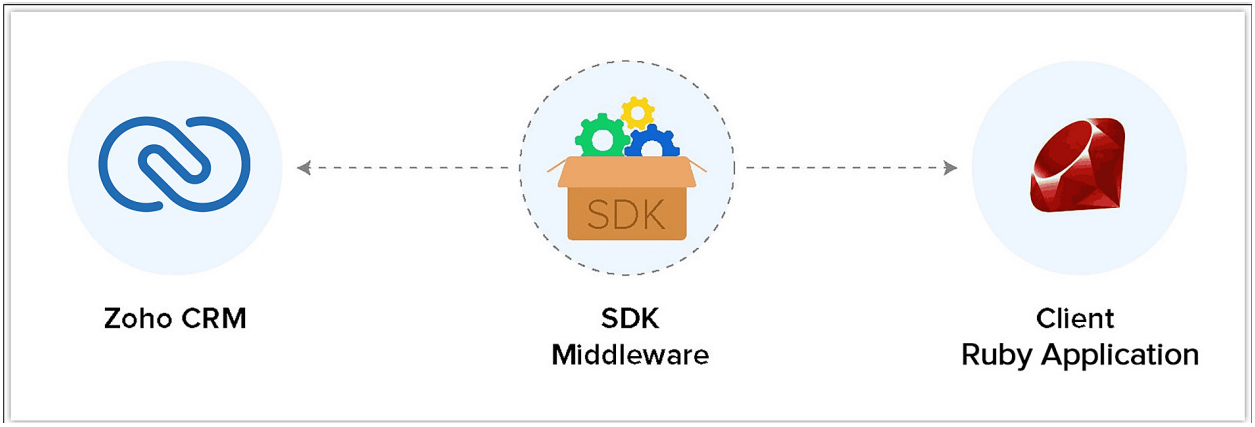
Ruby SDK
Version 2.x.x

Table of Contents

Overview

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

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



Ruby SDK allows you to:

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

Using the SDK

Add the below line in your client app Ruby files, where you would like to make use of the Ruby SDK.

```
1  require 'ZCRMSDK'
```

Through this line, you can access all the functionalities of the Ruby SDK.

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.

Installation of Ruby SDK

RUBY SDK requires Ruby (version 2.6 and above) to be set up in your development environment.

Including the SDK in your project

Ruby SDK is available through Gem . You can download the gem using:

```
gem install ZCRMSDK
```

You can include the SDK to your project using:

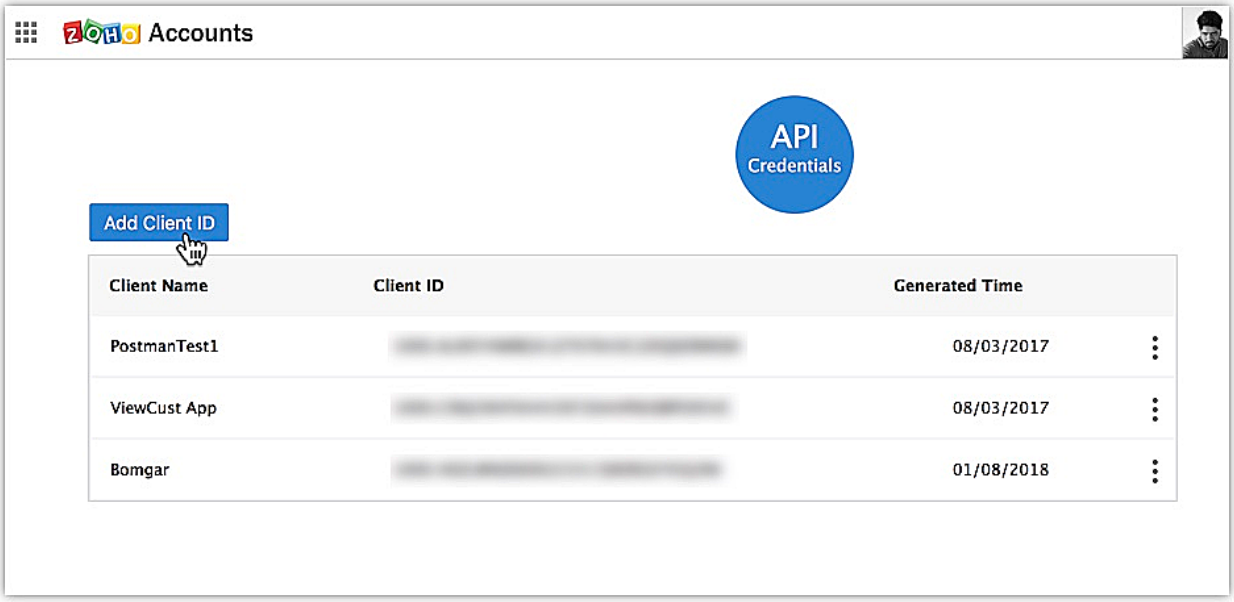
```
require 'ZCRMSDK'
```

Register your application

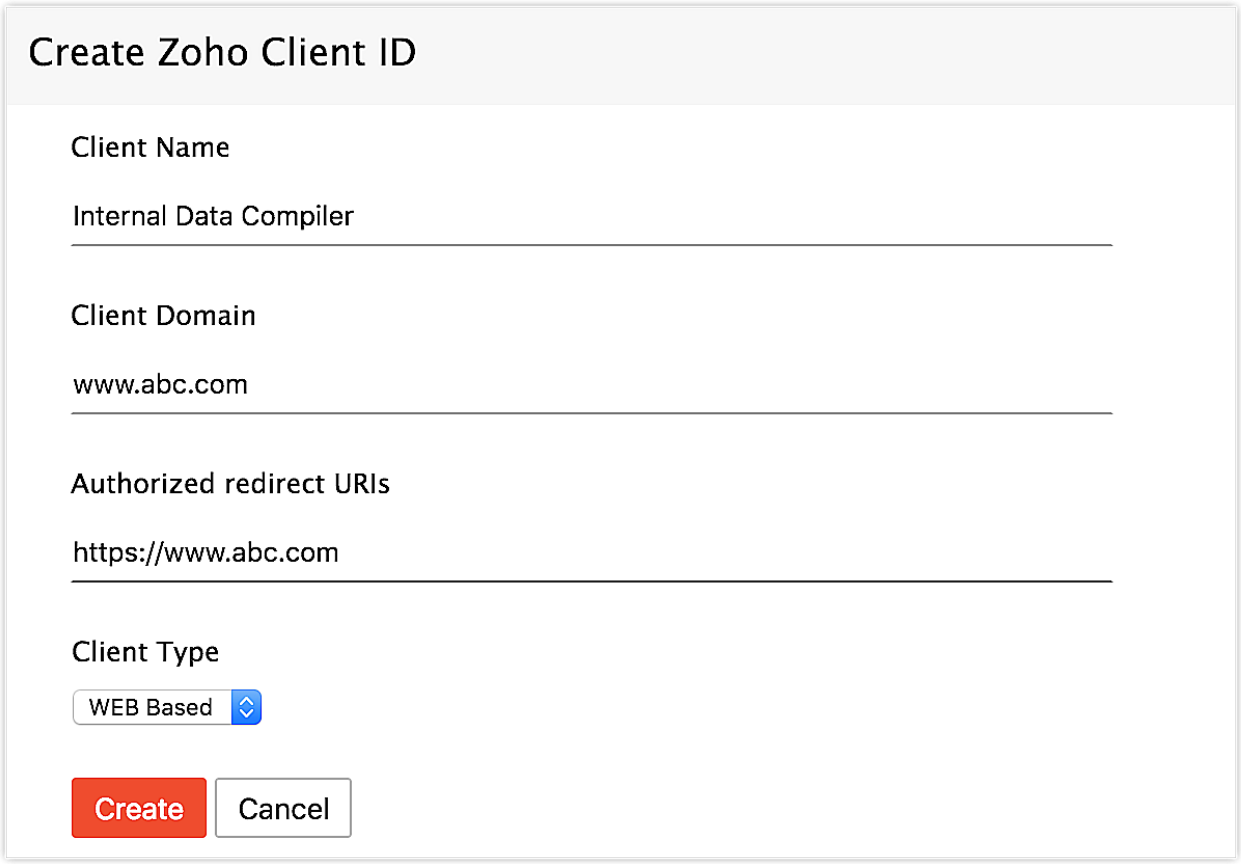
All the Zoho CRM APIs are authenticated with OAuth2 standards, so it is mandatory to register and authenticate your client app with Zoho.

To register:

1. Go to the site accounts.zoho.com/developerconsole
2. Click Add Client ID.



3. Enter the **Client Name**, **Client Domain** and **Authorized Redirect URL**.
4. Select the **Client Type** as **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 Ruby 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 SDKLog::Log **Logger** Class to log exception and API information.

```
#
# Create an instance of SDKLog::Log 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.
#
log = SDKLog::Log.initialize(Levels::INFO,"/Users/user_name/Documents/rubysdk_log.log")
```

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

```
#Create an UserSignature instance that takes user Email as parameter
```

```
user_signature = UserSignature.new('abc@zohocorp.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 DC::Domain::Environment
#Available Domains: USDataCenter, EUDataCenter, INDataCenter, CNDataCenter, AUDataCenter
#Available Environments: PRODUCTION, DEVELOPER, SANDBOX

environment = DC::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)

token = Authenticator::OAuthToken.new("clientId", "clientSecret", "REFRESH/GRANT token", TokenType::REFRESH/GRANT, "redirectURL")
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 = Store::DBStore.new("hostName", "dataBaseName",
```

```
    "userName", "password", "portNumber")
9
10 tokenstore =
   Store::FileStore.new("/Users/user_name/Documents/ruby_sdk_token.t
   xt")
11
12 tokenStore = CustomStore.new
```

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

```
1  # auto_refresh_fields
2      # if true - all the modules' fields will be auto-refreshed in
   the background, every    hour.
3      # 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
   (Util::ModuleFieldsHandler)
4      #
5      # pickListValidation
6      # if true - value for any picklist field will be validated
   with the available values.
7      # if false - value for any picklist field will not be
   validated, resulting in creation of a new value.
8      #
9      # open_timeout
10     # Number of seconds to wait for the connection to open
   (default 60 seconds)
11     #
12     # read_timeout
13     # Number of seconds to wait for one block to be read (via one
   read(2) call) (default 60 seconds)
14     #
15     # write_timeout
16     # Number of seconds to wait for one block to be written (via
   one write(2) call) (default 60 seconds)
17     #
18     # keep_alive_timeout
19     # Seconds to reuse the connection of the previous
   request(default 2 seconds)
20     #
```

```
21
22      sdk_config =
    SDKConfig::Builder.new.auto_refresh_fields(false).pick_list_valid
    ation(true).open_timeout(60).read_timeout(60).write_timeout(60).k
    eep_alive_timeout(2).build
```

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

```
1  resource_path = "/Users/user_name/Documents/rubysdk-application"
```

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

```
1  request_proxy = RequestProxy.new("proxyHost", "proxyPort",
   "proxyUser", "password")
```

9. Initialize the SDK and make API calls.

Token Persistence

Token persistence refers to storing and utilizing the authentication tokens that are provided by Zoho. There are three ways provided by the SDK in which persistence can be 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.

- **get_token(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.
- **save_token(UserSignature user, Token token)** - invoked after fetching access and

refresh tokens from Zoho.

- **delete_token(UserSignature user, Token token)** - invoked before saving the latest tokens.
- **get_tokens()** - The method to retrieve all the stored tokens.
- **delete_tokens()** - The method to delete all the stored tokens.

Custom Persistence

To use Custom Persistence, the user must extend Store::TokenStore and include the methods.

Here is the code:

```
using System;
require 'ZCRMSDK'
# This class stores the user token details to the file.
  class TokenStore
    # This method is used to get the user token details.
    # @param user A UserSignature class instance.
    # @param token A Token class instance.
    # @return A Token class instance representing the user token details.
    # @raise SDKException
    def get_token(user, token); end

    def get_tokens; end

    # This method is used to store the user token details.
    # @param user A UserSignature class instance.
    # @param token A Token class instance.
    # @raise SDKException
    def save_token(user, token); end

    # This method is used to delete the user token details.
    # @param user A User class instance.
    # @param token A Token class instance.
    # @raise SDKException
    def delete_token(token); end

    def delete_tokens; end
```

```
27  end
```

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:

```
1  /*
2  #Parameter containing the absolute file path to store tokens
3  tokenstore =
   FileStore.new("/Users/user_name/Documents/ruby_sdk_token.txt")
```

> Note
> You must not include "zcrm_oauthtokens.txt" in the path.

Database Persistence

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

- The MySQL should run in the same machine
- 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))

Note

- The default MySQL persistence requires MySQL2 to be installed. Use the below command to install MySQL2.

```
gem install mysql2 -v 0.5.2
```

- The order of precedence for token persistence is custom, file, followed by DB persistence. That is, if you provide the details for custom persistence (in persistence_handler_class_path) and file persistence details (in token_persistence_path), then custom persistence is given priority over file.

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"

```

```
8  tokenstore = Store::DBStore.new()
9  tokenstore = Store::DBStore.new("hostName", "dataBaseName",
   "userName", "password", "portNumber")
```

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.

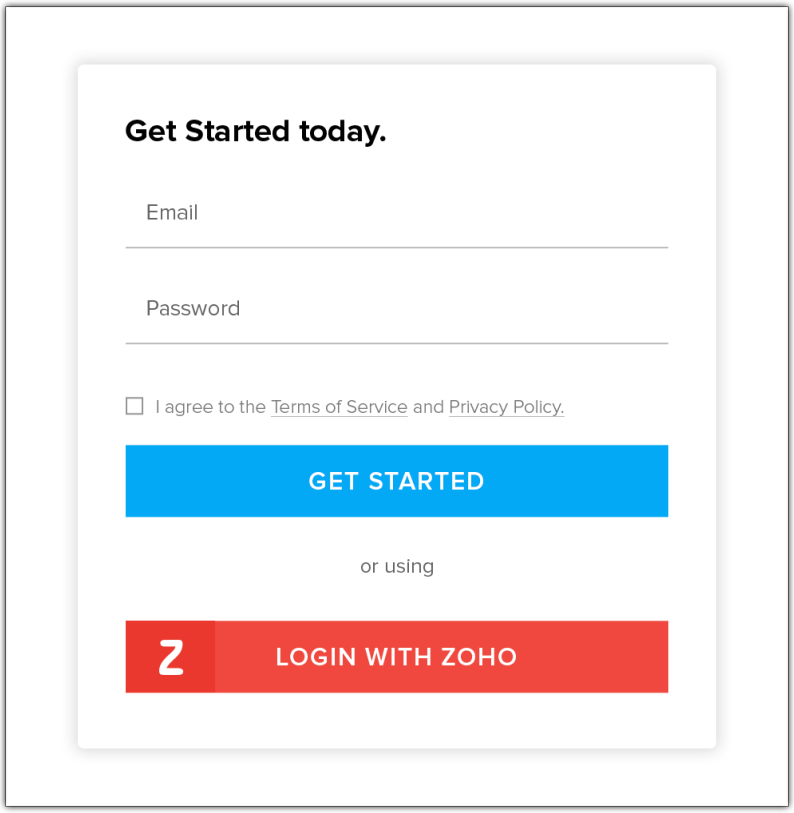
Note

- 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.

Initialize the SDK using following code

```
require 'ZCRMSDK'
class Initialize
    def self.initialize()
        # Create an instance of Log::SDKLog 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.
        log = SDKLog::Log.initialize(Levels::INFO,"/Users/user_name/Documents/rubysdk_log.log")

        #Create an UserSignature instance that takes user Email as parameter
        user_signature = UserSignature.new('abc@zohocorp.com')

        # Configure the environment
        # which is of the pattern Domain.Environment
        # Available Domains: USDataCenter, EUDataCenter, INDataCenter, CNDataCenter, AUDataCenter
        # Available Environments: PRODUCTION, DEVELOPER, SANDBOX
        environment = DC::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.(optional)
        token = Authenticator::OAuthToken.new("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 = Store::DBStore.new("hostName", "dataBaseName", "userName", "password", "portNumber")
```

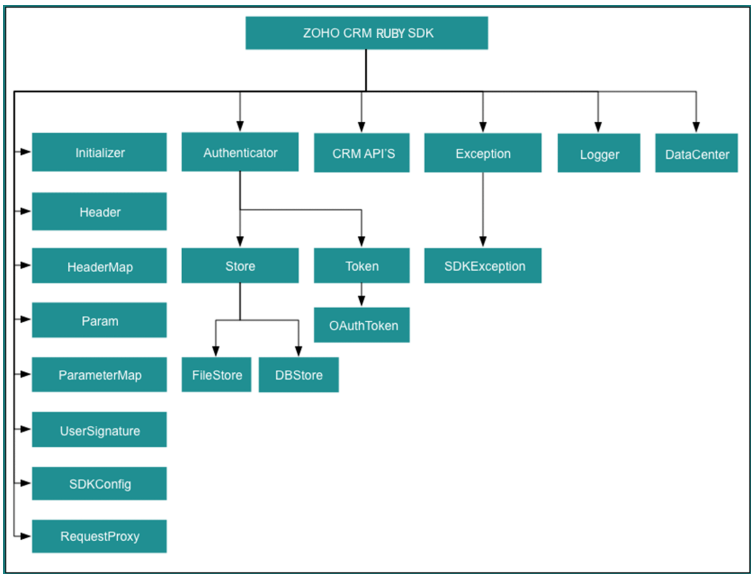
```
34
35          #tokenstore =
    Store::FileStore.new("/Users/user_name/Documents/ruby_sdk_token.txt")
36
37
38          # auto_refresh_fields
39      # if true - all the modules' fields will be auto-refreshed in the
    background, every    hour.
40      # if false - the fields will not be auto-refreshed in the
    background. The user can manually delete the file(s) or refresh the
    fields using methods from ModuleFieldsHandler
    (Util::ModuleFieldsHandler)
41      #
42      # pickListValidation
43      # if true - value for any picklist field will be validated with the
    available values.
44      # if false - value for any picklist field will not be validated,
    resulting in creation of a new value.
45      #
46      # open_timeout
47      # Number of seconds to wait for the connection to open (default 60
    seconds)
48      #
49      # read_timeout
50      # Number of seconds to wait for one block to be read (via one
    read(2) call) (default 60 seconds)
51      #
52      # write_timeout
53      # Number of seconds to wait for one block to be written (via one
    write(2) call) (default 60 seconds)
54      #
55      # keep_alive_timeout
56      # Seconds to reuse the connection of the previous request(default 2
    seconds)
57      #
58
59       sdk_config =
    SDKConfig::Builder.new.auto_refresh_fields(false).pick_list_validation(t
    rue).open_timeout(60).read_timeout(60).write_timeout(60).keep_alive_time
    out(2).build
60
61          resource_path = "/Users/user_name/Documents/rubysdk-application"
62          # Create an instance of RequestProxy class that takes the
    following parameters
```

```
63          # 1 -> Host
64          # 2 -> Port Number
65          # 3 -> User Name
66          # 4 -> Password
67          request_proxy = RequestProxy.new("proxyHost", "proxyPort",
    "proxyUser", "password")
68
69          # The initialize method of Initializer class that takes the
    following arguments
70          # 1 -> UserSignature instance
71          # 2 -> Environment instance
72          # 3 -> Token instance
73          # 4 -> TokenStore instance
74          # 5 -> SDKConfig instance
75          # 6 -> resourcePath -A String
76          # 7 -> Log instance (optional)
77          # 8 -> RequestProxy instance (optional)
78
79          #The following is the initialize method
80
81          Initializer.initialize(user, environment, token, store,
    sdk_config, resources_path, log, request_proxy)
82      end
83  end
```

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

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 Ruby SDK is depicted in the following image.



Responses and Exceptions

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

Use the **data_object** to get 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.

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

- For Record image download operation:
  -**APIResponse<DownloadHandler>**
- MassUpdate record operation:
  -**APIResponse<MassUpdateActionHandler>**
  -**APIResponse<MassUpdateResponseHandler>**

For GET Requests

- The **data_object** variable of the returned APIResponse instance returns the response handler interface.
- The ResponseHandler encompasses the **ResponseWrapper class** (for application/json responses), **FileBodyWrapper class** (for file download responses), and the **APIException class**.
- The CountHandler encompasses the **CountWrapper class** (for application/json responses) and the **APIException class**.
- The DeletedRecordsHandler encompasses the **DeletedRecordsWrapper class** (for application/json responses) and the **APIException class**.
- The DownloadHandler encompasses the **FileBodyWrapper class** (for file download responses responses) and the **APIException class**.
- The MassUpdateResponseHandler encompasses the **MassUpdateResponseWrapper class** (for application/json responses) and the **APIException class**.

For POST, PUT, DELETE Requests

- The **data_object** variable of the returned APIResponse instance returns the response handler interface.
- The ActionHandler encompasses the **ActionWrapper class** (for application/json responses) and the **APIException class**. The **ActionWrapper class** contains Property/Properties that may contain one/list of **ActionResponse interfaces**.
- The ActionResponse encompasses the **SuccessResponse class** (for application/json responses) and the **APIException class.**
- The ActionHandler interface encompasses the **ActionWrapper 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

BaseCurrency/ActionWrapper 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**.

> **Note**
>
> - 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.

under the **SDKException** class.

Sample Codes

All of Zoho CRM's APIs can be used through the Ruby 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**
Attachments::AttachmentsOperations.new(moduleAPIName, recordId)	Creates an AttachmentsOperations class instance with the moduleAPIName and recordId.

Method	**Description**
get_attachments	To fetch the list of attachments of a

	record.
upload_attachments	To upload attachments to a record.
delete_attachments	To delete the attachments that were added to a record.
delete_attachment	To delete an attachment that was added to a record.
download_attachment	To download an attachment that was uploaded to a record.
upload_link_attachments	To upload a link as an attachment to a record

Blueprint Operations

Constructor	**Description**
BluePrint::BluePrintOperations.new(recordId, moduleAPIName)	Creates a BluePrintOperations class instance with the recordId and moduleAPIName

Method	**Description**
get_blueprint	To get the next available transitions for that record, fields available for each transition, current value of each field, and

	validation(if any).
update_blueprint	To update a single transition at a time

Bulk Read Operations

Method	Description
create_bulk_read_job	To schedule a bulk read job to export records that match the criteria.
get_bulk_read_job_details	To know the status of the bulk read job scheduled previously.
download_result	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
upload_file	To upload a CSV file in ZIP format. The response contains the "file_id". Use this ID while making the bulk write request.
create_bulk_write_job	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.
get_bulk_read_job_details	To know the status of the bulk read job scheduled previously.
download_result	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	**Description**
get_contact_roles	To get the list of all contact roles.
create_contact_roles	To create contact roles.
update_contact_roles	To update contact roles.
delete_contact_roles	To delete contact roles.
get_contact_role	To get specific contact role.
update_contact_role	To update specific contact role.
delete_contact_role	To delete specific contact role

Currencies Operations

Method	Description
get_currencies	To get the list of all currencies available for your org.
add_currencies	To add new currencies to your org.
update_currencies	To update the currencies' details of your org.
enable_multiple_currencies	To enable multiple currencies for your org.
update_base_currency	To update the base currency details of your org.
get_currency	To get the details of specific currency.
update_currency	To update the details of specific currency

Custom View Operations

Constructor	Description
CustomViews::CustomViewsOperations.new(module)	Creates a CustomViewsOperations class instance with the moduleAPIName

Method	Description

get_custom_views	To get the list of all custom views in a module.
get_custom_view	To get the details of specific custom view in a module

Fields Metadata Operations

Constructor	**Description**
Fields::FieldsOperations.new(module)	Creates a FieldsOperations class instance with the module

Method	**Description**
get_fields	To get the meta details of all fields in a module.
get_field	To get the meta details of specific field in a module

Files Operations

Method	**Description**
upload_files	To upload files and get their encrypted IDs.
get_file	To get the uploaded file through its

	encrypted ID

Layouts Operations

Constructor	**Description**
Layouts::LayoutsOperations.new(module)	Creates a LayoutsOperations class instance with the moduleAPIName

Method	**Description**
get_layouts	To get the details of all the layouts in a module.
get_layout	To get the details (metadata) of a specific layout in a module

Modules Operations

Method	**Description**
get_modules	To get the details of all the modules.
get_module	To get the details (metadata) of a specific module.
update_module_by_api_name	To update the details of a module by its module API name.

update_module_by_id	To update the details of a module by its ID

Notes Operations

Method	Description
get_notes	To get the list of notes of a record.
create_notes	To add new notes to a record.
update_notes	To update the details of the notes of a record.
delete_notes	To delete the notes of a record.
get_note	To get the details of a specific note.
update_note	To update the details of an existing note.
delete_note	To delete a specific note

Notification Operations

Method	Description
enable_notifications	To enable instant notifications of actions performed on a module.
get_notification_details	To get the details of the notifications

	enabled by the user.
update_notifications	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.
update_notification	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.
disable_notifications	To stop all the instant notifications enabled by the user for a channel.
disable_notification	To disable notifications for the specified events in a channel

Organization Operations

Method	**Description**
get_organization	To get the details of your organization.
upload_organization_photo	To upload a photo of your organization

Profile Operations

Constructor	**Description**
Profiles::ProfilesOperations.new(OffsetDa	Creates a ProfilesOperations class

teTime ifModifiedSince)	instance with the value of the If-Modified-Since header

Method	**Description**
get_profiles	To get the list of profiles available for your organization.
get_profile	To get the details of a specific profile

Query (COQL) Operation

Method	**Description**
get_records	To get the records from a module through a COQL query

Records Operations

Method	**Description**
get_record	To get a specific record from a module.
update_record	To update a specific record in a module.
delete_record	To delete a specific record from a module.
get_records	To get all records from a module.

create_records	To insert records in a module.
update_records	To update records in a module.
delete_records	To delete records from a module.
upsert_records	To insert/update records in a module.
get_deleted_records	To get the deleted records from a module.
search_records	To search for records in a module that match certain criteria, email, phone number, or a word.
convert_lead	To convert records(Leads to Contacts/Deals).
get_photo	To get the photo of a record.
upload_photo	To upload a photo to a record.
delete_photo	To delete the photo of a record.
mass_update_records	To update the same field for multiple records in a module.
get_mass_update_status	To get the status of the mass update job scheduled previously.

Related List Operations

Method	Description
get_record	To get a specific record from a module.

Method	Description
get_related_lists	To get the details of all the related lists of a module.
get_related_list	To get the details of a specific related list of a module

Related Records Operations

Constructor	Description
RelatedRecords::RelatedRecordsOperations.new(relatedListAPIName, recordId, moduleAPIName)	Creates a RelatedRecordsOperations class instance with the relatedListAPIName, recordId, and moduleAPIName

Method	Description
get_related_records	To get list of records from the related list of a module.

update_related_records	To update the association/relation between the records.
delink_records	To delete the association between the records.
get_related_record	To get the records from a specific related list of a module.
update_related_record	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**
get_roles	To get the list of all roles available in your organization.
get_role	To get the details of a specific role

Shared Records Operations

Constructor	**Description**
ShareRecords::ShareRecordsOperations.new(recordId, moduleAPIName)	Creates a ShareRecordsOperations class instance with the recordId and

	moduleAPIName

Method	**Description**
get_shared_record_details	To get the details of a record shared with other users.
share_record	To share a record with other users in the organization.
update_share_permissions	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.
revoke_shared_record	To revoke access to a shared record

Tags Operations

Method	Description
get_tags	To get the list of all tags in your organization.
create_tags	To create tags.
update_tags	To update multiple tags.
update_tag	To update a specific tag.
delete_tag	To delete a specific tag from the module.
merge_tags	To merge two tags.
add_tags_to_record	To add tags to a specific record.
remove_tags_from_record	To remove tags from a record.
add_tags_to_multiple_records	To add tags to multiple records.
remove_tags_from_multiple_records	To remove tags from multiple records.
get_record_count_for_tag	To get the record count for a tag

Taxes Operations

Method	Description
get_taxes	To get the taxes of your organization.
create_taxes	To add taxes to your organization.
update_taxes	To update the existing taxes of your organization.
delete_taxes	To delete multiple taxes from your organization.
get_tax	To get the details of a specific tax.
delete_tax	To delete a specific tax from your organization

Territory Operations

Method	Description
get_territories	To get the list of all territories.
get_territory	To get the details of a specific territory

Users Operations

Method	Description
get_users	To get the list of users in your organization.
create_user	To add a user to your organization.
update_users	To update the existing users of your organization.
get_user	To get the details of a specific user.
update_user	To update the details of a specific user.
delete_user	To delete a specific user from your organization

Variable Groups Operations

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

Variables Operations

Method	Description
get_variables	To get the list of variables available for your organization.
create_variables	To add new variables to your organization.
update_variables	To update the details of variables.
delete_variables	To delete multiple variables.
get_variable_by_id	To get the details of a specific variable by its unique ID.
update_variable_by_id	To update the details of a specific variable by its unique ID.
delete_variable	To delete a specific variable.
get_variable_for_api_name	To get the details of a variable by its API name.
update_variable_by_api_name	To update the details of a variable by its API name

Threading

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

The Ruby SDK (from version 2.x.x) supports both single-threading and multi-threading irrespective of a single-user or a multi-user app.

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

Multi-threading in a Multi-user App

- The program execution starts from **execute()**.
- 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 **MultiThreading** class is created.
- When **t1.join** is called which in-turn invokes the **thread** which has the details of user1 are passed to the **switch_user** function through the **func1()**. Therefore, this creates a thread for user1.
- Similarly, When the **t2.join** is invoked , the details of user2 are passed to the switch_user function through the **func1()**. Therefore, this creates a thread for user2.

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

```
1  Initializer.switch_user(user, environment, token, sdk_config)
2
3  Initializer.switch_user(user, environment, token, sdk_config, proxy)
```

```
1  require 'ZCRMSDK'
2
3  module MultiUser
4
5      class MultiThreading
6          def initialize(module_api_name)
7              @module_api_name = module_api_name
8          end
9          def execute(user_signature, environment, token,tokenstore,
   sdk_config,resources_path, log, proxy)
10             Initializer.initialize(user_signature, environment, token,
   tokenstore, sdk_config, resources_path, log)
```

```
            token1 =Authenticator::OAuthToken.new("clientId", "clientSecret", "REFRESH/GRANT token", TokenType::REFRESH/GRANT, "redirectURL")
            user1 = UserSignature.new('abc@zohocorp.com')
            environment1 = DC::USDataCenter::PRODUCTION
            sdk_config1 = SDKConfig::Builder.new.auto_refresh_fields(false).pick_list_validation(true).build
            t1 = Thread.new{func1(user1,environment1,token1,sdk_config1)}
            token2 = Authenticator::OAuthToken.new("clientId", "clientSecret", "REFRESH/GRANT token", TokenType::REFRESH/GRANT, "redirectURL")
            user2 = UserSignature.new('dfg@zohocorp.com')
            environment2 = DC::USDataCenter::PRODUCTION
            sdk_config2 = SDKConfig::Builder.new.auto_refresh_fields(false).pick_list_validation(true).build
            t2 = Thread.new{func1(user2,environment2,token2,sdk_config2)}
            t1.join
            t2.join
        end
        def func1(user,environment,token,sdk_config)
            Initializer.switch_user(user,environment,token,sdk_config)
            print Initializer.get_initializer.user.email
            ro = Record::RecordOperations.new
            ro.get_records(nil,nil,@module_api_name)
        end
    end
end

log = SDKLog::Log.initialize(Levels::INFO,"/Users/user_name/Documents/rubysdk_log.log")
user_signature = UserSignature.new('abc@zohocorp.com')
environment = DC::USDataCenter::PRODUCTION
token = Authenticator::OAuthToken.new("clientId", "clientSecret", "REFRESH/GRANT token", TokenType::REFRESH/GRANT, "redirectURL")
tokenstore = Store::FileStore.new("/Users/user_name/Documents/ruby_sdk_token.txt")
sdk_config = SDKConfig::Builder.new.auto_refresh_fields(false).pick_list_validation(true).build
```

```
proxy = RequestProxy.new("proxyHost", "proxyPort", "proxyUser", "password")
module_api_name = "Leads"
resource_path = "/Users/user_name/Documents"
MultiUser::MultiThreading.new(module_api_name).execute(user_signature, environment, token,tokenstore, sdk_config,resource_path, log,proxy)
```

Multi-threading in a Single-user App

```
require 'ZCRMSDK'
module SingleUser
    class MultiThreading

        def execute(user_signature, environment, token,tokenstore, sdk_config,resources_path, log,proxy)
            Initializer.initialize(user_signature, environment, token, tokenstore, sdk_config, resources_path, log)
            t1 = Thread.new{func1("Leads")}
            t2 = Thread.new{func1("Deals")}
            t1.join
            t2.join
        end
        def func1(module_api_name)
            ro = Record::RecordOperations.new
            ro.get_records(nil,nil,module_api_name).inspect
        end

    end
end

log = SDKLog::Log.initialize(Levels::INFO,"/Users/user_name/Documents/rubysdk_log.log")
user_signature = UserSignature.new('abc@zohocorp.com')
environment = DC::USDataCenter::PRODUCTION
token = Authenticator::OAuthToken.new("clientId", "clientSecret", "REFRESH/GRANT token", TokenType::REFRESH/GRANT, "redirectURL")
tokenstore = Store::FileStore.new("/Users/user_name/Documents/ruby_sdk_token.txt")
sdk_config = SDKConfig::Builder.new.auto_refresh_fields(false).pick_list_validation(true).build
proxy = RequestProxy.new("proxyHost", "proxyPort", "proxyUser",
```

```
   "password")
27 resource_path = "/Users/user_name/Documents/rubysdk-application"
28 SingleUser::MultiThreading.new.execute(user_signature, environment,
   token,tokenstore, sdk_config,resource_path, log,proxy)
```

SDK Sample Code

```
1  require 'ZCRMSDK'
2  require 'date'
3  class Records
4    def get_records
5      # Create an instance of Log::SDKLog Class that takes two parameters
6      #1 -> Level of the log messages to be logged. Can be configured by
   typing Levels "::" and choose any level from the list displayed.
7      # 2 -> Absolute file path, where messages need to be logged.
8
9      log =
   SDKLog::Log.initialize(Levels::INFO,"/Users/user_name/Documents/rubysdk_
   log.log")
10
11     #Create an UserSignature instance that takes user Email as parameter
12     user_signature = UserSignature.new('abc@zohocorp.com')
13
14     # Configure the environment
15     # which is of the pattern Domain.Environment
16     # Available Domains: USDataCenter, EUDataCenter, INDataCenter,
   CNDataCenter, AUDataCenter
17     # Available Environments: PRODUCTION, DEVELOPER, SANDBOX
18
19     environment = DC::USDataCenter::PRODUCTION
20
21     #Create a Token instance
22     #1 -> OAuth client id.
23     #2 -> OAuth client secret.
24     #3 -> REFRESH/GRANT token.
25     #4 -> Token type(REFRESH/GRANT).
26     #5 -> OAuth redirect URL.(optional)
27
28     token = Authenticator::OAuthToken.new("clientId", "clientSecret",
   "REFRESH/GRANT token", TokenType::REFRESH/GRANT, "redirectURL")
29
30     #Create an instance of TokenStore.
31     #1 -> DataBase host name. Default "localhost"
```

```
32      #2 -> DataBase name. Default "zohooauth"
33      #3 -> DataBase user name. Default "root"
34      #4 -> DataBase password. Default ""
35      #5 -> DataBase port number. Default "3306"
36
37      store = Store::DBStore.new("hostName", "dataBaseName", "userName",
    "password", "portNumber")
38
39      #store =
    Store::FileStore.new("/Users/user_name/Documents/ruby_sdk_token.txt"
40
41      # auto_refresh_fields
42      # if true - all the modules' fields will be auto-refreshed in the
    background, every    hour.
43      # 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
    (Util::ModuleFieldsHandler)
44      #
45      # pick_list_validation
46      # 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.
47     # if true - the SDK validates the input. If the value does not exist
    in the pick list, the SDK throws an error.
48     # if false - the SDK does not validate the input and makes the API
    request with the user's input to the pick list
49      sdk_config =
    SDKConfig::Builder.new.auto_refresh_fields(false).pick_list_validation(t
    rue).build
50
51      resource_path = "/Users/user_name/Documents/rubysdk-application"
52      # Create an instance of RequestProxy class that takes the following
    parameters
53      # 1 -> Host
54      # 2 -> Port Number
55      # 3 -> User Name
56      # 4 -> Password
57
58      request_proxy = RequestProxy.new('proxyHost', 'proxyPort',
    'proxyUser', 'password')
59
60      # The initialize method of Initializer class that takes the
    following arguments
61      # 1 -> UserSignature instance
```

```
    # 2 -> Environment instance
    # 3 -> Token instance
    # 4 -> TokenStore instance
    # 5 -> SDKConfig instance
    # 6 -> resourcePath -A String
    # 7 -> Log instance (optional)
    # 8 -> RequestProxy instance (optional)

    #The following is the initialize method

    Initializer.initialize(user_signature, environment, token, store, sdk_config, resources_path, log, request_proxy)
    # Get instance of RecordOperations Class
    ro = Record::RecordOperations.new
    # Get instance of ParameterMap Class
    pm = ParameterMap.new
    pm.add(Record::RecordOperations::GetRecordParam.approved, 'false')
    pm.add(Record::RecordOperations::GetRecordParam.converted, 'false')
    hm = HeaderMap.new
    hm.add(Record::RecordOperations::GetRecordHeader.If_modified_since, DateTime.new(2019, 8, 10, 4, 11, 9, '+03:00'))
    module_api_name = "Leads"
    response = ro.get_records(pm, hm, module_api_name)
    unless response.nil?
      status_code = response.status_code
      # Get the status code from response
      print "\n Status Code :" + status_code.to_s
      if [204, 304].include? status_code
        print(status_code == 204 ? 'No Content' : 'Not Modified')
        return
      end
      # Check if expected instance is received.
      if response.is_expected
        # Get object from response
        response_handler = response.data_object
        # Check if expected ResponseWrapper instance is received
        if response_handler.is_a? Record::ResponseWrapper

          records = response_handler.data
          records.each do |record|
            # Get the ID of each Record
            print "\n Record ID: "
            print record.id.to_s
            created_by = record.created_by
```

```
104             # Check if created_by is not None
105             unless created_by.nil?
106               # Get the Name of the created_by User
107               print "\n Record Created By User-Name: "
108               print created_by.name
109               # Get the ID of the created_by User
110               print "\n Record Created By User-Id: "
111               print created_by.id.to_s
112               # Get the Email of the created_by User
113               print "\n Record Created By User-Email: "
114               print created_by.email
115             end
116             # Get the CreatedTime of each Record
117             print "\n Record CreatedTime: "
118             print record.created_time
119             # Get the modified_by User instance of each Record
120             modified_by = record.modified_by
121             # Check if modifiedBy is not None
122             unless modified_by.nil?
123               # Get the Name of the modified_by User
124               print "\n Record Modified By User-Name: "
125               print modified_by.name
126               # Get the ID of the modified_by User
127               print "\n Record Modified By User-Id: "
128               print modified_by.id.to_s
129               # Get the Email of the modified_by User
130               print "\n Record Modified By User-Email: "
131               print modified_by.email
132             end
133             # Get the ModifiedTime of each Record
134             print "\n Record ModifiedTime: "
135             print record.modified_time
136             tags = record.tag
137             if !tags.nil? && tags.size.positive?
138               tags.each do |tag|
139                 # Get the Name of each Tag
140                 print "\n Record Tag Name: "
141                 print tag.name
142                 # Get the Id of each Tag
143                 print "\n Record Tag ID: "
144                 print tag.id.to_s
145               end
146             end
147             # To get particular field value
```

```
            print "\n Record Field Value: "
            print record.get_key_value('Last_Name')
            # To get particular KeyValues
            print "\n Record KeyValues:"
            record.get_key_values.each do |key_name, value|
              print "\n "
              unless value.nil?
                print key_name
                print value
              end
            end
          end
        end
      end
    end
  end
end
Records.new.get_records
```

Release Notes

Current Version

1. ZCRMSDK -VERSION 2.1.0

Install command

```
gem install ZCRMSDK -v 2.1.0
```

Enhancements

- Supported External ID

Previous Versions

2. ZCRMSDK -VERSION 2.0.0

Install command

```
gem install ZCRMSDK -v 2.0.0
```

Notes

- 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.