Scala SDK
Version 1.x.x

Zoho CRM

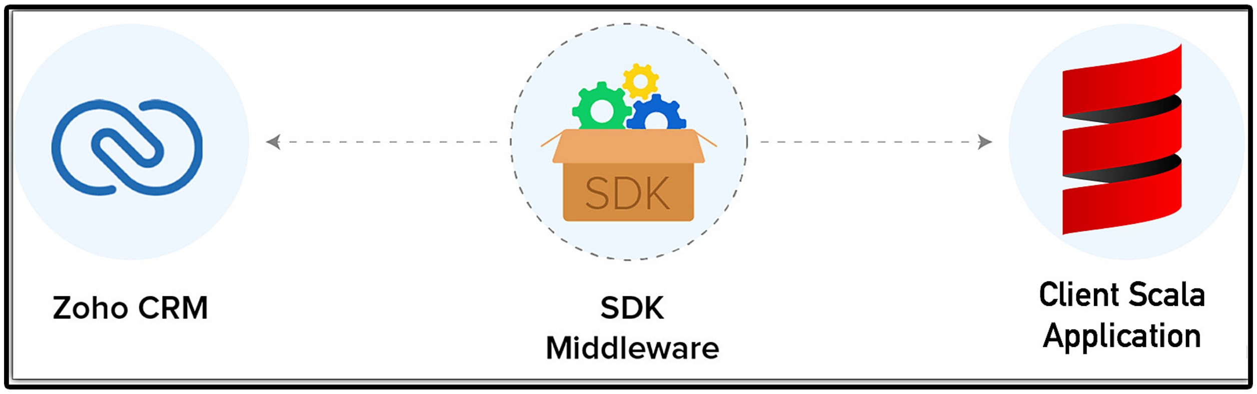
-zoho.com/crm-

Table of Contents

Scala SDK - Overview

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

A sample of how an SDK acts a middle ware or interface between Zoho CRM and a client JS application.



Environmental Setup

Scala SDK requires java (version 8 and above) and scala version 2.13 and above to be set up in your development environment.

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

```
libraryDependencies ++= Seq( "com.zoho.crm" % "scala-sdk" %
"1.1.0")
```

- Build.sbt
- Maven(pom.xml file)

```
        <dependencies>
    <dependency>
        <groupId>com.zoho.crm</groupId>
        <artifactId>scala-sdk</artifactId>
        <version>1.1.0</version>
    </dependency>
</dependencies>
```

- Gradle

```
        dependencies{
    implementation 'com.zoho.crm:scala-sdk:1.1.0'
}
```

- Downloadable JARs

[Download SDK]

The list of dependency JARs that you need are:

- commons-io-1.3.2.jar
- commons-logging-1.2.3.jar
- httpclient-4.5.3.jar
- httpcore-4.4.4.jar
- httpmime-4.5.3.jar
- json-20170516.jar
- mysql-connector-scala-5.1.47-bin.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.

Register your Application

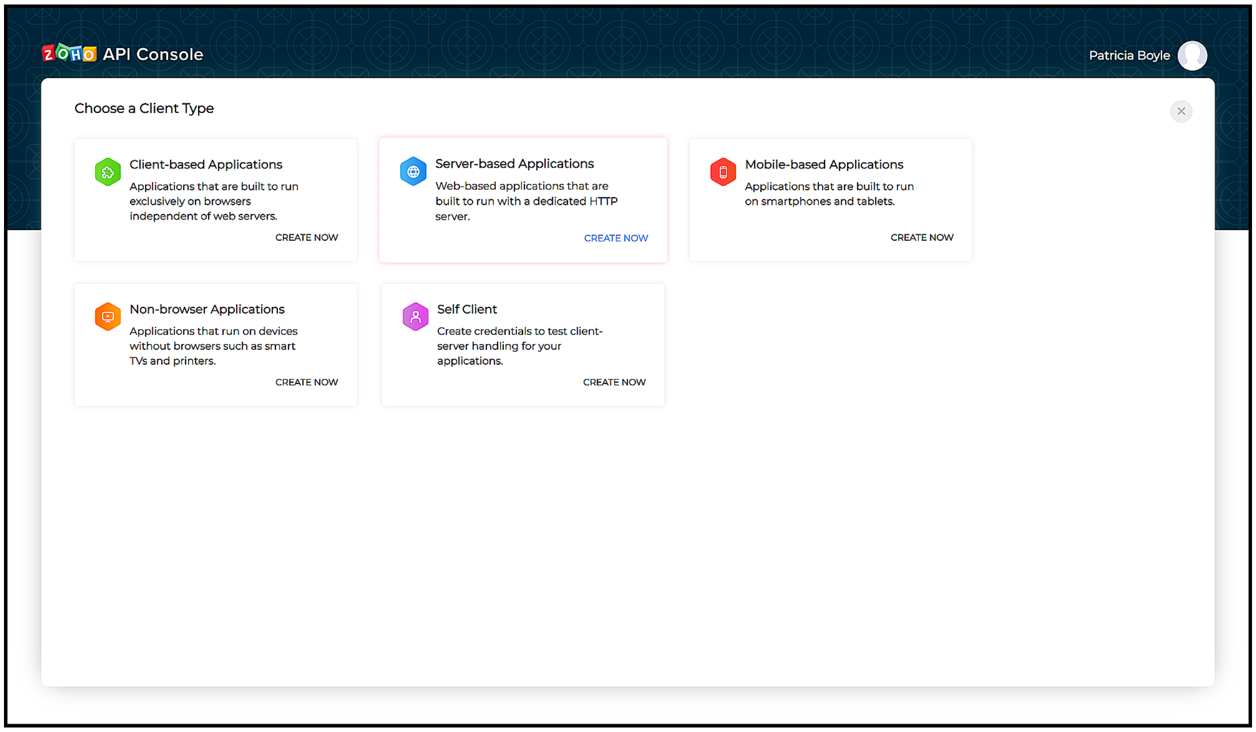
Before you get started with authorization and make any calls using the Zoho CRM APIs, you need to register your application with Zoho CRM.

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.



Client-based Applications
Applications that are built to run exclusively on browsers independent of web servers.

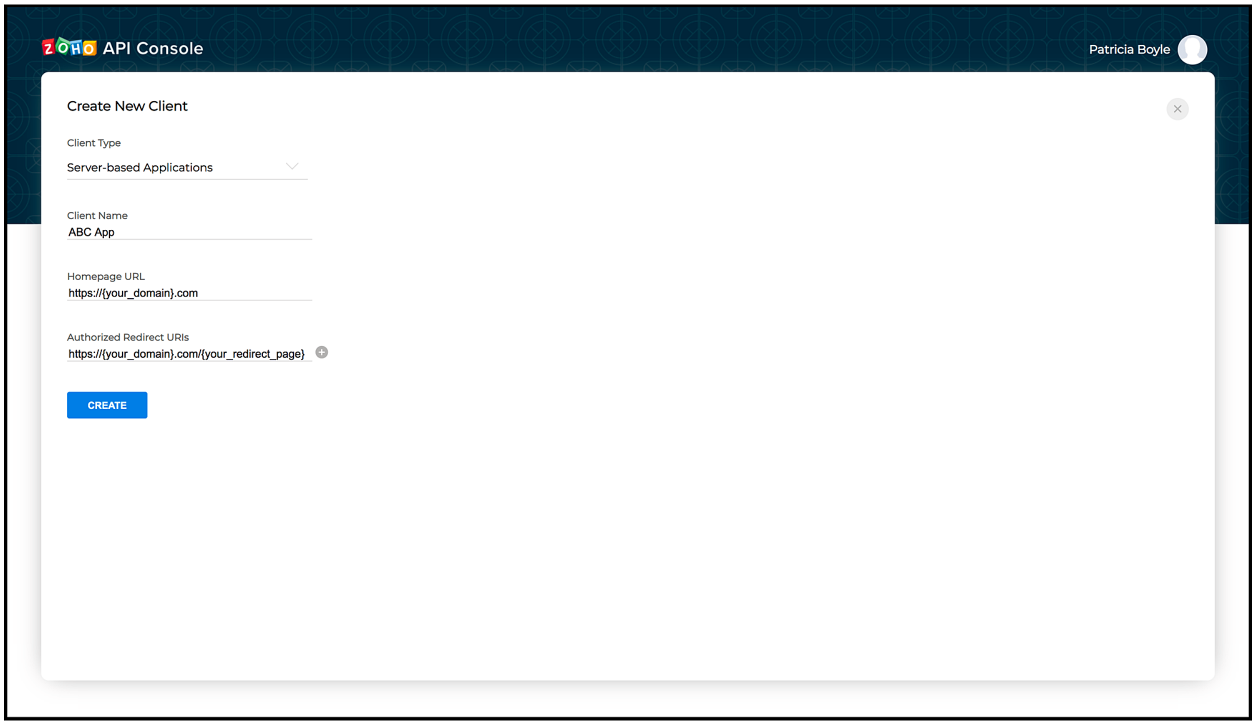
Server-based Applications
Web-based applications that are built to run with a dedicated HTTP server.

Mobile-based Applications
Applications that are built to run on smartphones and tablets.

Non-browser Applications
Applications that run on devices without browsers such as smart TVs and printers.

Self Client
Create credentials to test client-server handling for your applications.

- 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

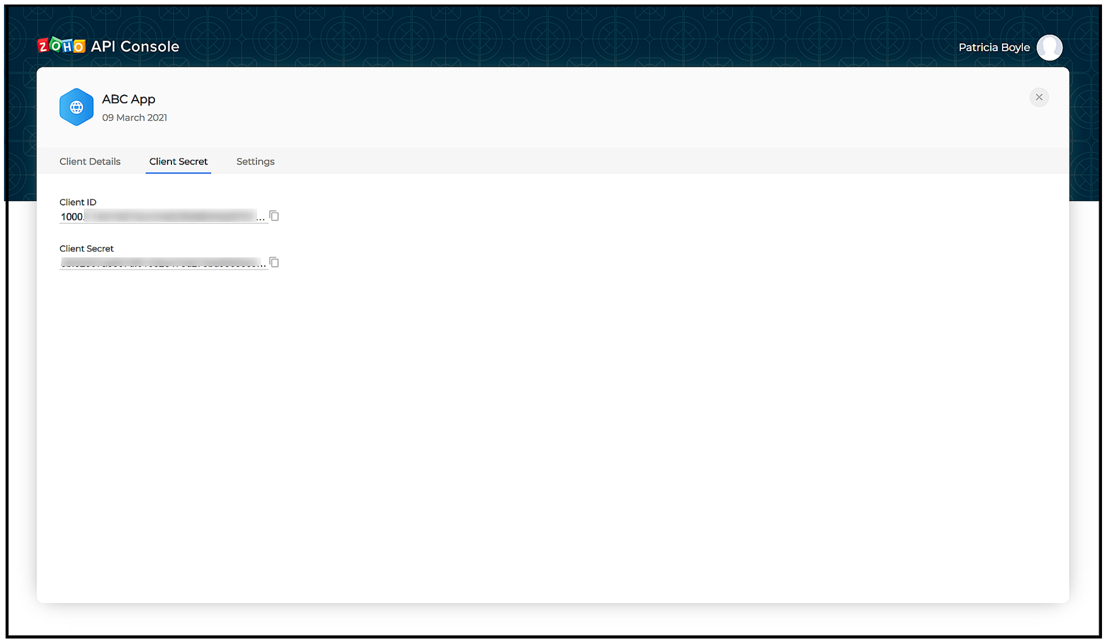
Client Type
Server-based Applications

Client Name
ABC App

Homepage URL
https://{your_domain}.com

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

- 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.



ABC App
09 March 2021

Client ID
1000.

Client Secret

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 Scala 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 **Logger** Class to log exception and API information.

```
import com.zoho.api.logger.Logger
import com.zoho.api.logger.Logger.Levels
/*
    * 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.
*/
var  logger = Logger.getInstance(Logger.Levels.ALL, "/Users/user_name/Documents/scala_sdk_log.log")
```

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

```
import com.zoho.crm.api.UserSignature

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

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

```
/*
```

```
    * Configure the environment
    * which is of the pattern Domain.Environment
    * Available Domains: USDataCenter, EUDataCenter, INDataCenter,
  CNDataCenter, AUDataCenter
    * Available Environments: PRODUCTION, DEVELOPER, SANDBOX
*/
val env = 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.(Optional)
*/
//var token = new OAuthToken("clientId", "clientSecret",
  "REFRESH/GRANT token", TokenType.REFRESH/GRANT)

var token = new OAuthToken("clientId", "clientSecret",
  "REFRESH/GRANT token", TokenType.REFRESH/GRANT,
  Option("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"
*/
//var tokenstore = new DBStore()

  var tokenstore = new DBStore(Option("hostName"),
Option("dataBaseName"), Option("userName"), Option("password"), Option("portNumber"))

//var tokenstore = new FileStore("/Users/user_name/Documents/scala_sdk_token.txt")

//var 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
*
* connectionTimeout
* A Integer field to set connection timeout
*
* requestTimeout
* A Integer field to set request timeout
*
* socketTimeout
* A Integer field to set socket timeout
*/
var sdkConfig = new SDKConfig.Builder().setPickListValidation(false).setAutoRefreshFields(false).connectionTimeout(1000).requestTimeout(1000).socketTimeout(1000).build
```

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

```
var resourcePath = "/Users/user_name/Documents/scalasdk-
  application"
```

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

```
var RequestProxy = new RequestProxy("proxyHost", "proxyPort",
  Option("proxyUser"), Option("password"), Option("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. There are three ways provided by the SDK in which persistence can be applied. They are custom persistence, file persistence, and DB persistence (default).

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(user :UserSignature, token :Token)** - invoked before firing a request to fetch the saved tokens. This method should return an implementation of the Token interface 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.

Database Persistence

In case the user prefers to use the default DataBase persistence, MySQL can be used.

- 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))
    - **access_token**(varchar(255))
    - **grant_token**(varchar(255))
    - **expiry_time**(varchar(20))

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:

```
/*
import com.zoho.api.authenticator.store.DBStore

/*
* 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 interface

var tokenstore = new DBStore(Option("hostName"),
  Option("dataBaseName"), Option("userName"), Option("password"),
    Option("portNumber"))
```

File Persistence

In case of default File Persistence, the user can persist tokens in the local drive, by providing the the absolute file path to the FileStore object.

The file contains:

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

Here is the code to create a FileStore object:

```
/*
import com.zoho.api.authenticator.store.FileStore

//Parameter containing the absolute file path to store tokens
var  tokenstore = new
  FileStore("/Users/user_name/Documents/scala_sdk_token.txt")
```

Custom Persistence

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

Here is the code:

```
using System;
import com.zoho.api.authenticator.Token
import com.zoho.crm.api.exception.SDKException
import com.zoho.crm.api.UserSignature
import com.zoho.api.authenticator.OAuthToken
import scala.collection.mutable.ArrayBuffer
import com.zoho.crm.api.UserSignature
import com.zoho.api.authenticator.store.TokenStore

 class CustomeStore extends TokenStore
 {
    /**
    * This method is used to get user token details.
  *
    * @param user A User class instance.
    * @param token A Token class instance.
    * @return A Token class instance representing the user token details.
    * @throws SDKException SDKException
    */
   override def getToken(user :UserSignature, token :Token) :Token

   /**
    * This method is used to store user token details.
  *
    * @param user A User class instance.
    * @param token A Token class instance.
    * @throws SDKException SDKException
    */
   override def saveToken(user :UserSignature, token :Token)

   /**
    * This method is used to delete user token details.
    * @param token A Token class instance.
    * @throws SDKException SDKException
    */
   override def deleteToken(token :Token)

   /**
    * The method to retrieve all the stored tokens.
    *
    * @throws SDKException if any problem occurs.
    */
   @throws[SDKException]
   override def getTokens: ArrayBuffer[OAuthToken]

   /**
    * The method to delete all the stored tokens.
    *
    * @throws SDKException if any problem occurs.
    */
   @throws[SDKException]
   override def deleteTokens(): Unit
 }
```

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.

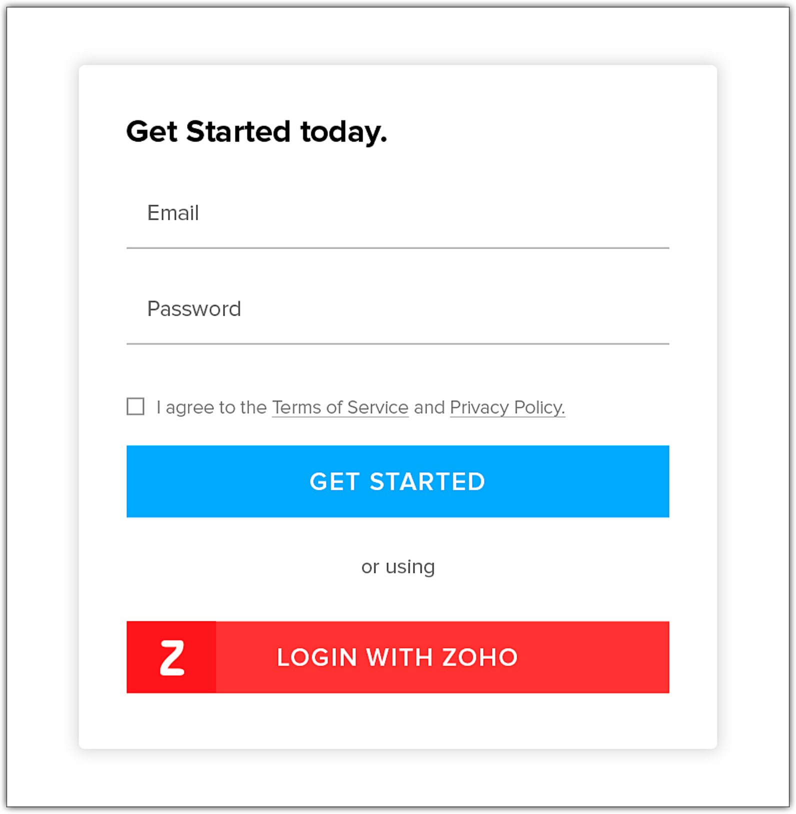
Note

- 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

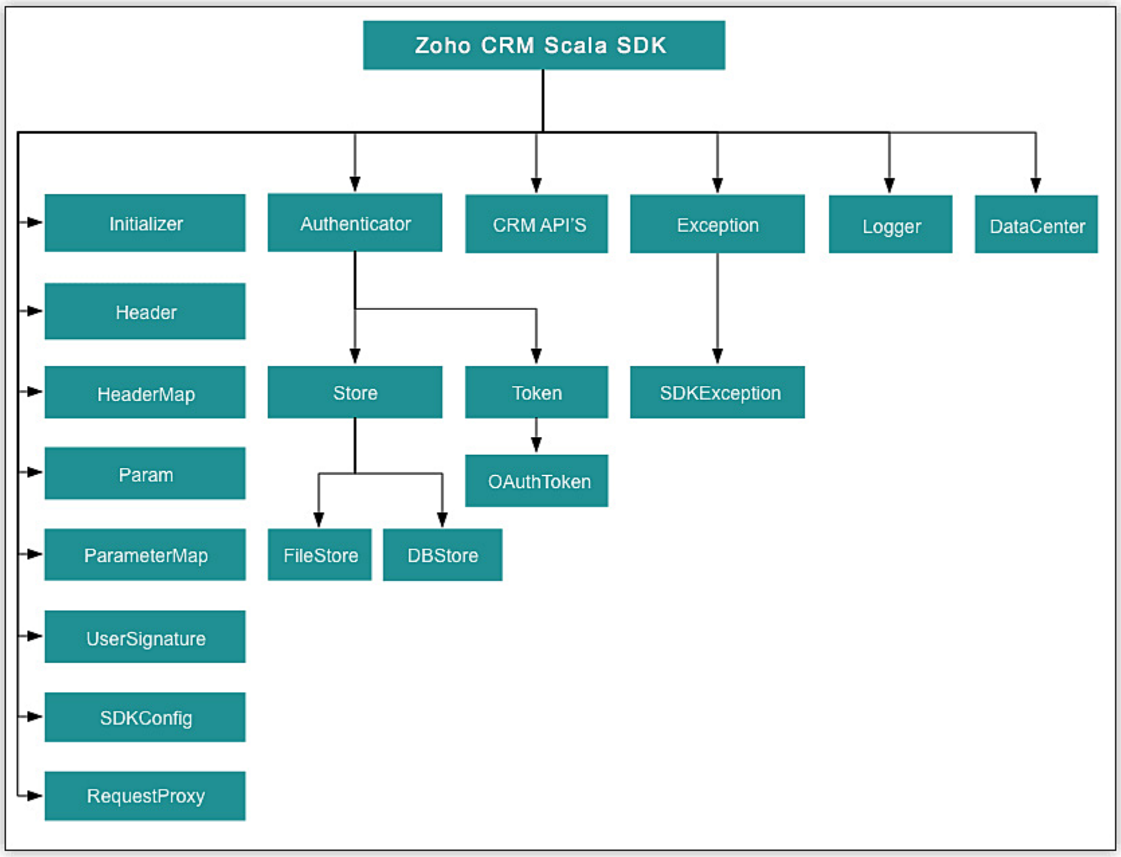
```
import com.zoho.api.authenticator.OAuthToken
import com.zoho.api.authenticator.Token
import com.zoho.api.authenticator.OAuthToken.TokenType
import com.zoho.api.authenticator.store.DBStore
import com.zoho.api.authenticator.store.FileStore
import com.zoho.api.authenticator.store.TokenStore
import com.zoho.crm.api
import com.zoho.crm.api.{Initializer, RequestProxy, SDKConfig,
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


object Initialize {
      @throws[Exception]
      def main(args: Array[String]): Unit = {
        initialize()
      }

      @throws[Exception]
      def initialize(): Unit = {
        /*
        * 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.
```

```
        */
        var logger = Logger.getInstance(Levels.INFO, "/Users/user_name/Documents/scala_sdk_log.log")

        //Create an UserSignature instance that takes user Email as parameter
        var 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
        */
        var 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.
        */
        var token = new OAuthToken("clientId", "clientSecret", "REFRESH/GRANT token", TokenType.REFRESH/GRANT, Option("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()
        var tokenstore = new DBStore(Option("hostName"), Option("dataBaseName"), Option("userName"), Option("password"),
Option("portNumber"))

        //var 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.
          *
          * connectionTimeout
          * A Integer field to set connection timeout
          *
          * requestTimeout
          * A Integer field to set request timeout
          *
          * socketTimeout
          * A Integer field to set socket timeout
         */
        var sdkConfig = new SDKConfig.Builder().setAutoRefreshFields(false).setPickListValidation(true).connectionTimeout(1000).requestTimeout(1000).socketTimeout(1000).build()

        var resourcePath = "/Users/user_name/Documents/scalasdk-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
          */
         var requestProxy = new RequestProxy("proxyHost", "proxyPort", Option ("proxyUser"), Option ("password"), Option ("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, Option(logger), Option(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 Node JS SDK is depicted in the following image.



Sample Codes

All of Zoho CRM's APIs can be used through the Scala 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, String recordId)	Creates an AttachmentsOperations class instance 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.
	APIResponse<ResponseWr	To know the status of the

getBulkReadJobDetails	apper>	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

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<ActionHandl	To update specific contact

	er>	role.
deleteContactRole	APIResponse<ActionHandler>	To delete specific contact 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**
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

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<ActionHandler>	To update the details of the 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.
	APIResponse<ActionHandl	To update only specific

updateNotification	er>	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.
	APIResponse<ActionHandl	To share a record with

shareRecord	er>	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**
	APIResponse<ResponseHa	To get the list of all tags in

getTags	ndler>	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.
getTax	APIResponse<ResponseHandler>	To get the details of a 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<ResponseHa	To get the details of a

	ndler>	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.
	APIResponse<ActionHandl	To update the details of a

updateVariableById	er>	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

Responses and Exceptions

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

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

APIResponse<ResponseHandler> and **APIResponse<ActionHandler>** are the common wrapper objects for Zoho CRM APIs' responses.

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

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

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

- For operations involving records in Tags

-APIResponse<recordactionhandler>

- For getting Record Count for a specific Tag operation

-APIResponse<CountHandler>

- For operations involving BaseCurrency

-APIResponse<BaseCurrencyActionHandler>

- For Lead convert operation

-APIResponse<ConvertActionHandler>

- For retrieving Deleted records operation

-APIResponse<DeletedRecordsHandler>

- For Record image download operation

-APIResponse<DownloadHandler>

- For MassUpdate record operations

-APIResponse<MassUpdateActionHandler>
-APIResponse<MassUpdateResponseHandler>

For GET Requests

The getObject() of the returned APIResponse instance returns the response handler interface.

- The ResponseHandler interface interface encompasses the following
    - ResponseWrapper class (for application/json responses)
    - FileBodyWrapper class (for File download responses)
    - APIException class
- The CountHandler interface interface encompasses the following
    - CountWrapper class (for application/json responses)
    - APIException class
- The DeletedRecordsHandler interface interface encompasses the following
    - DeletedRecordsWrapper class (for application/json responses)
    - APIException class
- The DownloadHandler interface interface encompasses the following
    - FileBodyWrapper class (for File download responses)
    - APIException class
- The MassUpdateResponseHandler interface interface encompasses the following

    - MassUpdateResponseWrapper class (for File download responses)
    - APIException class

For POST, PUT, DELETE Requests

The **getObject()** of the returned APIResponse instance returns the response handler interface.

- The **getObject()** returns an instance of one of the following classes
    - **ActionWrapper**
    - **RecordActionWrapper**
    - **BaseCurrencyActionWrapper**
    - **MassUpdateActionWrapper**
    - **ConvertActionWrapper**
    - **APIException**
- The **ActionHandler** interface encompasses the following
    - **ActionWrapper class** (for File download responses)
    - **APIException class**
- The **ActionResponse** interface encompasses the following
    - **SuccessResponse** class (for application/json responses)
    - **APIException class**
- The **ActionWrapper class** contains **Property/Properties** that may contain one/list of **ActionResponse** interfaces.
- The **ActionHandler** interface encompasses the following
    - **ActionWrapper** class (for application/json responses)
    - **APIException** class
- The **RecordActionHandler** interface encompasses the following
    - **RecordActionWrapper** class (for application/json responses)
    - **APIException** class
- The **BaseCurrencyActionHandler** interface encompasses the following
    - **BaseCurrencyActionWrapper** class (for application/json responses)
    - **APIException** class
- The **MassUpdateActionHandler** interface encompasses the following
    - **MassUpdateActionWrapper** class (for application/json responses)
    - **APIException** class
- The **ConvertActionHandler** interface encompasses the following
    - **ConvertActionWrapper** class (for application/json responses)

- **APIException** class

Threading in the Scala SDK

Threads in a scala program help you achieve parallelism. By using multiple threads, you can make a scala program run faster and do multiple things simultaneously.

The **Scala SDK** 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

```
import com.zoho.crm.api.Initializer
Initializer.switchUser(user, environment, token,  sdkConfig)
Initializer.switchUser(user, environment, token,  sdkConfig,
Option(proxy))
```

```
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, FileStore,
TokenStore}
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.{DataCenter, USDataCenter,EUDataCenter}
import com.zoho.crm.api.exception.SDKException
import com.zoho.api.logger.Logger
import com.zoho.crm.api.record.RecordOperations


object MultiThread {
  @throws[SDKException]
  def main(args: Array[String]): Unit = {
    val logger = Logger.getInstance(Logger.Levels.INFO, "/Users/user_name/Documents/scala-sdk-logs.log")
    val environment1 = USDataCenter.PRODUCTION
    val tokenStore = new FileStore("/Users/user_name/Documents/scala-sdk-tokens.txt")
    val user1 = new UserSignature("user1@zoho.com")
    val token1 = new OAuthToken("clientId1", "clientSecret1", "REFRESH/GRANT token", TokenType.REFRESH / GRANT)
    val resourcePath = "/Users/user_name/Documents/scalasdk-application"
    val sdkConfig = new SDKConfig.Builder().setAutoRefreshFields(false).setPickListValidation(true).build
    Initializer.initialize(user, environment, token, tokenstore, sdkConfig, resourcePath, Option(logger), Option(requestProxy))
    var multiThread = new MultiThread(user1, environment1, token1, "Leads", sdkConfig, null)
    multiThread.start()
    val environment2 = EUDataCenter.PRODUCTION
    val user2 = new UserSignature("user2@zoho.eu")
    val user2Proxy = new RequestProxy("proxyHost", 80, Option("proxyUser"), Option("password"), Option("userDomain"))
    val token2 = new OAuthToken("clientId2", "clientSecret2", "REFRESH/GRANT token", TokenType.REFRESH / GRANT, Option("redirectURL"))
    val sdkConfig2 = new SDKConfig.Builder().setAutoRefreshFields(true).setPickListValidation(false).build
    multiThread = new MultiThread(user2, environment2, token2, "Leads", sdkConfig2,user2Proxy )
    multiThread.start()
  }
}

class MultiThread(var user: UserSignature, var environment: DataCenter.Environment, var token: Token, var moduleAPIName: String, var sdkConfig: SDKConfig, var requestProxy: RequestProxy) extends Thread {
  override def run(): Unit = {
    try {
      Initializer.switchUser(user, environment, token, sdkConfig,
Option(requestProxy))
      println("Getting Records for: " +
Initializer.getInitializer.getUser.getEmail)
      val cro = new RecordOperations
      val getResponse = cro.getRecords(this.moduleAPIName, None,
None)
    } catch {
      case e: Exception =>
        e.printStackTrace()
    }
  }
}
```

- 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.

Multi-threading in a Single-user App

```
import com.zoho.api.authenticator.OAuthToken
import com.zoho.api.authenticator.OAuthToken.TokenType
import com.zoho.api.authenticator.store.FileStore
import com.zoho.crm.api.Initializer
import com.zoho.crm.api.SDKConfig
import com.zoho.crm.api.UserSignature
import com.zoho.crm.api.dc.USDataCenter
import com.zoho.api.logger.Logger
import com.zoho.crm.api.record.RecordOperations

```

```

object MultiThread {
  @throws[Exception]
  def main(args: Array[String]): Unit = {
    val logger = Logger.getInstance(Logger.Levels.INFO,
"/Users/user_name/Documents/scala-sdk-logs.log")
    val environment = USDataCenter.PRODUCTION
    val tokenStore = new
FileStore("/Users/user_name/Documents/scala-sdk-tokens.txt")
    val user = new UserSignature("user1@zoho.com")
    val token = new OAuthToken("clientId1", "clientSecret1",
"REFRESH/GRANT token", TokenType.REFRESH / GRANT)
    val sdkConfig = new
SDKConfig.Builder().setAutoRefreshFields(false).setPickListValidat
ion(true).build
    val resourcePath = "/Users/user_name/Documents/scalasdk-
application"
    Initializer.initialize(user, environment, token, tokenstore,
sdkConfig, resourcePath, Option(logger), Option(requestProxy))
    var mtsu = new MultiThread("Deals")
    mtsu.start()
    mtsu = new MultiThread("Leads")
    mtsu.start()
  }
}

class MultiThread(var moduleAPIName: String) extends Thread {
  override def run(): Unit = {
    try {
      val cro = new RecordOperations
      @SuppressWarnings(Array("rawtypes")) val getResponse =
cro.getRecords(this.moduleAPIName, None, None)
      println(getResponse.get.getObject)
    } catch {
      case e: Exception =>
        e.printStackTrace()
    }
  }
}
```

- The program execution starts from **main()** where the SDK is initialized with the details of user and 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.
- The **MultiThread** object is reinitialized with a different moduleAPIName.
- 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.

SDK Sample code

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

import com.zoho.api.authenticator.Token
import com.zoho.api.authenticator.store.DBStore
import com.zoho.api.authenticator.store.TokenStore
import com.zoho.crm.api.exception.SDKException
import com.zoho.api.logger.Logger
import com.zoho.api.logger.Logger.Levels

import com.zoho.api.authenticator.OAuthToken
import com.zoho.api.authenticator.OAuthToken.TokenType
import com.zoho.crm.api.HeaderMap
import com.zoho.crm.api.Initializer
import com.zoho.crm.api.ParameterMap
import com.zoho.crm.api.SDKConfig
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.record.RecordOperations
import com.zoho.crm.api.record.APIException
import com.zoho.crm.api.record.ResponseHandler
import com.zoho.crm.api.record.ResponseWrapper
import com.zoho.crm.api.tags.Tag
import com.zoho.crm.api.record.RecordOperations.GetRecordsHeader
import com.zoho.crm.api.record.RecordOperations.GetRecordsParam
```

```scala
import com.zoho.crm.api.util.APIResponse
import java.time.OffsetDateTime
import java.time.ZoneOffset
import java.util


object Record {
  @throws[SDKException]
  def main(args: Array[String]): Unit = {
    /*
           * 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.
           */
    val logger = Logger.getInstance(Logger.Levels.INFO, "/Users/user_name/Documents/scala-sdk-logs.log")
    //Create an UserSignature instance that takes user Email as parameter
    val 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
            */
     val 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.
            */
     val token = new OAuthToken("clientId", "clientSecret", 
"REFRESH/GRANT token", TokenType.REFRESH / GRANT)
    /*
             * 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()
    val tokenstore = new DBStore(Option("hostName"),Option(
"dataBaseName"), Option("userName"), Option("password"),
Option("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.
              *
              * connectionTimeout
              * A Integer field to set connection timeout
              *
              * requestTimeout
              * A Integer field to set request timeout
              *
              * socketTimeout
              * A Integer field to set socket timeout
              */
    val sdkConfig = new
```

```
SDKConfig.Builder().setAutoRefreshFields(false).setPickListValidation(true).connectionTimeout(1000).requestTimeout(1000).socketTimeout(1000).build
    val resourcePath = "/Users/user_name/Documents/scalasdk-application"
    /*
            * Call static initialize method of Initializer class that takes the arguments
            * 1 -> UserSignature instance
            * 2 -> Environment instance
            * 3 -> Token instance
            * 4 -> TokenStore instance
            * 5 -> SDKConfig instance
            * 6 -> resourcePath - A String
            * 7 -> Logger instance
            */
    Initializer.initialize(user, environment, token, tokenstore, sdkConfig, resourcePath, Option(logger), Option(requestProxy))
    val moduleAPIName = "Leads"
    val recordOperations = new RecordOperations
    val paramInstance = new ParameterMap
    paramInstance.add(new GetRecordsParam().approved, "both")
    val headerInstance = new HeaderMap
    val enddatetime = OffsetDateTime.of(2020, 5, 20, 10, 0, 1, 0, ZoneOffset.of("+05:30"))
    headerInstance.add(new GetRecordsHeader().IfModifiedSince, enddatetime)
    //Call getRecords method
    val responseOption = recordOperations.getRecords(moduleAPIName, Option(paramInstance), Option(headerInstance))
    if (responseOption.isDefined) {
      val response = responseOption.get
      println("Status Code: " + response.getStatusCode)
      if (util.Arrays.asList(204, 304).contains(response.getStatusCode)) {
        println(if (response.getStatusCode == 204) "No Content"
        else "Not Modified")
        return
      }
      if (response.isExpected) { //Get the object from response
        val responseHandler = response.getObject
        responseHandler match {
          case responseWrapper : ResponseWrapper =>
          //Get the obtained Record instances
          val records = responseWrapper.getData()

          for (record <- records) {
            println("Record ID: " + record.getId)
            var createdByOption = record.getCreatedBy()
            if (createdByOption.isDefined) {
              var createdBy= createdByOption.get
              println("Record Created By User-ID: " + createdBy.getId)
              println("Record Created By User-Name: " + createdBy.getName)
              println("Record Created By User-Email: " + createdBy.getEmail)
            }
            println("Record CreatedTime: " + record.getCreatedTime)
            var modifiedByOption = record.getModifiedBy()
            if (modifiedByOption.isDefined) {
              var modifiedBy = modifiedByOption.get
              println("Record Modified By User-ID: " + modifiedBy.getId)
              println("Record Modified By User-Name: " + modifiedBy.getName)
              println("Record Modified By User-Email: " + modifiedBy.getEmail)
            }
            println("Record ModifiedTime: " + record.getModifiedTime)
            val tags = record.getTag()
            if (tags.nonEmpty) {

              for (tag <- tags) {
                println("Record Tag Name: " + tag.getName)
                println("Record Tag ID: " + tag.getId)
              }
            }
            println("Record Field Value: " + record.getKeyValue("Last_Name"))
            println("Record KeyValues: ")

          }
          //Get the Object obtained Info instance
          val infoOption = responseWrapper.getInfo
          //Check if info is not null
          if (infoOption.isDefined) {
            var info = infoOption.get
            if (info.getPerPage().isDefined) { //Get the PerPage of the Info
              println("Record Info PerPage: " + info.getPerPage.toString)
            }
            if (info.getCount.isDefined) { //Get the Count of the Info
              println("Record Info Count: " + info.getCount.toString)
            }
            if (info.getPage.isDefined) { //Get the Page of the Info
              println("Record Info Page: " + info.getPage().toString)
            }
            if (info.getMoreRecords().isDefined) { //Get the MoreRecords of the Info
              println("Record Info MoreRecords: " + info.getMoreRecords().toString)
            }
          }
        case exception : APIException =>
          println("Status: " + exception.getStatus().getValue)
          println("Code: " + exception.getCode().getValue)
          println("Details: ")

          exception.getDetails().foreach(entry=>{
            println(entry._1 + ": " + entry._2)
          })
```

```scala
          println("Message: " + exception.getMessage().getValue)
        case _ =>
}
      }
      else {
        val responseObject = response.getModel
        val clas = responseObject.getClass
        val fields = clas.getDeclaredFields
        for (field <- fields) {
          println(field.getName + ":" + field.get(responseObject))
        }
      }
    }
  }
}

class Record {}
```

Release Notes

Current Version

1. ZCRMSDK -VERSION 1.1.0

Install command

```
libraryDependencies ++= Seq( "com.zoho.crm" % "scala-sdk" % "1.1.0")
```

Enhancement

- Supported External ID.
- Fixed None redirect url bug.

Previous Versions

1. ZCRMSDK -VERSION 1.0.0

Install command

```
libraryDependencies ++= Seq( "com.zoho.crm" % "scala-sdk" % "1.0.0")
```

Enhancement

- Improve the capabilities of the SDK
- Incorporate customer feedback
- Upgrade our dependencies
- Improve performance
- 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.