C# SDK Version 3

Zoho CRM

--zoho.com/crm--

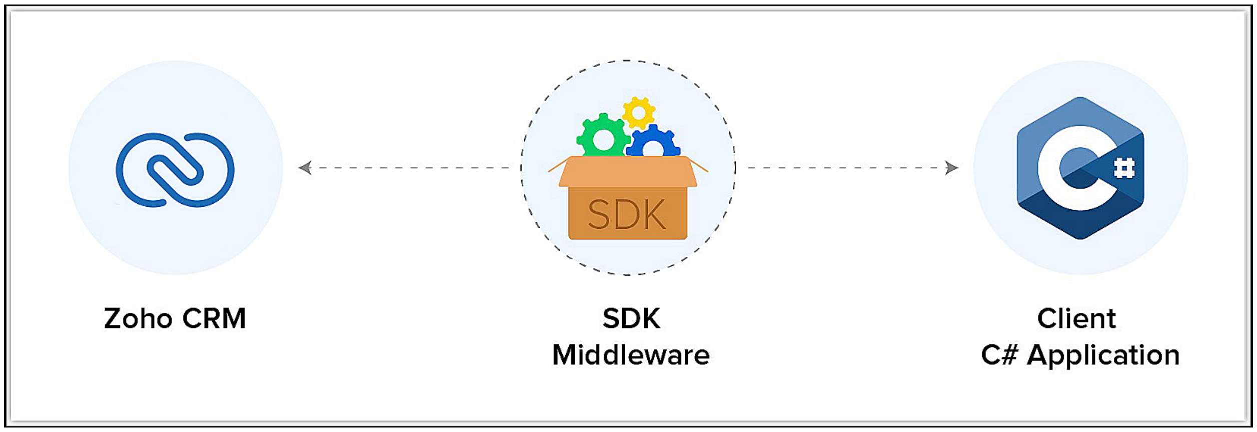
Table of Contents

Overview

C# SDK offers a way to create client C# applications that can be integrated with Zoho CRM. This SDK makes the access and use of necessary CRM APIs with ease. 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 interface implementations, instances or objects. Authentication to access Zoho CRM APIs is through OAuth2.0 authentication mechanism. Invariably, HTTP requests and responses are taken care of by the SDK.

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



C# SDK allows you to

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

Environmental Setup

C# SDK requires .NET Framework 4.6.1(or above) or .Net Core 2.X(or above) to be set up in your development environment. The compatibility warning can be ignored

> Note
> The warning occurs because we have supported .NET Framework 4.x.x to make the C# SDK compatible with older versions of Visual Studio (For instance, Visual Studio 2015). We assure you that all the functionalities in the SDK works to the utmost perfection.

Including the SDK in your project

You can include the SDK to your project using:

Install Visual Studio IDE from Visual Studio (if not installed).

C# SDK is available as a Nuget package. The SDK requires the following from the client app:

The ZCRMSDK assembly can be installed through the Nuget Package Manager and through the following options:

- Packet Manager

```
Install-Package ZCRMSDK -Version 3.0.0
Install-Package MySql.Data -Version 6.9.12
Install-Package Newtonsoft.Json -Version 11.0.1
```

- .NET CLI

```
dotnet add package ZCRMSDK --version 3.0.0
dotnet add package Newtonsoft.Json --version 11.0.1
dotnet add package MySql.Data --version 6.9.12
```

- Package Reference

For projects that support PackageReference, copy this XML node into the project file to refer the package.

```
<ItemGroup>
<PackageReference Include="ZCRMSDK" Version="3.0.0" />
<PackageReference Include="Newtonsoft.Json" Version="11.0.1" />
<PackageReference Include="MySql.Data" Version="6.9.12" />
</ItemGroup>
```

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.

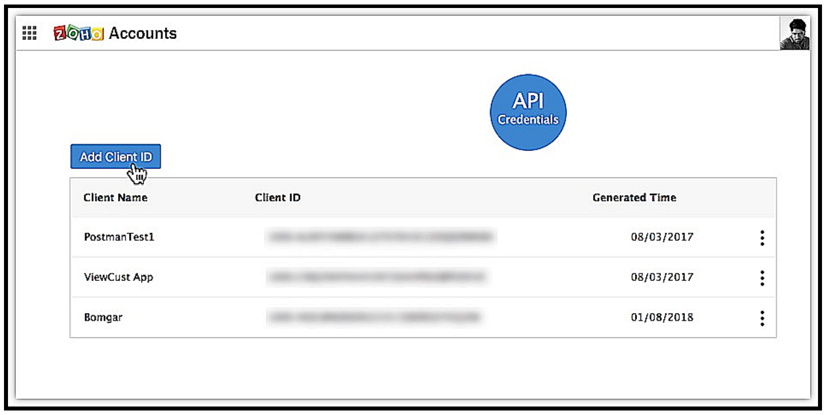
Register your application

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 the site: **https://api-console.zoho.com**

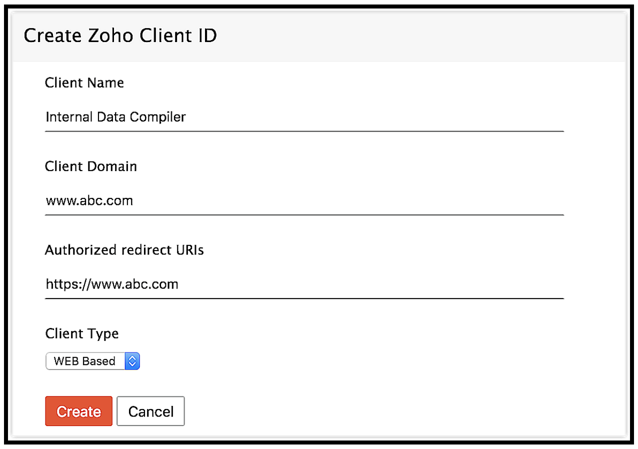
2. Click **Add Client ID**.



Client Name	Client ID	Generated Time
PostmanTest1	[redacted]	08/03/2017
ViewCust App	[redacted]	08/03/2017
Bomgar	[redacted]	01/08/2018

3. Enter the **Client Name**, **Client Domain** and **Authorized Redirect URL**.

4. Select the **Client Type** as **Web based**



Client Name: Internal Data Compiler

Client Domain: www.abc.com

Authorized redirect URIs: https://www.abc.com

Client Type: WEB Based

5. Click **Create**.

6. Your Client app would have been created and displayed by now.

7. The newly registered app's Client ID and Client Secret can be found by clicking **Options → Edit.**

Note:
Options is the three dot icon at the right corner.

Registered applications 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.
- Redirect URI – The Callback URL that you registered during the app registration.

Configuration

Before you get started with creating your C# 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.

```
/*
   * 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(Logger.Levels.ALL, "/Users/Documents/csharp_sdk_log.log");
```

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

```
//Create a 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.

```
/*
   * 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 config = 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/csharpsdk-application";
```

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. There are three ways provided by the SDK in which persistence can be applied. They are file persistence, DB persistence (default) and Custom persistence.

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 an implementation of the Token interface object for the library to process it.
- **SaveToken(UserSignature user, Token token)** - invoked after fetching access and refresh tokens from Zoho.
- **DeleteToken(UserSignature user, 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:

- Database Persistence
- File Persistence
- Custom Persistence

Database Persistence

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

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

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

MySQL Query

```
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");
```

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:

```
//The parameter containing the absolute file path to store tokens
TokenStore tokenstore = new
```

```
    FileStore("/Users/username/Documents/csharp_sdk_token.txt");
```

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:

```
using System;
using Com.Zoho.API.Authenticator;
using Com.Zoho.API.Authenticator.Store;
using Com.Zoho.Crm.API;
namespace user.store
{
    public class CustomStore : TokenStore
    {
        public CustomStore()
        {
        }
        ///

        /// A UserSignature class instance.
        /// A Token (Com.Zoho.API.Authenticator.OAuthToken) class instance.
        /// A Token class instance representing the user token details.
        public Token GetToken(UserSignature user, Token token)
        {
            // Add code to get the token
            return null;
        }

        ///

        /// A UserSignature class instance.
        /// A Token (Com.Zoho.API.Authenticator.OAuthToken) class instance.
        public void SaveToken(UserSignature user, Token token)
        {
            // Add code to save the token
        }

        ///

        /// A UserSignature class instance.
        /// A Token (Com.Zoho.API.Authenticator.OAuthToken) class
instance.
        public void DeleteToken(UserSignature user, Token token)
        {
            // Add code to delete the token
        }

        public void GetTokens()
        {
            // Add code to get the all stored tokens
        }

        public void DeleteTokens()
        {
            // Add code to delete the all stored token
        }
    }
}
```

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.

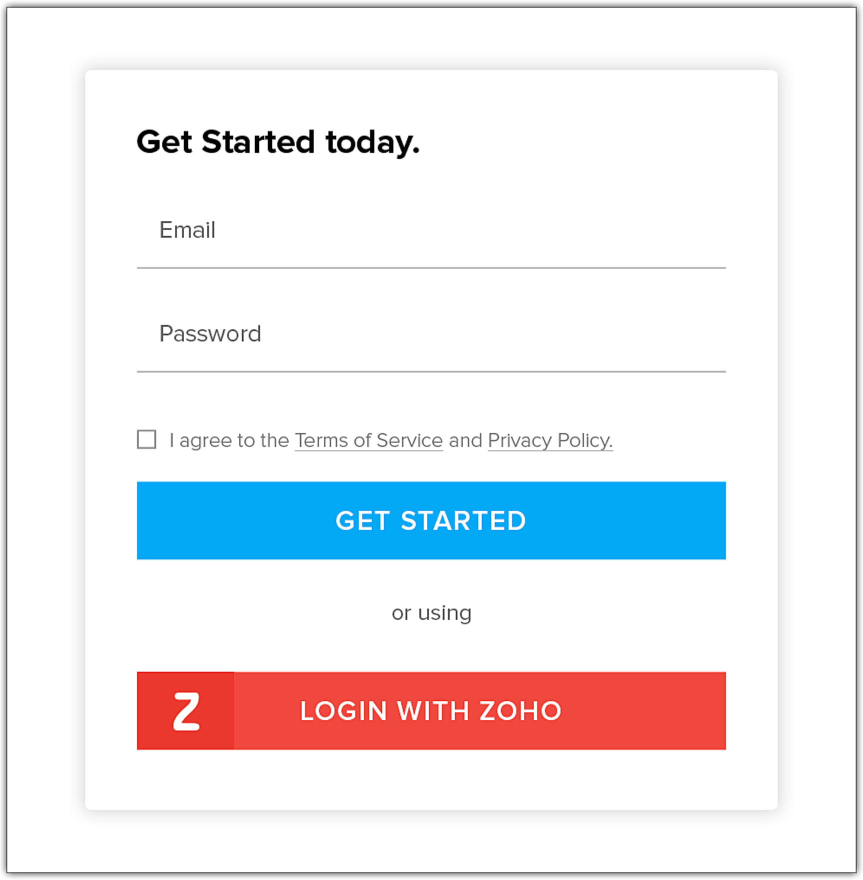
1. Login to your Zoho account.
2. Visit https://api-console.zoho.com
3. Click **Self Client** option of the client for which you wish to authorize.

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.
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.ALL, 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.

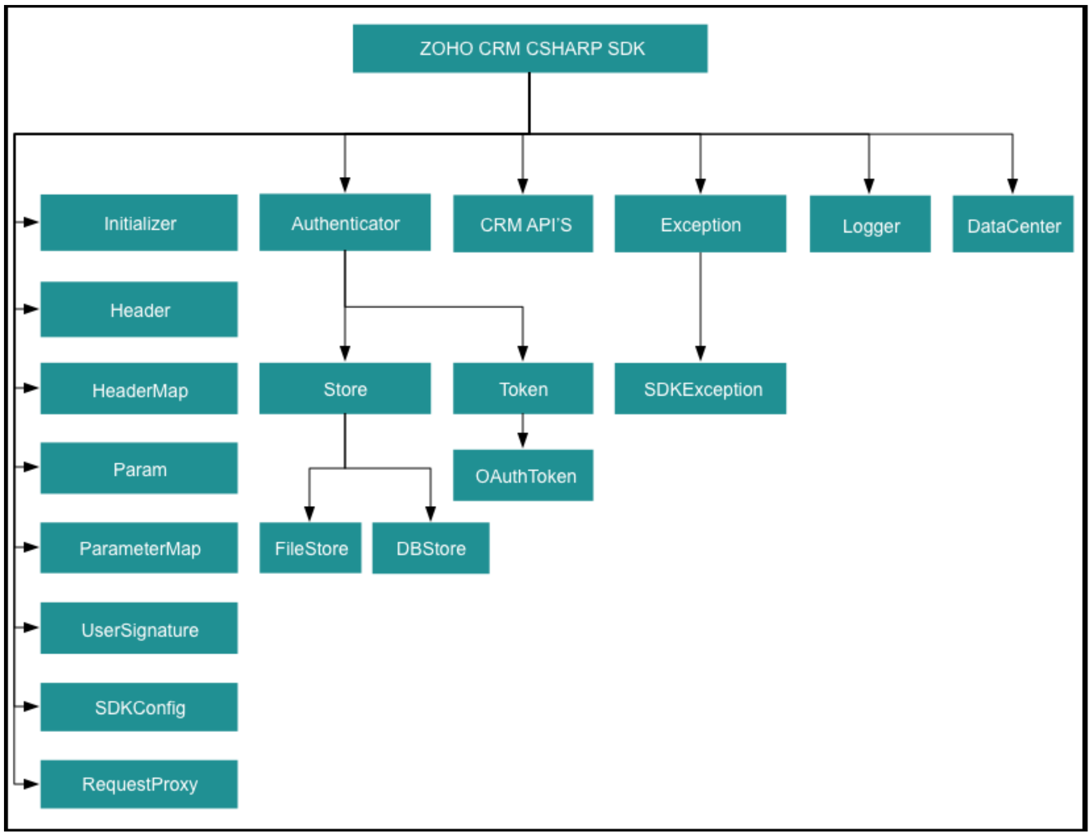
Initialize the SDK using the following code.

```csharp
using System;
using Com.Zoho.API.Authenticator;
using Com.Zoho.API.Authenticator.Store;
using Com.Zoho.Crm.API;
using Com.Zoho.Crm.API.Dc;
using Com.Zoho.Crm.API.Logger;
using static Com.Zoho.API.Authenticator.OAuthToken;
using Environment = Com.Zoho.Crm.API.Dc.DataCenter.Environment;
using SDKInitializer = Com.Zoho.Crm.API.Initializer;

namespace Com.Zoho.Crm.Sample.Initializer
{
    public class Initialize
    {
        public static void SDKInitialize()
        {
            /*
            * 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(Logger.Levels.ALL, "/Users/Documents/csharp_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/csharpsdk-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

            SDKInitializer.Initialize(user, environment, token, tokenstore, sdkConfig, resourcePath);

            SDKInitializer.Initialize(user, environment, token, tokenstore, sdkConfig, resourcePath, logger);

            SDKInitializer.Initialize(user, environment, token, tokenstore, sdkConfig, resourcePath, requestProxy);

            SDKInitializer.Initialize(user, environment, token, tokenstore, sdkConfig, resourcePath, logger, requestProxy);
        }
    }
```

```
107}
```

Class Hierarchy

All Zoho CRM entities are modelled as classes having members and methods applicable to that particular entity.
The class hierarchy of various Zoho CRM entities in the C# 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 **Object** property 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 response will be an instance of **APIException** class.

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

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

- 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 **Object** property of the returned APIResponse instance 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

- The **Object** property of the returned APIResponse instance returns the response handler 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 C# 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	Description
GetAttachments	To fetch the list of attachments of a record.

UploadAttachments	To upload attachments to a record.
DeleteAttachments	To delete the attachments that were added to a record.
DeleteAttachment	To delete an attachment that was added to a record.
DownloadAttachment	To download an attachment that was uploaded to a record.
UploadLinkAttachments	To upload a link as an attachment to a record.

Blueprint Operations

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

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

Bulk Read Operations

Method	Description
CreateBulkReadJob	To schedule a bulk read job to export records that match the criteria.

GetBulkReadJobDetails	To know the status of the bulk read job scheduled previously.
DownloadResult	To download the result of the bulk read job. The response contains a zip file. Extract it to get the CSV or ICS file depending on the "file_type" you specified while creating the bulk read job

Bulk Write Operations

Method	Description
UploadFile	To upload a CSV file in ZIP format. The response contains the "file_id". Use this ID while making the bulk write request.
CreateBulkWriteJob	To create a bulk write job to insert, update, or upsert records. The response contains the "job_id". Use this ID while getting the status of the scheduled bulk write job.
GetBulkWriteJobDetails	To know the status of the bulk write job scheduled previously.
DownloadResult	To download the result of the bulk write job. The response contains a zip file. Extract it to get the CSV or ICS file depending on the "file_type" you specified while creating the write job

Contact Roles Operations

Method	Description
GetContactRoles	To get the list of all contact roles.

CreateContactRoles	To create contact roles.
UpdateContactRoles	To update contact roles.
DeleteContactRoles	To delete contact roles.
GetContactRole	To get specific contact role.
UpdateContactRole	To update specific contact role.
DeleteContactRole	To delete specific contact role.

Currencies Operations

Method	Description
GetCurrencies	To get the list of all currencies available for your org.
AddCurrencies	To add new currencies to your org.
UpdateCurrencies	To update the currencies' details of your org.
EnableMultipleCurrencies	To enable multiple currencies for your org.
UpdateBaseCurrency	To update the base currency details of your org.
GetCurrency	To get the details of specific currency.
UpdateCurrency	To update the details of specific currency.

Custom View Operations

Constructor	Description

CustomViewsOperations(string module)	Creates a CustomViewsOperations class instance with the moduleAPIName.

Method	Description
GetCustomViews	To get the list of all custom views in a module.
GetCustomView	To get the details of specific custom view in a module.

Fields Metadata Operations

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

Method	Description
GetFields	To get the meta details of all fields in a module.
GetField	To get the meta details of specific field in a module.

Files Operations

Method	Description
UploadFiles	To upload files and get their encrypted IDs.

GetFile	To get the uploaded file through its encrypted ID.

Layouts Operations

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

Method	Description
GetLayouts	To get the details of all the layouts in a module.
GetLayout	To get the details (metadata) of a specific layout in a module.

Modules Operations

Method	Description
GetModules	To get the details of all the modules.
GetModule	To get the details (metadata) of a specific module.
UpdateModuleByAPIName	To update the details of a module by its module API name.
UpdateModuleById	To update the details of a module by its ID.

Notes Operations

Method	Description

GetNotes	To get the list of notes of a record.
CreateNotes	To add new notes to a record.
UpdateNotes	To update the details of the notes of a record.
DeleteNotes	To delete the notes of a record.
GetNote	To get the details of a specific note.
UpdateNote	To update the details of an existing note.
DeleteNote	To delete a specific note.

Notification Operations

Method	Description
EnableNotifications	To enable instant notifications of actions performed on a module.
GetNotificationDetails	To get the details of the notifications enabled by the user.
UpdateNotifications	To update the details of the notifications enabled by a user. All the provided details would be persisted and rest of the details would be removed.
UpdateNotification	To update only specific details of a specific notification enabled by the user. All the provided details would be persisted and rest of the details will not be removed.

DisableNotifications	To stop all the instant notifications enabled by the user for a channel.
DisableNotification	To disable notifications for the specified events in a channel.

Organization Operations

Method	Description
GetOrganization	To get the details of your organization.
UploadOrganizationPhoto	To upload a photo of your organization.

Profile Operations

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

Method	Description
GetProfiles	To get the list of profiles available for your organization.
GetProfile	To get the details of a specific profile.

Query (COQL) Operation

Method	Description

getRecords	To get the records from a module through a COQL query.

Records Operations

Method	Description
GetRecord	To get a specific record from a module.
UpdateRecord	To update a specific record in a module.
DeleteRecord	To delete a specific record from a module.
GetRecords	To get all records from a module.
CreateRecords	To insert records in a module.
UpdateRecords	To update records in a module.
DeleteRecords	To delete records from a module.
UpsertRecords	To insert/update records in a module.
GetDeletedRecords	To get the deleted records from a module.
SearchRecords	To search for records in a module that match certain criteria, email, phone number, or a word.
ConvertLead	To convert records(Leads to Contacts/Deals).
GetPhoto	To get the photo of a record.
UploadPhoto	To upload a photo to a record.
DeletePhoto	To delete the photo of a record.
MassUpdateRecords	To update the same field for multiple records in

	a module.
GetMassUpdateStatus	To get the status of the mass update job scheduled previously.

Related List Operations

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

Method	Description
GetRelatedLists	To get the details of all the related lists of a module.
GetRelatedList	To get the details of a specific related list of a module.

Related Records Operations

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

Method	Description
GetRelatedRecords	To get list of records from the related list of a module.
UpdateRelatedRecords	To update the association/relation between the

	records.
DelinkRecords	To delete the association between the records.
GetRelatedRecord	To get the records from a specific related list of a module.
UpdateRelatedRecord	To update the details of a specific record of a related list in a module.
Delink Record	To delete a specific record from the related list of a module.

Role Operations

Method	Description
GetRoles	To get the list of all roles available in your organization.
GetRole	To get the details of a specific role.

Shared Records Operations

Constructor	Description
ShareRecordsOperations(long recordId, string moduleAPIName)	Creates a ShareRecordsOperations class instance with the recordId and moduleAPIName.

Method	Description
GetSharedRecordDetails	To get the details of a record shared with other users.
ShareRecord	To share a record with other users in the organization.
UpdateSharePermissions	To ● Update the sharing permissions of a record granted to users as Read-Write, Read-only, or grant full access. ● Revoke access given to users to a shared record. ● Update the access permission to the related lists of the record that was shared with the user.
RevokeSharedRecord	To revoke access to a shared record.

Tags Operations

Method	Description
GetTags	To get the list of all tags in your organization.
CreateTags	To create tags.
UpdateTags	To update multiple tags.
UpdateTag	To update a specific tag.
DeleteTag	To delete a specific tag from the module.
MergeTags	To merge two tags.

AddTagsToRecord	To add tags to a specific record.
RemoveTagsFromRecord	To remove tags from a record.
AddTagsToMultipleRecords	To add tags to multiple records.
RemoveTagsFromMultipleRecords	To remove tags from multiple records.
GetRecordCountForTag	To get the record count for a tag.

Taxes Operations

Method	Description
GetTaxes	To get the taxes of your organization.
CreateTaxes	To add taxes to your organization.
UpdateTaxes	To update the existing taxes of your organization.
DeleteTaxes	To delete multiple taxes from your organization.
GetTax	To get the details of a specific tax.
DeleteTax	To delete a specific tax from your organization.

Territory Operations

Method	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	Description
GetUsers	To get the list of users in your organization.
CreateUser	To add a user to your organization.
UpdateUsers	To update the existing users of your organization.
GetUser	To get the details of a specific user.
UpdateUser	To update the details of a specific user.
DeleteUser	To delete a specific user from your organization.

Variable Groups Operations

Method	Description
GetVariableGroups	To get the list of all variable groups available for your organization.
GetVariableGroupById	To get the details of a variable group by its group ID.
GetVariableGroupByAPIName	To get the details of a specific variable group by its API name.

Variables Operations

Method	Description
GetVariables	To get the list of variables available for your organization.

CreateVariables	To add new variables to your organization.
UpdateVariables	To update the details of variables.
DeleteVariables	To delete multiple variables.
GetVariableById	To get the details of a specific variable by its unique ID.
UpdateVariableById	To update the details of a specific variable by its unique ID.
DeleteVariable	To delete a specific variable.
GetVariableForAPIName	To get the details of a variable by its API name.
UpdateVariableByAPIName	To update the details of a variable by its API name.

Threading in the C# SDK

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

The C# 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

Multi-threading for multi-users is achieved using Initializer's static SwitchUser().

```
Initializer.SwitchUser(UserSignature user, Environment environment,
Token token, SDKConfig sdkConfig);
```

```
Initializer.SwitchUser(UserSignature user, Environment environment,
Token token, SDKConfig sdkConfig, RequestProxy proxy);
```

- 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.

```
usingSystem;
using System.Collections.Generic;
using System.Threading;
using Com.Zoho.API.Authenticator;
using Com.Zoho.API.Authenticator.Store;
using Com.Zoho.Crm.API;
using Com.Zoho.Crm.API.Dc;
using Com.Zoho.Crm.API.Logger;
using Com.Zoho.Crm.API.Record;
using Com.Zoho.Crm.API.Util;
using Newtonsoft.Json;
using static Com.Zoho.API.Authenticator.OAuthToken;

namespace csharpsdksampleapplication
{
    public class MultiThread
    {
        DataCenter.Environment environment;

        UserSignature user;

        Token token;

        string moduleAPIName;

        public MultiThread(UserSignature user,
DataCenter.Environment environment, Token token, string
moduleAPIName)
        {
            this.environment = environment;

            this.user = user;

            this.token = token;

            this.moduleAPIName = moduleAPIName;
        }

        static void Main(string[] args)
        {
            Logger logger = Logger.GetInstance(Logger.Levels.ALL,
"/Users/user_name/Documents/csharp_sdk_log.log");

            DataCenter.Environment environment1 =
USDataCenter.PRODUCTION;

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

            TokenStore tokenstore = new
FileStore("/Users/user_name/Documents/csharp_sdk_token.txt");

            Token token1 = new OAuthToken("clientId1",
"clientSecret1", "REFRESH/GRANT token", TokenType.REFRESH,
"redirectURL1");

            string resourcePath =
"/Users/user_name/Documents/csharpsdk-application";

            SDKConfig config = new
SDKConfig.Builder().SetAutoRefreshFields(true).Build();

            DataCenter.Environment environment2 =
EUDataCenter.PRODUCTION;

            UserSignature user2 = new
UserSignature("user2@zoho.eu");
```

```
            Token token2 = new OAuthToken("clientId2",
"clientSecret2", "REFRESH/GRANT token", TokenType.REFRESH,
"redirectURL2");

            Initializer.Initialize(user1, environment1, token1,
tokenstore, config, resourcePath, logger);

            MultiThread multiThread1 = new MultiThread(user1,
environment1, token1, "Vendors");

            Thread thread1 = new Thread(() =>
multiThread1.GetRecords());

            thread1.Start();

            MultiThread multiThread2 = new MultiThread(user2,
environment2, token2, "Quotes");

            Thread thread2 = new Thread(() =>
multiThread2.GetRecords());

            thread2.Start();

            thread1.Join();

            thread2.Join();
        }

        public void GetRecords()
        {
            try
            {
                SDKConfig config = new
SDKConfig.Builder().SetAutoRefreshFields(true).Build();

                Initializer.SwitchUser(this.user,
this.environment, this.token, config);

                Console.WriteLine("Fetching records for user - "
+ Initializer.GetInitializer().User.Email);
```

```
                RecordOperations recordOperation = new
RecordOperations();

                APIResponse response =
recordOperation.GetRecords(this.moduleAPIName, null, null);

                if (response != null)
                {
                 Get the status code from response
                    Console.WriteLine("Status Code: " +
response.StatusCode);

                    if (new List() { 204, 304
}.Contains(response.StatusCode))
                    {
                        Console.WriteLine(response.StatusCode ==
204 ? "No Content" : "Not Modified");

                        return;
                    }


                    if (response.IsExpected)
                    {
                        //Get object from response
                        ResponseHandler responseHandler =
response.Object;

                        if (responseHandler is ResponseWrapper)
                        {
                            //Get the received ResponseWrapper
instance
                            ResponseWrapper responseWrapper =
(ResponseWrapper)responseHandler;

                            //Get the list of obtained Record
instances
                            List records = responseWrapper.Data;

                            foreach (Record record in records)
                            {
Console.WriteLine(JsonConvert.SerializeObject(record));
                            }
                        }
                        //Check if the request returned an
exception
                        else if (responseHandler is APIException)
                        {
                            //Get the received APIException
instance
                            APIException exception =
(APIException)responseHandler;

                            //Get the Status
                            Console.WriteLine("Status: " +
exception.Status.Value);

                            //Get the Code
                            Console.WriteLine("Code: " +
exception.Code.Value);

                            Console.WriteLine("Details: ");

                            //Get the details map
                            foreach (KeyValuePair entry in
exception.Details)
                            {
                                //Get each value in the map
                                Console.WriteLine(entry.Key + ":
" + JsonConvert.SerializeObject(entry.Value));
                            }

                            //Get the Message
                            Console.WriteLine("Message: " +
exception.Message.Value);
                        }
                    }
                }
            }
            catch (System.Exception ex)
```

```
                {

    Console.WriteLine(JsonConvert.SerializeObject(ex));
                }
            }
        }
    }
    ,>
```

Multi-threading in a Single-user App

- 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 the Start() is called which in-turn invokes the run(), the moduleAPIName is switched through the method parameter. Therefore, this creates a thread for the particular method called with the MultiThread instance.

```
using System;
using System.Collections.Generic;
using System.Threading;
using Com.Zoho.API.Authenticator;
using Com.Zoho.API.Authenticator.Store;
using Com.Zoho.Crm.API;
using Com.Zoho.Crm.API.Dc;
using Com.Zoho.Crm.API.Logger;
using Com.Zoho.Crm.API.Record;
using Com.Zoho.Crm.API.Util;
using Newtonsoft.Json;
using static Com.Zoho.API.Authenticator.OAuthToken;

namespace csharpsdksampleapplication
{
    public class MultiThread
    {
        static void Main(string[] args)
        {
            Logger logger = Logger.GetInstance(Logger.Levels.ALL,
"/Users/user_name/Documents/csharp_sdk_log.log");

            DataCenter.Environment env = USDataCenter.PRODUCTION;

            UserSignature user = new
```

```
UserSignature("user1@zoho.com");

            TokenStore tokenstore = new
FileStore("/Users/user_name/Documents/csharp_sdk_token.txt");

            Token token = new OAuthToken("clientId",
"clientSecret", "REFRESH/GRANT token", TokenType.REFRESH);

            string resourcePath =
"/Users/user_name/Documents/csharpsdk-application";

            SDKConfig config = new
SDKConfig.Builder().SetAutoRefreshFields(true).Build();

            Initializer.Initialize(user, env, token, tokenstore,
config, resourcePath, logger);

            MultiThread multiThread1 = new MultiThread();

            Thread thread1 = new Thread(() =>
multiThread1.GetRecords("Quotes"));

            thread1.Start();

            Thread thread2 = new Thread(() =>
multiThread1.GetRecords("Leads"));

            thread2.Start();

            thread1.Join();

            thread2.Join();
        }

        public void GetRecords(string moduleAPIName)
        {
            try
            {
                Console.WriteLine("Fetching records for user - "
+ Initializer.GetInitializer().User.Email);
```

```
                RecordOperations recordOperation = new
RecordOperations();

                APIResponse response =
recordOperation.GetRecords(moduleAPIName, null, null);

                if (response != null)
                {
                    //Get the status code from response
                    Console.WriteLine("Status Code: " +
response.StatusCode);

                    if (new List() { 204, 304
}.Contains(response.StatusCode))
                    {
                        Console.WriteLine(response.StatusCode ==
204 ? "No Content" : "Not Modified");

                        return;
                    }

                    //Check if expected response is received
                    if (response.IsExpected)
                    {
                        //Get object from response
                        Com.Zoho.Crm.API.Record.ResponseHandler
responseHandler = response.Object;

                        if (responseHandler is
Com.Zoho.Crm.API.Record.ResponseWrapper)
                        {
                            //Get the received ResponseWrapper
instance

Com.Zoho.Crm.API.Record.ResponseWrapper responseWrapper =
(Com.Zoho.Crm.API.Record.ResponseWrapper)responseHandler;

                            //Get the list of obtained Record
instances
                            List records = responseWrapper.Data;
                            foreach
(Com.Zoho.Crm.API.Record.Record record in records)
                            {

Console.WriteLine(JsonConvert.SerializeObject(record));
                            }
                        }
                        //Check if the request returned an
exception
                        else if (responseHandler is
Com.Zoho.Crm.API.Record.APIException)
                        {
                            //Get the received APIException
instance
                            APIException exception =
(APIException)responseHandler;

                            //Get the Status
                            Console.WriteLine("Status: " +
exception.Status.Value);

                            //Get the Code
                            Console.WriteLine("Code: " +
exception.Code.Value);

                            Console.WriteLine("Details: ");

                            //Get the details map
                            foreach (KeyValuePair entry in
exception.Details)
                            {
                                //Get each value in the map
                                Console.WriteLine(entry.Key + ":
" + JsonConvert.SerializeObject(entry.Value));
                            }

                            //Get the Message
                            Console.WriteLine("Message: " +
exception.Message.Value);
                        }
                    }
                }
            }
            catch (System.Exception ex)
            {

Console.WriteLine(JsonConvert.SerializeObject(ex));
            }
        }
    }
}
,>
```

Single-threading in a Multi-user App

```
using System;
using System.Collections.Generic;
using System.Threading;
using Com.Zoho.API.Authenticator;
using Com.Zoho.API.Authenticator.Store;
using Com.Zoho.Crm.API;
using Com.Zoho.Crm.API.Dc;
using Com.Zoho.Crm.API.Logger;
using Com.Zoho.Crm.API.Record;
using Com.Zoho.Crm.API.Util;
using Newtonsoft.Json;
using static Com.Zoho.API.Authenticator.OAuthToken;

namespace csharpsdksampleapplication
{
    public class MultiThread
    {
        DataCenter.Environment environment;

        UserSignature user;

        Token token;

        string moduleAPIName;

        public MultiThread(UserSignature user,
DataCenter.Environment environment, Token token, string
```

```
moduleAPIName)
        {
            this.environment = environment;

            this.user = user;

            this.token = token;

            this.moduleAPIName = moduleAPIName;
        }

        static void Main(string[] args)
        {
            Logger logger = Logger.GetInstance(Logger.Levels.ALL,
"/Users/user_name/Documents/csharp_sdk_log.log");

            DataCenter.Environment environment1 =
USDataCenter.PRODUCTION;

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

            TokenStore tokenstore = new
FileStore("/Users/user_name/Documents/csharp_sdk_token.txt");

            Token token1 = new OAuthToken("clientId1",
"clientSecret1", "REFRESH/GRANT token", TokenType.REFRESH,
"redirectURL1");

            string resourcePath =
"/Users/user_name/Documents/csharpsdk-application";

            SDKConfig config = new
SDKConfig.Builder().SetAutoRefreshFields(true).Build();

            DataCenter.Environment environment2 =
EUDataCenter.PRODUCTION;

            UserSignature user2 = new
UserSignature("user2@zoho.eu");
```

```
            Token token2 = new OAuthToken("clientId2",
"clientSecret2", "REFRESH/GRANT token", TokenType.REFRESH,
"redirectURL2");

            Initializer.Initialize(user1, environment1, token1,
tokenstore, config, resourcePath, logger);

            MultiThread multiThread1 = new MultiThread(user1,
environment1, token1, "Vendors");

            Thread thread1 = new Thread(() =>
multiThread1.GetRecords());

            thread1.Start();

            MultiThread multiThread2 = new MultiThread(user2,
environment2, token2, "Quotes");

            Thread thread2 = new Thread(() =>
multiThread2.GetRecords());

            thread2.Start();

            thread1.Join();

            thread2.Join();
        }

        public void GetRecords()
        {
            try
            {
                SDKConfig config = new
SDKConfig.Builder().SetAutoRefreshFields(true).Build();

                Initializer.SwitchUser(this.user,
this.environment, this.token, config);

                Console.WriteLine("Fetching records for user - "
+ Initializer.GetInitializer().User.Email);
```

```
                RecordOperations recordOperation = new
RecordOperations();

                APIResponse response =
recordOperation.GetRecords(this.moduleAPIName, null, null);

                if (response != null)
                {
                    //Get the status code from response
                    Console.WriteLine("Status Code: " +
response.StatusCode);

                    if (new List() { 204, 304
}.Contains(response.StatusCode))
                    {
                        Console.WriteLine(response.StatusCode ==
204 ? "No Content" : "Not Modified");

                        return;
                    }

                    //Check if expected response is received
                    if (response.IsExpected)
                    {
                        //Get object from response
                        ResponseHandler responseHandler =
response.Object;

                        if (responseHandler is ResponseWrapper)
                        {
                            //Get the received ResponseWrapper
instance
                            ResponseWrapper responseWrapper =
(ResponseWrapper)responseHandler;

                            //Get the list of obtained Record
instances
                            List< Com.Zoho.Crm.API.Record.Record>
records = responseWrapper.Data;

                            foreach
(Com.Zoho.Crm.API.Record.Record record in records)
                            {

Console.WriteLine(JsonConvert.SerializeObject(record));
                            }
                        }
                        //Check if the request returned an
exception
                        else if (responseHandler is APIException)
                        {
                            //Get the received APIException
instance
                            APIException exception =
(APIException)responseHandler;

                            //Get the Status
                            Console.WriteLine("Status: " +
exception.Status.Value);

                            //Get the Code
                            Console.WriteLine("Code: " +
exception.Code.Value);

                            Console.WriteLine("Details: ");

                            //Get the details map
                            foreach (KeyValuePair entry in
exception.Details)
                            {
                                //Get each value in the map
                                Console.WriteLine(entry.Key + ":
" + JsonConvert.SerializeObject(entry.Value));
                            }

                            //Get the Message
                            Console.WriteLine("Message: " +
exception.Message.Value);
                        }
                    }
                }
            }
```

```
                catch (System.Exception ex)
                {

Console.WriteLine(JsonConvert.SerializeObject(ex));
                }
            }
        }
    }
    ,object>
```

Single-threading in a Single-user App

```
using System;
using System.Collections.Generic;
using System.Threading;
using Com.Zoho.API.Authenticator;
using Com.Zoho.API.Authenticator.Store;
using Com.Zoho.Crm.API;
using Com.Zoho.Crm.API.Dc;
using Com.Zoho.Crm.API.Logger;
using Com.Zoho.Crm.API.Record;
using Com.Zoho.Crm.API.Util;
using Newtonsoft.Json;
using static Com.Zoho.API.Authenticator.OAuthToken;

namespace csharpsdksampleapplication
{
    public class MultiThread
    {
        DataCenter.Environment environment;

        UserSignature user;

        Token token;

        string moduleAPIName;

        public MultiThread(UserSignature user,
DataCenter.Environment environment, Token token, string
moduleAPIName)
        {
            this.environment = environment;

            this.user = user;

            this.token = token;

            this.moduleAPIName = moduleAPIName;
        }

        static void Main(string[] args)
        {
            Logger logger = Logger.GetInstance(Logger.Levels.ALL,
"/Users/user_name/Documents/csharp_sdk_log.log");

            DataCenter.Environment environment1 =
USDataCenter.PRODUCTION;

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

            TokenStore tokenstore = new
FileStore("/Users/user_name/Documents/csharp_sdk_token.txt");

            Token token1 = new OAuthToken("clientId1",
"clientSecret1", "REFRESH/GRANT token", TokenType.REFRESH,
"redirectURL1");

            string resourcePath =
"/Users/user_name/Documents/csharpsdk-application";

            SDKConfig config = new
SDKConfig.Builder().SetAutoRefreshFields(true).Build();

            DataCenter.Environment environment2 =
EUDataCenter.PRODUCTION;

            UserSignature user2 = new
UserSignature("user2@zoho.eu");

            Token token2 = new OAuthToken("clientId2",
"clientSecret2", "REFRESH/GRANT token", TokenType.REFRESH,
"redirectURL2");

            Initializer.Initialize(user1, environment1, token1,
tokenstore, config, resourcePath, logger);

            MultiThread multiThread1 = new MultiThread(user1,
environment1, token1, "Vendors");

            Thread thread1 = new Thread(() =>
multiThread1.GetRecords());

            thread1.Start();

            MultiThread multiThread2 = new MultiThread(user2,
environment2, token2, "Quotes");

            Thread thread2 = new Thread(() =>
multiThread2.GetRecords());

            thread2.Start();

            thread1.Join();

            thread2.Join();
        }

        public void GetRecords()
        {
            try
            {
                SDKConfig config = new
SDKConfig.Builder().SetAutoRefreshFields(true).Build();

                Initializer.SwitchUser(this.user,
this.environment, this.token, config);

                Console.WriteLine("Fetching records for user - "
+ Initializer.GetInitializer().User.Email);

                RecordOperations recordOperation = new
RecordOperations();
```

```
                APIResponse response =
recordOperation.GetRecords(this.moduleAPIName, null, null);

                if (response != null)
                {
                    //Get the status code from response
                    Console.WriteLine("Status Code: " +
response.StatusCode);

                    if (new List() { 204, 304
}.Contains(response.StatusCode))
                    {
                        Console.WriteLine(response.StatusCode ==
204 ? "No Content" : "Not Modified");

                        return;
                    }

                    //Check if expected response is received
                    if (response.IsExpected)
                    {
                        //Get object from response
                        ResponseHandler responseHandler =
response.Object;

                        if (responseHandler is ResponseWrapper)
                        {
                            //Get the received ResponseWrapper
instance
                            ResponseWrapper responseWrapper =
(ResponseWrapper)responseHandler;

                            //Get the list of obtained Record
instances
                            List< Com.Zoho.Crm.API.Record.Record>
records = responseWrapper.Data;

                            foreach
(Com.Zoho.Crm.API.Record.Record record in records)
                            {
```

```
Console.WriteLine(JsonConvert.SerializeObject(record));
                            }
                        }
                        //Check if the request returned an
exception
                        else if (responseHandler is APIException)
                        {
                            //Get the received APIException
instance
                            APIException exception =
(APIException)responseHandler;

                            //Get the Status
                            Console.WriteLine("Status: " +
exception.Status.Value);

                            //Get the Code
                            Console.WriteLine("Code: " +
exception.Code.Value);

                            Console.WriteLine("Details: ");

                            //Get the details map
                            foreach (KeyValuePair entry in
exception.Details)
                            {
                                //Get each value in the map
                                Console.WriteLine(entry.Key + ":
" + JsonConvert.SerializeObject(entry.Value));
                            }

                            //Get the Message
                            Console.WriteLine("Message: " +
exception.Message.Value);
                        }
                    }
                }
            }
            catch (System.Exception ex)
            {
```

```
Console.WriteLine(JsonConvert.SerializeObject(ex));
                }
            }
        }
    }
,object>
```

Release Notes

Current Version

1. **ZCRMSDK - VERSION 3.1.0**

Install command

```
Install-Package ZCRMSDK -Version 3.1.0
dotnet add package ZCRMSDK --version 3.1.0
```

Notes

- Supported External ID.

Previous Versions

2. **ZCRMSDK - VERSION 3.0.1**

Install command

```
Install-Package ZCRMSDK -Version 3.0.1
dotnet add package ZCRMSDK --version 3.0.1
```

Notes

- Handled different field types.

3. **ZCRMSDK - VERSION 3.0.0**

Install command

```
Install-Package ZCRMSDK -Version 3.0.0
dotnet add package ZCRMSDK --version 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 CSharp 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 behaviours.
- **StreamWrapper** - A wrapper class for File operations.
- **APIResponse** - A common response instance for all the SDK method calls.