThread = new MultiThread(user1, env, token1, "Students", user1Config, null);

        multiThread.start();

        Environment environment = USDataCenter.PRODUCTION;

        UserSignature user2 = new UserSignature("user2@xyz.com");

        Token token2 = new OAuthToken("1000xxxda7f", "1000xxxxa42", TokenType.REFRESH);

        RequestProxy user2Proxy = new RequestProxy("proxyHost", 80, "proxyUser", "password", "userDomain");

        SDKConfig user2Config = new SDKConfig.Builder().setAutoRefreshFields(true).setPickListValidation(false).build();

        multiThread = new MultiThread(user2, environment, token2, "Leads", user2Config, user2Proxy);

        multiThread.start();

    }
}
```

Multi-threading in a Single-user App

```
package threading.singleuser;
```

```java
import com.zoho.api.authenticator.OAuthToken;
import com.zoho.api.authenticator.Token;
import com.zoho.api.authenticator.OAuthToken.TokenType;
import com.zoho.api.authenticator.store.DBStore;
import com.zoho.api.authenticator.store.TokenStore;
import com.zoho.api.exception.SDKException;
import com.zoho.crm.api.Initializer;
import com.zoho.crm.api.UserSignature;
import com.zoho.crm.api.dc.USDataCenter;
import com.zoho.crm.api.dc.DataCenter.Environment;
import com.zoho.crm.api.logger.Logger;
import com.zoho.crm.api.record.RecordOperations;
import com.zoho.crm.api.util.APIResponse;

public class MultiThread extends Thread
{
    String moduleAPIName;

    public MultiThread(String moduleAPIName)
    {
        this.moduleAPIName = moduleAPIName;
    }

    public void run()
    {
        try
        {
            RecordOperations record = new RecordOperations();

            @SuppressWarnings("rawtypes")
            APIResponse getResponse = record.getRecords(null, null, this.moduleAPIName);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) throws SDKException
    {
        Logger logger = Logger.getInstance(Levels.INFO,
"/Users/user_name/Documents/java-sdk-logs.log");

        Environment environment = USDataCenter.PRODUCTION;

        TokenStore tokenStore = new
FileStore("/Users/user_name/Documents/java-sdk-tokens.txt");

        UserSignature user = new UserSignature("user1@zoho.com");

        Token token = new OAuthToken("clientId1",
"clientSecret1", "REFRESH/GRANT token", TokenType.REFRESH/GRANT);

        Boolean autoRefreshFields = true;

        String resourcePath =
"/Users/user_name/Documents/javasdk-application";

        Initializer.initialize(user1, environment, token1,
tokenStore, autoRefreshFields, resourcePath, logger);

        MultiThread mtsu = new MultiThread("Deals");

        mtsu.start();

        mtsu = new MultiThread("Leads");

        mtsu.start();
    }
}
```

Single-threading in a Multi-user App

```java
package com.zoho.crm.sample.threading.multiuser;
import com.google.gson.Gson;
import com.zoho.api.authenticator.OAuthToken;
import com.zoho.api.authenticator.Token;
import com.zoho.api.authenticator.OAuthToken.TokenType;
import com.zoho.api.authenticator.store.DBStore;
import com.zoho.api.authenticator.store.TokenStore;
import com.zoho.crm.api.Initializer;
import com.zoho.crm.api.UserSignature;
import com.zoho.crm.api.dc.USDataCenter;
import com.zoho.crm.api.dc.DataCenter.Environment;
import com.zoho.crm.api.logger.Logger;
import com.zoho.crm.api.record.RecordOperations;
import com.zoho.crm.api.util.APIResponse;

public class SingleThread
{
  Environment environment;

  UserSignature user;

  Token token;

  String moduleAPIName;

  public SingleThread( String moduleAPIName)
  {
        this.moduleAPIName = moduleAPIName;
  }

  public SingleThread(UserSignature user, Environment environment, Token token,  String moduleAPIName)
  {
        this.environment= environment;

        this.user = user;

        this.token = token;

        this.moduleAPIName = moduleAPIName;
  }
```

```java

public void run()
  {
     try
     {
Initializer.switchUser(user, environment, token, false);


System.out.println(Initializer.getInitializer().getUser().getEmail());

RecordOperations cro = new RecordOperations();

    Gson gson = new Gson();

    @SuppressWarnings("rawtypes")
        APIResponse getResponse = cro.getRecords(null, null, this.moduleAPIName);
    System.out.println(gson.toJson(getResponse.getObject()));

     }
     catch (Exception e)
     {
        e.printStackTrace();
     }
  }

public static void main(String[] args) throws Exception
{
    Logger loggerInstance = Logger.getInstance(Logger.Levels.ALL, "/Users/username/Documents");

    Environment env = USDataCenter.PRODUCTION;

    UserSignature user1 = new UserSignature("p.boyle@abc.com");

    TokenStore tokenstore = new DBStore(null, null, null,
"password", null);

        Token token1 = new OAuthToken("1000xxxxxKI3DH",
"5500xxxxxx94cb2a", "https://www.zoho.com", "1000xxxxxxfc2401",
TokenType.REFRESH);

        String resourcePath = "/Users/username/Documents";

        Initializer.initialize(user1, env, token1, tokenstore,
sdkConfig, resourcePath, loggerInstance);

        SingleThread singleThread = new SingleThread(user1, env,
token1, "Students");

        singleThread.run();

        Environment environment = USDataCenter.PRODUCTION;

        UserSignature user2 = new UserSignature("boyle1@abc.com");

        Token token2 = new OAuthToken("1000xxxxxYZX3",
"efd63xxxxxxe786c", "https://www.zoho.com", "1000xxxxxx4286ad",
TokenType.REFRESH);

        singleThread = new SingleThread(user2, environment,
token2, "Leads");

        singleThread.run();
  }
}
```

Single-threading in a Single-user App

```
package com.zoho.crm.sample.threading.singleuser;

import com.google.gson.Gson;
import com.zoho.api.authenticator.OAuthToken;
import com.zoho.api.authenticator.Token;
```

```java
import com.zoho.api.authenticator.OAuthToken.TokenType;
import com.zoho.api.authenticator.store.DBStore;
import com.zoho.api.authenticator.store.TokenStore;
import com.zoho.api.exception.SDKException;
import com.zoho.crm.api.Initializer;
import com.zoho.crm.api.UserSignature;
import com.zoho.crm.api.dc.USDataCenter;
import com.zoho.crm.api.dc.DataCenter.Environment;
import com.zoho.crm.api.logger.Logger;
import com.zoho.crm.api.record.RecordOperations;
import com.zoho.crm.api.util.APIResponse;

public class SingleThread extends Thread
{
 String moduleAPIName;

 public SingleThread(String moduleAPIName)
 {
     this.moduleAPIName = moduleAPIName;
 }

 public void run()
   {
      try
      {
System.out.println(Initializer.getInitializer().getUser().getEmail());

 RecordOperations cro = new RecordOperations();

     Gson gson = new Gson();

     @SuppressWarnings("rawtypes")
         APIResponse getResponse = cro.getRecords(null, null, this.moduleAPIName);
     System.out.println(gson.toJson(getResponse.getObject()));

      }
```

```java
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

 public static void main(String[] args) throws SDKException
 {

        Logger loggerInstance = 
Logger.getInstance(Logger.Levels.ALL, 
"/Users/username/Documents");

        Environment env = USDataCenter.PRODUCTION;

        UserSignature user = new UserSignature("p.boyle@abc.com");

        TokenStore tokenstore = new DBStore(null, null, null, 
"password", null);

        Token token1 = new OAuthToken("1000xxxxx3DH", 
"5500xxxxxxcb2a", "https://www.zoho.com", "1000xxxxxa7fc2401", 
TokenType.REFRESH);

        String resourcePath = "/Users/username/Documents";

        Initializer.initialize(user, env, token1, tokenstore, 
sdkConfig, resourcePath, loggerInstance, userProxy);

        SingleThread stsu = new SingleThread("Leads");

        stsu.start();
 }
}
```

Release Notes

Current Version: ZCRMSDK - VERSION 3.1.0

- Install command - Maven:(in pom.xml)

```
<repositories>
<repository>
<url>https://maven.zohodl.com</url>
</repository>
</repositories>

<dependencies>
<dependency>
<groupId>com.zoho.crm</groupId>
<artifactId>java-sdk</artifactId>
<version>3.1.0</version>
</dependency>
</dependencies>
```

- Install command - Gradle

```
repositories{
maven { url "https://maven.zohodl.com" }
}
dependencies{
implementation 'com.zoho.crm:java-sdk:3.1.0'
}
```

- Notes
  – This version supports External ID.

Previous Version:

1. ZCRMSDK - VERSION 3.0.1

- Install command - Maven:(in pom.xml)

```xml
<repositories>
<repository>
<url>https://maven.zohodl.com</url>
</repository>
</repositories>

<dependencies>
<dependency>
<groupId>com.zoho.crm</groupId>
<artifactId>java-sdk</artifactId>
<version>3.0.1</version>
</dependency>
</dependencies>
```

- Install command - Gradle

```
repositories{
maven { url "https://maven.zohodl.com" }
}
dependencies{
implementation 'com.zoho.crm:java-sdk:3.0.1'
}
```

- Notes
  – This version handles different data types for the autonumber fields for the records operations.

2. ZCRMSDK - VERSION 3.0.0

- Install command - Maven:(in pom.xml)

```xml
<repositories>
<repository>
<url>https://maven.zohodl.com</url>
</repository>
</repositories>

```

```
<dependencies>
<dependency>
<groupId>com.zoho.crm</groupId>
<artifactId>java-sdk</artifactId>
<version>3.0.0</version>
</dependency>
</dependencies>
```

- Install command - Gradle

```
repositories{
maven { url "https://maven.zohodl.com" }
}
dependencies{
implementation 'com.zoho.crm:java-sdk:3.0.0'
}
```

- **Notes**

  – Version 3 is a new major version of the SDK that represents a significant effort to improve the capabilities of the SDK, incorporate customer feedback, upgrade our dependencies, improve performance, and adopt the latest Java standards.

  – The basic usage pattern of the SDK has changed from Version 2 to Version 3. Refer to the samples.

  – 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.