

Ruby SDK

Version 1.x.x



Zoho CRM
zoho.com/crm-

Table of Contents

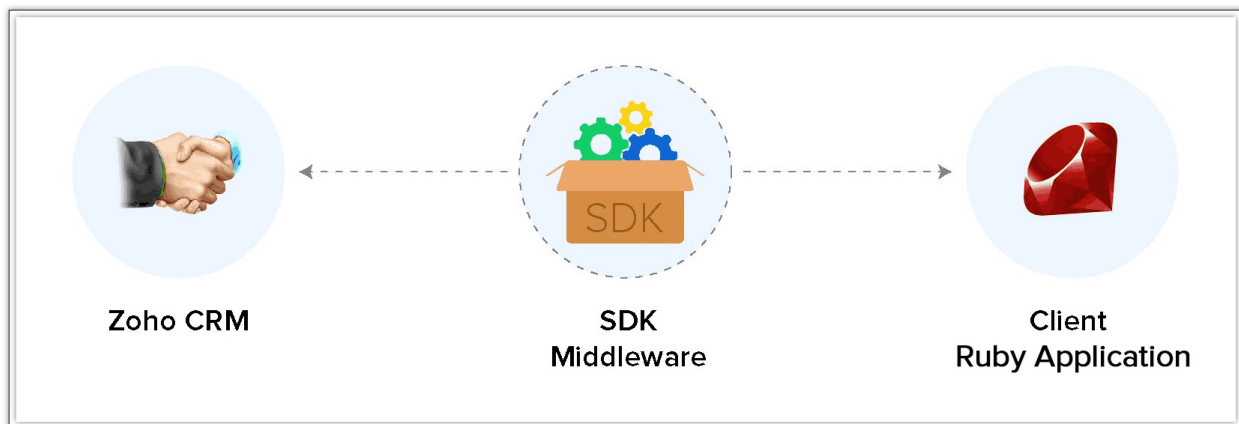
1. Overview	3
a. Using the SDK	
2. Registering a client	4
3. Installation of Ruby SDK	6
4. Configurations	6
a. Configuration Hash	
5. Token Persistence	10
a. Custom Persistence	
b. File Persistence	
c. Database Persistence	
6. Initialization	12
a. Generating self-authorized grant and refresh token	
b. App Startup	
7. Class Hierarchy	14
8. Response & Exceptions	16
9. Sample Codes	18
a. Rest Client Operations	
b. Organization Operations	
c. Module Operations	
d. Record Operations	
e. Note and Tag Operations	
12. Release Notes	25
a. Current Version	
b. Previous Version(s)	



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 middle ware 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

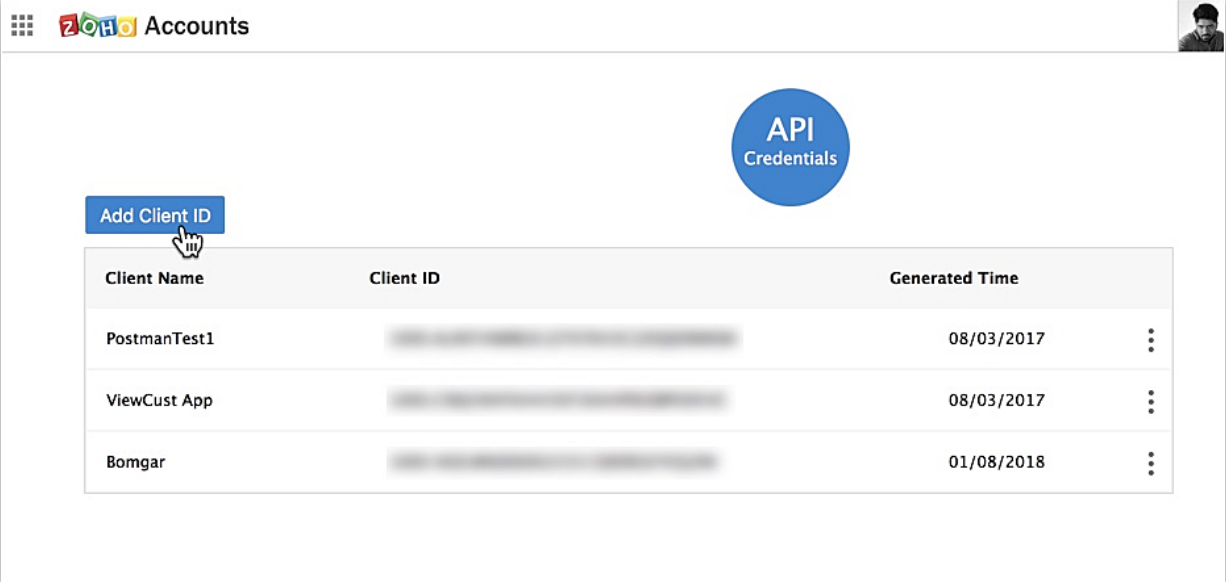
- **The access and refresh tokens are environment-specific and domain-specific.** When you handle various environments and domains such as Production, Sandbox, or Developer and IN, CN, US, EU, or AU, respectively, you must use the access token and refresh token generated only in those respective environments 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.

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.



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**.

Create Zoho Client ID

Client Name

Internal Data Compiler


Client Domain

www.abc.com

Authorized redirect URIs

https://www.abc.com

Client Type

WEB Based 

Create

Cancel

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.



Zoho CRM
zoho.com/crm-

Installation of Ruby SDK

Run the following command

```
1 gem install ZCRMSDK
```

The Ruby SDK will be installed.

Configuration

To access CRM services through the SDK, the client application must be first authenticated. This can be done by passing a key-value configuration pair to the initialization process.

- Create the config_details hash containing the authentication credentials required.
- Pass the configuration hash using the method `ZCRMSDK::RestClient::ZCRMRestClient.init(config_details)`.

Sample:

```
1 config_details = { 'client_id' => 'xxxxx', #mandatory
2 'client_secret' => 'xxxxxx', #mandatory
3 'redirect_uri' => 'https://www.zoho.com', #mandatory
4 'api_base_url' => 'https://www.zohoapis.com', #optional, default
  is https://www.zohoapis.com
5 'accounts_url' => 'https://accounts.zoho.com', #optional, default
  is https://accounts.zoho.com
6 'current_user_email' => 'abc@xyz.com', #mandatory (should be
  either set in config hash or
  ZCRMSDK::RestClient::ZCRMRestClient.current_user_email = "current
  user email id" )
7 'api_version' => 'v2', #optional,default is v2
8 'sandbox' => 'false', #optional default is sandbox
9 'application_log_file_path' => "/Users/xxxx/Desktop/log.log",
  #optional, absolute path of log file
10 'log_in_console' => 'true', #optional default is false, to log on
  the console
11 'persistence_handler_class_path' =>
  'absolute/path/to/customClass.ruby', #absolute path to the custom
  db implementation
12 'persistence_handler_class' =>'classname', #name of the class
13 'token_persistence_path'=>'relativepath/to/zcrm_oauthtokens.txt',
```



```

    #for file persistence
14 'db_username' => 'username', #optional, default root
15 'db_password' => 'password', #optional, default ''
16 'db_port' => 'port_number', #optional, default 3306
17 }

```

Configuration hash

The user must pass the configuration values in a hash map (key-value pairs) as argument to the `ZCRMRestClient.init(config_details)` function. Below is the list of keys that must be in the hash.

Mandatory Keys	Optional Keys
client_id	application_log_file_path
client_secret	sandbox
redirect_uri	api_base_url
current_user_email	api_version
	accounts_url
	persistence_handler_class
	persistence_handler_class_path
	token_persistence_path
	db_port



	db_username
	db_password

Mandatory properties

- **client_id**, **client_secret**, and **redirect_uri** are your OAuth client's configurations that you get after registering your Zoho client.
- **current_user_email** - In case of single user, this configuration can be set using "ZCRMSDK::RestClient::ZCRMRestClient.current_user_email = "current user email id".

Additional properties

- **api_base_url** - URL to be used when calling an API. It denotes the domain of the user. This URL may be:
 - <https://www.zohoapis.com> (default)
 - <https://www.zohoapis.eu>
 - <https://www.zohoapis.com.cn>
 - <https://www.zohoapis.jp>
- **api_version** is "v2".
- **accounts_url** - Default value is set as US domain. This value can be changed based on your domain (EU, CN).
 - <https://accounts.zoho.com>
 - <https://accounts.zoho.eu>
 - <https://accounts.zoho.com.cn>
 - <https://accounts.zoho.jp>
- **sandbox** - To make API calls to sandbox account, change the value of this key to true. By default, the value is false.
- **application_log_file_path** - The SDK stores the log information in a file.
 - The user must specify this path.
 - In case the path is not specified, the log file will be created inside the project.
- **persistence_handler_class** is the class name that implements token persistence. Refer to this page for more details.
- **persistence_handler_class_path** is the absolute path to the customclass.ruby file. Refer to this page for more details.



- **log_in_console** - To print the exceptions on the console screen, when set to "true".

Note

- If the optional keys are not specified, their default values will be assigned automatically.
- The **api_base_url** and **accounts_url** keys are mandatory in case the user is not in the "com" domain.
- 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.

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).

Custom Persistence

In Ruby SDK, the user can have his own implementation of persistence. You need to provide the file path of the Ruby file that implements the custom persistence, in the **persistence_handler_class_path** and the name of the class in **persistence_handler_class keys** of the configuration array (key-value pair), where **persistence_handler_class_path** is the absolute path to the custom DB implementation and **persistence_handler_class** is the class name.

You can write an implementation of the inbuilt **ZohoPersistenceHandler** interface,



which must contain the following callback methods.

- **saveOAuthData(ZohoOAuthTokens tokens)** - invoked while fetching access and refresh tokens from Zoho.
- **deleteOAuthTokens(\$userEmailId)** - invoked before saving the newly received tokens.
- **getOAuthTokens(\$userEmailId)** - invoked before firing a request to fetch the saved tokens. This method should return ZohoOAuthTokens object for the library to process it.

File Persistence

In case of file persistence, you can also set up persistence as a file in the local drive, and set the file path in the **token_persistence_path**, of the configuration array (key-value pair).

- If you use file persistence, you must create an empty file with the file name **zcrm_oauthtokens.txt**.
- Provide the file path of the **folder** containing the zcrm_oauthtokens.txt file in the token_persistence_path key.

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
 - useridentifier (varchar(100))
 - accesstoken (varchar(100))
 - refreshtoken (varchar(100))
 - expirytime (bigint)

Note

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

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

- The order of precedence for token persistence is custom, file, followed by DB

Initialization

The app would be ready to be initialized after defining the OAuth configuration hash. The user can now proceed to generate the required tokens to run the app.

The generation of the grant token can be done using two methods.

- Self-Client
- [Redirection-based code generation](#)

We will be using the self-client option here to demonstrate the process.

Generating self-authorized grant and refresh token

For self client apps, the self authorized grant token should be generated from the Zoho Developer Console (<https://accounts.zoho.com/developerconsole>)

1. Go to [Zoho Developer Console](#)
2. Click **Options** → **Self Client** of the client for which you wish to authorize.



3. Enter one or more (comma separated) valid Zoho CRM scopes, that you wish to authorize, in the Scope field and choose a time of expiry. Provide **aaaserver.profile.READ** scope along with Zoho CRM scopes.
4. Copy the **grant token** for backup.
5. Generate refresh_token from grant token by making a POST request with the URL below

```
1 https://accounts.zoho.com/oauth/v2/token?code={grant_token}&redir
```

6. Copy the **refresh token** for backup.

Please note that the generated grant token is valid only for the stipulated time you choose while generating it. Hence, the access and refresh tokens should be generated within that time.

Generating an access token

Access token can be generated from a grant token or refresh token. Following any one of the two methods is sufficient.

Generating Access token from Grant token

The following code snippet should be executed from your main class to get access and refresh tokens. Paste the copied grant token in the string literal mentioned below. This is a one-time process.

```
1 client = ZCRMSDK::OAuthClient::ZohoOAuth.getClientInstance
2 grant_token = 'grant_token'
3 client.generate_access_token(grant_token)
```

Please note that the above code snippet is valid only once per grant token. Upon successful execution of the above code snippet, the generated access and refresh tokens would have been persisted through our persistence handler class.

Generating Access token from Refresh token

The following code snippet should be executed from your main class to get access and refresh tokens. Please paste the generated refresh token in the string literal mentioned

below. This is a one-time process.

```
1 client = ZCRMSDK::OAuthClient::ZohoOAuth.get_client_instance
2 refresh_token = 'refresh_token'
3 client.generate_access_token_from_refresh_token(refresh_token)
```

Upon successful execution of the above code snippet, the generated access token and given refresh token would have been persisted through our persistence handler class.

Once the OAuth tokens have been persisted, subsequent API calls would use the persisted access and refresh tokens. The Ruby SDK will take care of refreshing the access token using refresh token, as and when required.

App Startup

The Ruby SDK requires the following line of code invoked every time your client app is started.

```
1 ZCRMSDK::RestClient::ZCRMRestClient.init(config_details)
```

Once the SDK has been initialized by the above line, you can use any APIs of the SDK to get proper results.

Note

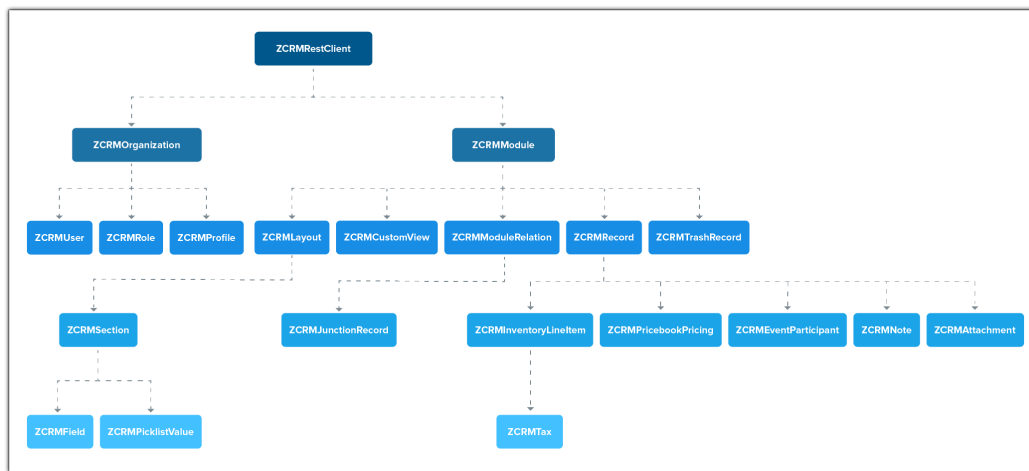
- The **access and refresh tokens are environment-specific and domain-specific**. When you handle various environments and domains such as Production, Sandbox, or Developer and IN, CN, US, EU, or AU, respectively, you must use the access token and refresh token generated only in those respective environments 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

Class Hierarchy

All Zoho CRM entities are modelled as classes having members and methods applicable to that particular entity.

- ZCRMRestClient is the base class of the SDK. This class has methods to get instances of various other Zoho CRM entities.
- The class relations and hierarchy of the SDK follows the entity hierarchy inside Zoho CRM.
- Each class entity has functions to fetch its own properties and to fetch data of its immediate child entities through an API call. For example, a Zoho CRM module (ZCRMModule) object will have member functions to get a module's properties like display name, module ID etc, and will also have the functions to fetch all its child objects (like Layout).

The class hierarchy of various Zoho CRM entities is depicted as:



As appearing in the hierarchy, every entity class will have instance variables to fetch its own properties and to fetch data of its immediate child entities through an API call.

Instance Objects

It is not always effective to follow the complete class hierarchy from the top to fetch the data of an entity at some lower level, since this would involve API calls at every level. In order to handle this, every entity class will have a **get_instance()** method to get its own dummy object and methods to get dummy objects of its child entities.

Note

get_instance() methods would not have any of its properties filled since it would not fire an API call. This would just return a dummy object that shall be only used to access the non-static methods of the class.



Summing it up,

- **ZCRMRestClient.get_module("Contacts")** would return the actual Contacts module, that has all the properties of the Contacts module filled through an API call.
- **ZCRMRestClient.get_module_instance("Contacts")** would return a dummy ZCRMModule object that would refer to the Contacts module, with no properties filled, since this doesn't make an API call.

Hence, to get records from a module, there's no need to start from ZCRMRestClient. Instead, you could get a ZCRMModule instance with ZCRMModule.get_instance() and then invoke its non-static get_records() method from the created instance. This would avoid the API call that would otherwise have been triggered to populate the ZCRMModule object.

Accessing record properties

Since record properties are dynamic across modules, we have only given the common fields like **created_time**, **created_by**, **owner etc**, as **ZCRMRecord's** default members. All other record properties are available as a map in **ZCRMRecord** object.

To access the individual field values of a record, use

- **record_instance.field_data[field_api_name] = "value"** to set the value of a field
- **variable_name = record_instance.field_data[field_api_name]** to retrieve the value of the field.

API Names

The keys of the record properties map are the API names of the module's fields. They are available in your **CRM under Setup** → **Developer Space** → **APIs** → **CRM API** → **API Names**.

Note

While setting a field value, make sure that the data types of the value and the field match.



Responses & Exceptions

APIResponse, **BulkAPIResponse** and **FileAPIResponse** are the wrapper objects for Zoho CRM APIs' responses. All methods that invoke the APIs return one of these three objects.

- A method-seeking entity would return **APIResponse** object, whereas a method-seeking list of entities would return **BulkAPIResponse** object.
- Use the instance variable "**data**" to get the entity data alone from the response wrapper objects. **APIResponse.data** returns a single Zoho CRM entity object, while **BulkAPIResponse.data** returns a list of Zoho CRM entity objects.
- **FileAPIResponse** will be returned for file download APIs to download a photo or an attachment from a record or note such as **record.download_photo()**, **record.download_attachment(attachment_id)** etc.

Other than data, these response wrappers have the following properties.

1. **ResponseInfo** - any other information, if provided by the API, besides the actual data. It is available through the member **response.info**.
2. **EntityResponse list** - status of the individual entities in a bulk API. For example, an insert records API may partially fail because of a few records. This array gives the individual records' creation status. It is available through **member bulk_entity_response: response.bulk_entity_response**

Start the App

The SDK requires the following line of code being invoked every time your app gets started.

```
ZCRMRestClient.initialize(configuration)
```



This method should be called from the main class of your Ruby application to start the application. It needs to be invoked without any exception.

Check Exceptions

All unexpected behaviors like faulty API responses, SDK anomalies are handled by the SDK and are thrown only as a single exception – **ZCRMException**. Hence, it's enough to catch this exception alone in the client app code.

Sample Codes

All of Zoho CRM's APIs can be used through the Ruby SDK, to enable your custom application perform data sync to the best degree. Here are the sample codes for all the API methods available in our SDK.

Rest Client Operations

These methods involve authentications procedures that are to be included in your application, to provide access to Zoho CRM's data.

Methods	Description
get_organization_details	To fetch information about your CRM account organization.
get_all_modules	To fetch information about all modules in



	your CRM account.
get_module	To fetch information about a particular module in your CRM account.
get_current_user	To fetch information about the user who is currently accessing Zoho CRM's data through your application.

Organization Operations

These methods involve actions that can be performed in your application, to modify the data that pertains to your Zoho CRM's organization. For instance, you can get the list of all the users (employees) that are present in your organization at any point of time.

Methods	Description
get_all_roles	To fetch information about a particular role in your CRM account.
get_role	To fetch information about a particular role in your CRM account.
get_profiles	To fetch the list of all the profiles that were created in your CRM account.
get_profile	To fetch information about a particular profile in your CRM account.
	To fetch the list of all the users from your



get_all_users	CRM account.
get_all_active_users	To fetch the list of all the active users in your CRM account.
get_all_deactive_users	To fetch the list of all the non-active users in your CRM account.
get_all_confirmed_users	To fetch the list of all the confirmed users in your CRM account.
get_all_not_confirmed_users	To fetch the list of all the users who are not confirmed in your CRM account.
get_all_deleted_users	To fetch the list of all the users who were deleted from your CRM account.
get_all_active_confirmed_users	To fetch the list of all the active and confirmed users in your CRM account.
get_all_admin_users	To fetch the list of all the users who have admin level permissions in your CRM account.
get_all_active_confirmed_admin_users	To fetch the list of all the active and confirmed users who have admin level permissions in your CRM account.
get_user	To fetch information about a specific user in your CRM account.
	To create a new user in your CRM



create_user	account.
update_user	To update details of an existing user in your CRM account.
delete_user	To delete a user from your CRM account.
get_organization_taxes	To fetch the list of all organization taxes.
get_organization_tax	To fetch information about a particular organization tax.
create_organization_taxes	To create a new organization tax.
update_organization_taxes	To update the details of a particular organization tax.
delete_organization_taxes	To delete all the taxes associated to the organization.
delete_organization_tax	To delete a particular tax associated to the organization.
search_user_by_criteria	To search for records in a module based on a criteria specified by the user

Module Operations

These methods involve actions that can be performed in your application, to modify the data in your CRM at the module level. For instance, you can get all the records from a module, search for specific ones, delete them, and do more.

Methods	Description
get_records_from_custom_view	To fetch the records that belong to a particular custom view in a module.
get_all_fields	To fetch the list of all the fields that are available in a module.
get_field	To fetch information about a particular field available in a module.
get_layout	To fetch information about a particular layout of a module.
get_all_layouts	To fetch the list of all the layouts that are available for a module.
get_customview	To fetch information about a particular custom view of a module.
get_all_customviews	To fetch the list of all the custom views that are available for a module.
update_custom_view	To update a custom view of a module.
get_all_relatedlists	To fetch the list of all the related lists that are available for a module.
get_relatedlist	To fetch the list of all the related lists that are available for a module.
	To fetch the list of all the records that are



get_records	available in a module.
get_record	To fetch information about a particular record in a module.
get_deleted_records	To fetch the list of all the records that were deleted from a module.
get_recyclebin_records	To fetch the list of all the records that were deleted from a module and stored in the recycle bin.
get_permanently_deleted_records	To fetch the list of all the records that were permanently deleted from a module.
get_tags	To fetch the list of all the tags that were created for a module.
get_tag_count	To fetch the list of all the tags that were created for a module.
create_tags	To create new tags for a module.
update_tags	To update details of existing tags for a module.
add_tags_to_multiple_records	To associate tags to records in a module.
remove_tags_from_multiple_records	To disassociate tags from records in a module.
	To search for records in a module based



search_records	on a Word(text).
search_records_by_phone	To search for records in a module based on the Phone number.
search_record_email	To search for records in a module based on Email address.
search_records_by_criteria	To search for records in a module based on a criteria specified by the user.
update_records	To update details of multiple records in a module.
create_records	To create a new record in a module.
delete_records	To delete existing records from a module.
upsert_records	To insert/update records in a module.
mass_update_records	To update single field value for multiple records. Same value will get updated for all the mentioned records

Record Operations

These methods involve actions that can be performed in your application, to access or modify data that are stored in a particular record. You could fetch the details of a record, create new ones, update existing ones, upload notes, attachments, photos, etc.

Methods	Description
---------	-------------



get_record	To fetch the details of a specific record.
create_record	To create a new record.
update_record	To update an existing record.
delete_record	To delete an existing record.
convert_record	To convert a record (Leads to Contacts/Deals).
get_attachments	To fetch the list of attachments of a record.
upload_attachment	To upload an attachment to a record.
upload_link_as_attachment	To upload a link as an attachment to a record.
download_attachment	To download an attachment that was uploaded to a record.
delete_attachment	To delete an attachment that was added to a record.
upload_photo	To upload a photo to a record.
download_photo	To download a photo that was added to a record.
	To delete a photo that was added to a



delete_photo	record.
add_relation	To make a relation between two records.
remove_relation	To remove a relation between two records.
get_related_records	To fetch the list of records in a related list

Note and Tag Operations

These methods involve actions that can be performed in your application, to access or modify notes or tags of data in your CRM.

Methods	Description
add_tags	To add tags to a record.
remove_tags	To remove tags from a record.
delete_tag	To delete a tag.
merge_tag	To merge two tags.
update_tag	To update details of an existing tag.
get_notes	To fetch the notes that were attached to a record.
add_note	To add a note to a record.



update_note	To update a note that was previously added to a record.
delete_note	To delete a note from a record.
upload_attachment_to_note	To upload an attachment to a note.
download_attachment_from_note	To download an attachment from a note.
get_attachments_from_note	To get the details of attachments from a note.
delete_attachment_of_note	To delete attachments of a note

Release Notes

Current Version

1. ZCRMSDK - VERSION 1.0.5

Install command

```
1 gem install ZCRMSDK -v 1.0.5
```

Notes

- Fixed the error when you retrieve Events using the `get_records` method and the participant(s) was invited only through an email.
- Incorrect reference in Participants, pricing detail, and product details is removed.

Previous Versions

2. ZCRMSDK - VERSION 1.0.4

Install command

```
1 gem install ZCRMSDK -v 1.0.4
```



Notes

- Changed the JSON dependency version(>=2.0).

3. ZCRMSDK - VERSION 1.0.3

Install command

```
1 gem install ZCRMSDK -v 1.0.3
```

Notes

- SDK throws exception if the accounts scope (aaaserver.profile.READ) is not included in the generated grant token or when the user's email cannot be fetched with the generated access token.
- Handled the module name check for the Product_Details, Pricing_Details, and Participants keys in GET records.
- Fixed current user bug.

4. ZCRMSDK - VERSION 1.0.2

Install command

```
1 gem install ZCRMSDK -v 1.0.2
```

Notes

- Fixed file persistence encoding.

5. ZCRMSDK - VERSION 1.0.1

Install command

```
1 gem install ZCRMSDK -v 1.0.1
```

Notes

- Handled the change in the OAuth token response.

6. ZCRMSDK - VERSION 1.0.0

Install command

```
1 gem install ZCRMSDK -v 1.0.0
```



Notes

- Initial release.