Python SDK Version 3

Zoho CRM
-zoho.com/crm-

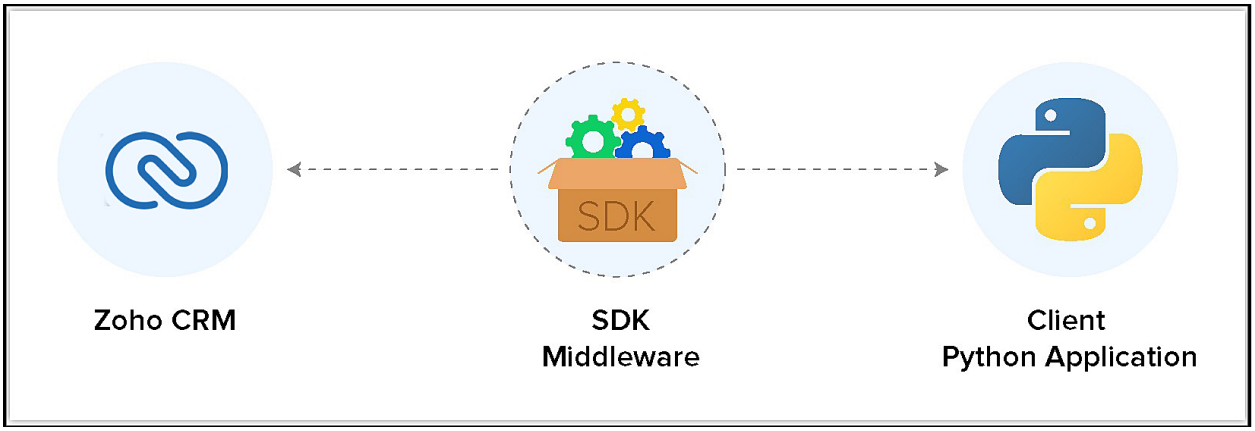
Table of Contents

Overview

Python SDK offers a way to create client python 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 Python application.



Python 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 member and methods in your Python application.
3. Push data into Zoho CRM, by accessing appropriate APIs of the CRM Service.

Environmental Setup

Python SDK is installable through "pip". Pip is a tool for dependency management in Python. The SDK requires the following from the client app:

- Client app must have Python (version 3 and above)
  - https://www.python.org/downloads/
- Python SDK must be installed through pip.

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.

Installation of SDK

The SDK can be installed for your client application through Pip. Pip is a tool for dependency management in Python.

Installing Pip

Please refer the document below to install pip

- https://pip.pypa.io/en/stable/installing/

Including the SDK in your project

You can include the SDK in your project using:

- Install Python from python.org (if not installed).
- Install **Python SDK**. Here's how you install the Python SDK
  Navigate to the workspace of your client app.
  Run the command below:

```
pip install zcrmsdk==3.x.x
```

The Python SDK will be installed in your client application.

Python SDK will be installed and a package named 'zcrmsdk' will be created in the installation directory of python (ex. '/Library/Python/3.7/site-packages').

Upgrade the SDK

- Run this command to upgrade the Python SDK to the latest version.

```
pip install --upgrade zcrmsdk==3.x.x
```

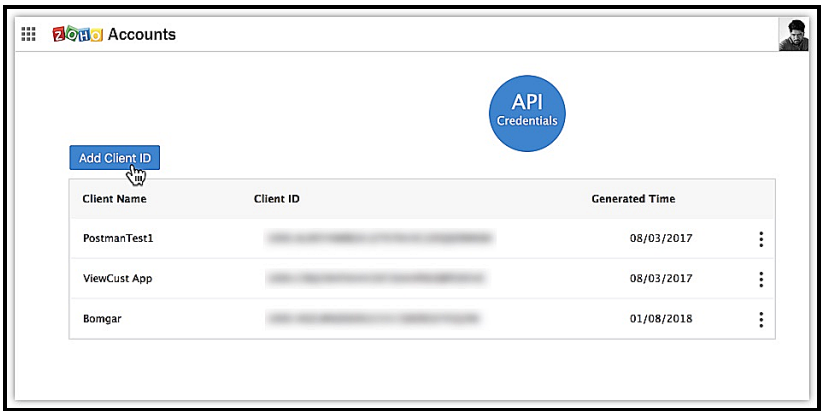
Register your application

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:

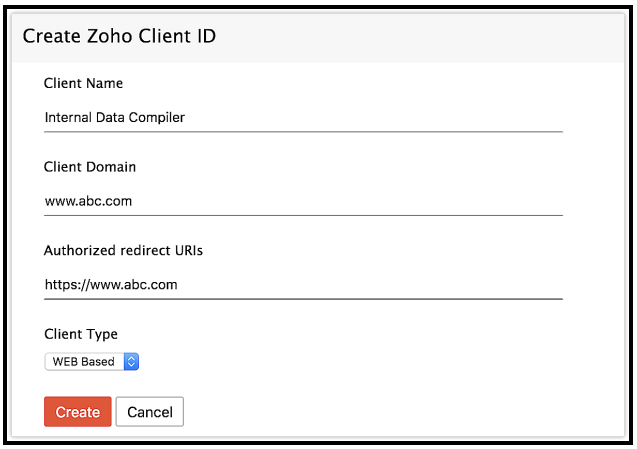
1. Go to the site: **https://api-console.zoho.com**

2. Click **Add Client ID**.



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 Python 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.

```
from zcrmsdk.src.com.zoho.api.logger import Logger
"""
Create an instance of Logger Class that takes two parameters
1 -> Level of the log messages to be logged. Can be
configured by typing Logger.Levels "." and choose any level
```

```
   from the list displayed.
5  2 -> Absolute file path, where messages need to be logged.
6  """
7  logger = Logger.get_instance(level=Logger.Levels.INFO,
   file_path="/Users/user_name/Documents/python_sdk_log.log")
```

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

```
1  from zcrmsdk.src.com.zoho.crm.api.user_signature import
   UserSignature
2  # Create an UserSignature instance that takes user Email as
   parameter
3  user = UserSignature(email="abc@zoho.com")
```

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

```
1  from zcrmsdk.src.com.zoho.crm.api.dc import USDataCenter
2  """
3  Configure the environment
4  which is of the pattern Domain.Environment
5  Available Domains: USDataCenter, EUDataCenter, INDataCenter,
   CNDataCenter, AUDataCenter
6  Available Environments: PRODUCTION(), DEVELOPER(), SANDBOX()
7  """
8  environment = USDataCenter.PRODUCTION()
```

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

```
1  from zcrmsdk.src.com.zoho.api.authenticator.oauth_token
   import OAuthToken, TokenType
2    """
3    Create a Token instance that takes the following parameters
4    1 -> OAuth client id.
5    2 -> OAuth client secret.
6    3 -> REFRESH/GRANT token.
```

```
    4 -> token type.
    5 -> OAuth redirect URL. Default value is None
    """
token = OAuthToken(client_id='clientId',
client_secret='clientSecret', token='REFRESH/ GRANT Token',
token_type=TokenType.REFRESH / TokenType.GRANT,
redirect_url='redirectURL')
```

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

```
from zcrmsdk.src.com.zoho.api.authenticator.store import
DBStore, FileStore
"""
DBStore takes the following parameters
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"
"""
store = DBStore()
#store = DBStore(host='host_name',
database_name='database_name', user_name='user_name',
password='password', port_number='port_number')
"""
FileStore takes the following parameter
1 -> Absolute file path of the file to persist tokens
"""
#store =
FileStore(file_path='/Users/username/Documents/python_sdk_tok
ens.txt')
```

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

```
from zcrmsdk.src.com.zoho.crm.api.sdk_config import SDKConfig
  """
```

```
  auto_refresh_fields
    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(zcrmsdk/src/com/zoho/crm/api/util/module_fields_handler.py)
  pick_list_validation
    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.
    if True - the SDK validates the input. If the value does not exist in the pick list, the SDK throws an error.
    if False - the SDK does not validate the input and makes the API request with the user's input to the pick list
  """
config = SDKConfig(auto_refresh_fields=True, pick_list_validation=False)
```

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

```
resource_path = "/Users/user_name/Documents/python-app";
```

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

```
from zcrmsdk.src.com.zoho.crm.api.request_proxy import RequestProxy
"""
RequestProxy takes the following parameters
1 -> Host
2 -> Port Number
3 -> User Name. Default value is None
4 -> Password. Default value is an empty string
"""
request_proxy = RequestProxy(host='proxyHost', port=80)
```

```
10 request_proxy = RequestProxy(host='proxyHost', port=80,
   user='userName', password='password')
```

9. Initialize the SDK and make API calls.

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.

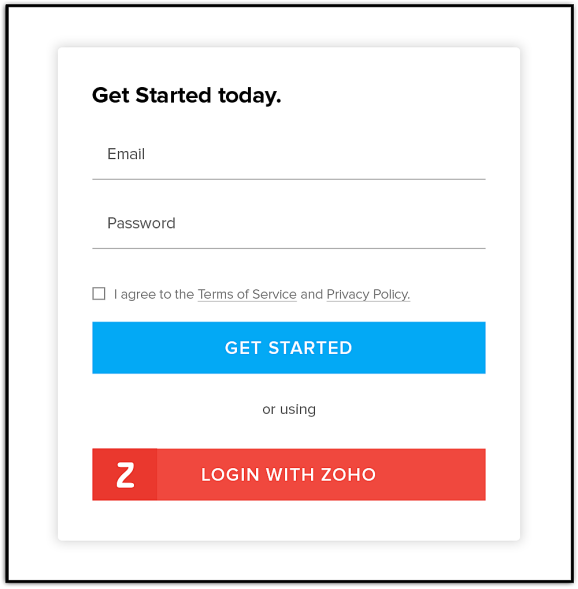
Note

- 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.

- **UserSignature** - The email ID of the user that is making the API calls. The tokens are also specific to this user.
- **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.

- **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.
- **Tokenstore** - The token persistence method. The possible methods are DB persistence, File persistence and Custom 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. For Custom persistence, you must provide the instance of the class that implements TokenStore (Abstract Base Class) and overrides the methods
- **Logger** - To log the messages. You can choose the level of logging of messages through Logger.Levels, and provide the absolute file path to the file where you want the SDK to write the messages in.
- **SDKConfig** - The class that contains the values of autoRefresh and pickListValidation fields.
- **resourcePath** - The absolute directory path to store user-specific files containing information about the fields of a module.
- **RequestProxy** - An instance containing the proxy details of the request.

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

Initialize the SDK using the following code.

```
from zcrmsdk.src.com.zoho.crm.api.user_signature import
UserSignature
from zcrmsdk.src.com.zoho.crm.api.dc import USDataCenter
from zcrmsdk.src.com.zoho.api.authenticator.store import
DBStore, FileStore
from zcrmsdk.src.com.zoho.api.logger import Logger
from zcrmsdk.src.com.zoho.crm.api.initializer import
```

```
Initializer
from zcrmsdk.src.com.zoho.api.authenticator.oauth_token import OAuthToken, TokenType
from zcrmsdk.src.com.zoho.crm.api.sdk_config import SDKConfig
from zcrmsdk.src.com.zoho.crm.api.request_proxy import RequestProxy

class SDKInitializer(object):

    @staticmethod
    def initialize():

        """
        Create an instance of Logger Class that takes two parameters
        1 -> Level of the log messages to be logged. Can be configured by typing Logger.Levels "." and choose any level from the list displayed.
        2 -> Absolute file path, where messages need to be logged.
        """
        logger = Logger.get_instance(level=Logger.Levels.INFO, file_path='/Users/user_name/Documents/python_sdk_log.log')

        # Create an UserSignature instance that takes user Email as parameter
        user = UserSignature(email='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()
        """
```

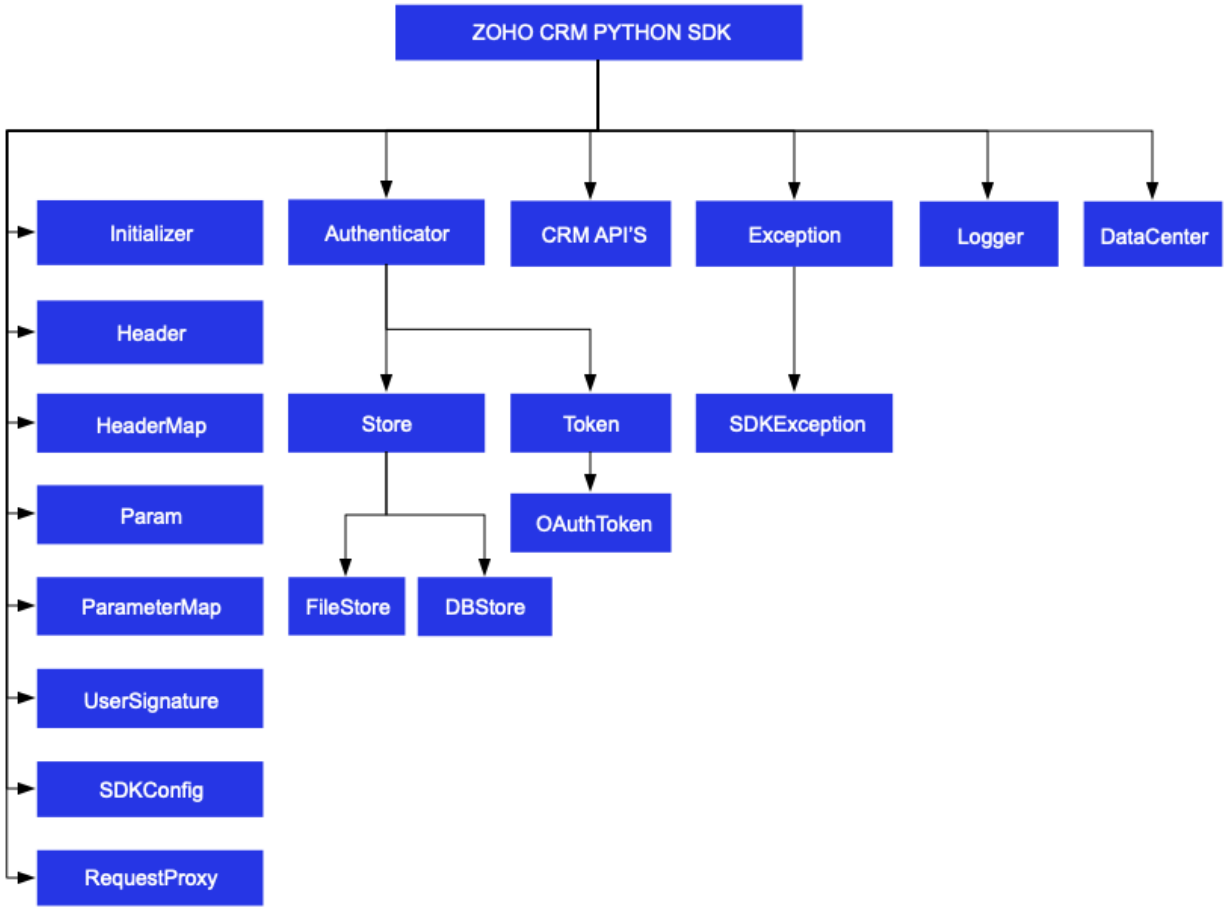
```
31          environment = USDataCenter.PRODUCTION()
32
33          """
34          Create a Token instance that takes the following
    parameters
35          1 -> OAuth client id.
36          2 -> OAuth client secret.
37          3 -> REFRESH/GRANT token.
38          4 -> token type.
39          5 -> OAuth redirect URL.
40          """
41          token = OAuthToken(client_id='clientId',
    client_secret='clientSecret', token='REFRESH/ GRANT Token',
    token_type=TokenType.REFRESH / TokenType.GRANT,
    redirect_url='redirectURL')
42
43          """
44          Create an instance of TokenStore
45          1 -> Absolute file path of the file to persist tokens
46          """
47          store =
    FileStore(file_path='/Users/username/Documents/python_sdk_tok
    ens.txt')
48
49          """
50          Create an instance of TokenStore
51          1 -> DataBase host name. Default value "localhost"
52          2 -> DataBase name. Default value "zohooauth"
53          3 -> DataBase user name. Default value "root"
54          4 -> DataBase password. Default value ""
55          5 -> DataBase port number. Default value "3306"
56          """
57          store = DBStore()
58          store = DBStore(host='host_name',
    database_name='database_name', user_name='user_name',
    password='password',port_number='port_number')
```

```
59
60          """
61          auto_refresh_fields
62              if True - all the modules' fields will be auto-refreshed in the background, every hour.
63              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(zcrmsdk/src/com/zoho/crm/api/util/module_fields_handler.py)
64
65          pick_list_validation
66              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.
67              if True - the SDK validates the input. If the value does not exist in the pick list, the SDK throws an error.
68              if False - the SDK does not validate the input and makes the API request with the user's input to the pick list
69          """
70          config = SDKConfig(auto_refresh_fields=True, pick_list_validation=False)
71
72          """
73          The path containing the absolute directory path (in the key resource_path) to store user-specific files containing information about fields in modules.
74          """
75          resource_path = '/Users/user_name/Documents/python-app'
76
77          """
78          Create an instance of RequestProxy class that takes the following parameters
```

```
        1 -> Host
        2 -> Port Number
        3 -> User Name. Default value is None
        4 -> Password. Default value is None
        """
        request_proxy = RequestProxy(host='proxyHost', port=8080)

        request_proxy = RequestProxy(host='proxyHost', port=8080, user='userName', password='password')

        """
        Call the static 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 -> resource_path
        7 -> Logger instance. Default value is None
        8 -> RequestProxy instance. Default value is None
        """
        Initializer.initialize(user=user, environment=environment, token=token, store=store, sdk_config=config, resource_path=resource_path, logger=logger, proxy=request_proxy)

 SDKInitializer.initialize()
```

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 Python SDK is depicted in the following image.



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 Abstract Base Class, which has the following callback methods.

- **get_token(self, user, token)** - invoked before firing a request to fetch the saved tokens. This method should return implementation of Token Abstract Base Class object for the library to process it.
- **save_token(self, user, token)** - invoked after fetching access and refresh tokens from Zoho.
- **delete_token(self, token)** - invoked before saving the latest tokens.
- **get_tokens(self)** - The method to retrieve all the stored tokens.
- **delete_tokens(self)** - The method to delete all the stored tokens.

Note:

- User is an instance of UserSignature class.
- Token is an instance of class that implements Token Abstract Base Class.

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

Note

- The Database persistence requires the following libraries :
  - mysql-connector
  - mysql-connector-python

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:

```
/*
from zcrmsdk.src.com.zoho.api.authenticator.store import DBStore
"""
DBStore takes the following parameters
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"
"""
store = DBStore()

store = DBStore(host='host_name', database_name='database_name', user_name='user_name', password='password', port_number='port_number')
```

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:

```
from zcrmsdk.src.com.zoho.api.authenticator.store import FileStore
"""
FileStore takes the following parameter
1 -> Absolute file path of the file to persist tokens
"""
store = FileStore(file_path='/Users/username/Documents/python_sdk_tokens.txt')
```

Custom Persistence

To use Custom Persistence, you must implement the Abstract Base Class TokenStore (zcrmsdk/src/com/zoho/api/authenticator/store/token_store) and override the methods.
Here is the code:

```
namespace store;
from zcrmsdk.src.com.zoho.api.authenticator.store import TokenStore
class CustomStore(TokenStore):
    def __init__(self):
        pass

    def get_token(self, user, token):

```

```
9           """
10          Parameters:
11              user (UserSignature) : A UserSignature class
    instance.
12              token (Token) : A Token
    (zcrmsdk.src.com.zoho.api.authenticator.OAuthToken) class
    instance
13          """
14
15          # Add code to get the token
16          return None
17
18      def save_token(self, user, token):
19
20          """
21          Parameters:
22              user (UserSignature) : A UserSignature class
    instance.
23              token (Token) : A Token
    (zcrmsdk.src.com.zoho.api.authenticator.OAuthToken) class
    instance
24          """
25
26          # Add code to save the token
27
28      def delete_token(self, token):
29
30          """
31          Parameters:
32              token (Token) : A Token
    (zcrmsdk.src.com.zoho.api.authenticator.OAuthToken) class
    instance
33          """
34
35          # Add code to delete the token
36      def get_tokens(self):
37
38          """
39          Returns:
40              list: List of stored tokens
```

```
41          """
42
43          # Add code to get the all stored tokens
44      def delete_tokens(self):
45          # Add code to delete the all stored tokens
```

Responses and Exceptions

All SDK methods return an instance of the APIResponse class.

After a successful API request, the **get_object()** method returns an instance of the **ResponseWrapper (for GET)** or the **ActionWrapper** (for **POST, PUT, DELETE**).

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

ResponseWrapper (for GET requests) and ActionWrapper (for POST, PUT, DELETE requests) are the expected objects for Zoho CRM API's responses.

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

- Operations involving records in Tags:
  -RecordActionWrapper
- Getting Record Count for a specific Tag operation:
  -CountWrapper
- Operations involving BaseCurrency
  -BaseCurrencyActionWrapper
- Lead convert operation
  -ConvertActionWrapper
- Retrieving Deleted records operation:
  -DeletedRecordsWrapper
- Record image download operation
  -FileBodyWrapper
- MassUpdate record operations
  -MassUpdateActionWrapper
  -MassUpdateResponseWrapper

For GET Requests

- The get_object() returns an instance of one of the following classes, based on the return type.
    - For application/json responses
      -ResponseWrapper
      -CountWrapper
      -DeletedRecordsWrapper
      -MassUpdateResponseWrapper
      -APIException
    - For file download responses
      -FileBodyWrapper
      -APIException

For POST, PUT, DELETE Requests

- The get_object() returns an instance of one of the following classes, based on the return type.
    - For application/json responses
      -ActionWrapper
      -RecordActionWrapper
      -BaseCurrencyActionWrapper
      -MassUpdateActionWrapper
      -ConvertActionWrapper
      -APIException

These wrapper classes may contain one or a list of instances of the following classes, depending on the response.

- **SuccessResponse** Class, if the request is successful.
- **APIException** Class, if the request is erroneous.

For example, when you insert two records, and one of them was inserted successfully while the other one failed, the ActionWrapper will contain one instance each of the SuccessResponse and APIException classes.

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

Threading in Python SDK

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

The Python SDK (from version 3.x.x) supports both single user and a multi-user app.

Multithreading in a Multi-user App

Multi-users functionality is achieved using Initializer's static switch_user() method.

```
# without proxy
Initializer.switch_user(user=user, environment=environment, token=token, sdk_config=sdk_config_instance)
# with proxy
Initializer.switch_user(user=user, environment=environment, token=token, sdk_config=sdk_config_instance, proxy=request_proxy)
```

Here is a sample code to depict multi-threading for a multi-user app.

```
import threading
from zcrmsdk.src.com.zoho.crm.api.user_signature import UserSignature
from zcrmsdk.src.com.zoho.crm.api.dc import USDataCenter, EUDataCenter
from zcrmsdk.src.com.zoho.api.authenticator.store import DBStore
from zcrmsdk.src.com.zoho.api.logger import Logger
from zcrmsdk.src.com.zoho.crm.api.initializer import Initializer
from zcrmsdk.src.com.zoho.api.authenticator.oauth_token import OAuthToken, TokenType
from zcrmsdk.src.com.zoho.crm.api.record import *
from zcrmsdk.src.com.zoho.crm.api.request_proxy import RequestProxy
from zcrmsdk.src.com.zoho.crm.api.sdk_config import SDKConfig


class MultiThread(threading.Thread):
    def __init__(self, environment, token, user, module_api_name, sdk_config, proxy=None):
        super().__init__()
```

```
        self.environment = environment
        self.token = token
        self.user = user
        self.module_api_name = module_api_name
        self.sdk_config = sdk_config
        self.proxy = proxy

    def run(self):
        try:
            Initializer.switch_user(user=self.user,
environment=self.environment, token=self.token,
sdk_config=self.sdk_config, proxy=self.proxy)

            print('Getting records for User: ' +
Initializer.get_initializer().user.email)

            response =
RecordOperations().get_records(self.module_api_name)

            if response is not None:

                # Get the status code from response
                print('Status Code: ' +
str(response.get_status_code()))

                if response.get_status_code() in [204, 304]:
                    print('No Content' if
response.get_status_code() == 204 else 'Not Modified')
                    return

                # Get object from response
                response_object = response.get_object()

                if response_object is not None:

                    # Check if expected ResponseWrapper instance
is received.
                    if isinstance(response_object,
ResponseWrapper):
                        # Get the list of obtained Record
```

```
    instances
48                          record_list = response_object.get_data()
49
50                          for record in record_list:
51                              for key, value in
    record.get_key_values().items():
52                                  print(key + " : " + str(value))
53
54                      # Check if the request returned an exception
55                      elif isinstance(response_object,
    APIException):
56                          # Get the Status
57                          print("Status: " +
    response_object.get_status().get_value())
58
59                          # Get the Code
60                          print("Code: " +
    response_object.get_code().get_value())
61
62                          print("Details")
63
64                          # Get the details dict
65                          details = response_object.get_details()
66
67                          for key, value in details.items():
68                              print(key + ' : ' + str(value))
69
70                          # Get the Message
71                          print("Message: " +
    response_object.get_message().get_value())
72
73          except Exception as e:
74              print(e)
75
76      @staticmethod
77      def call():
78          logger = Logger.get_instance(level=Logger.Levels.INFO,
    file_path="/Users/user_name/Documents/python_sdk_log.log")
79
80          user1 = UserSignature(email="abc@zoho.com")
```

```
81
82          token1 = OAuthToken(client_id="clientId1",
    client_secret="clientSecret1", token="GRANT Token",
    token_type=TokenType.GRANT)
83
84          environment1 = USDataCenter.PRODUCTION()
85
86          store = DBStore()
87
88          sdk_config_1 = SDKConfig(auto_refresh_fields=True,
    pick_list_validation=False)
89
90          resource_path = '/Users/user_name/Documents/python-app'
91
92          user1_module_api_name = 'Leads'
93
94          user2_module_api_name = 'Contacts'
95
96          environment2 = EUDataCenter.SANDBOX()
97
98          user2 = UserSignature(email="abc@zoho.eu")
99
100          sdk_config_2 = SDKConfig(auto_refresh_fields=False,
    pick_list_validation=True)
101
102          token2 = OAuthToken(client_id="clientId2",
    client_secret="clientSecret2",
103                              token="REFRESH Token",
    token_type=TokenType.REFRESH, redirect_url="redirectURL")
104
105          request_proxy_user_2 = RequestProxy(host="host",
    port=8080)
106
107          Initializer.initialize(user=user1,
    environment=environment1, token=token1, store=store,
    sdk_config=sdk_config_1, resource_path=resource_path,
    logger=logger)
108
109          t1 = MultiThread(environment1, token1, user1,
    user1_module_api_name, sdk_config_1)
```

```
110         t2 = MultiThread(environment2, token2, user2,
   user2_module_api_name, sdk_config_2, request_proxy_user_2)
111
112         t1.start()
113         t2.start()
114
115         t1.join()
116         t2.join()
```

1. MultiThread.call()The program execution starts from **call()**.
2. The details of **user1** is given in the variables **user1, token1, environment1**.
3. Similarly, the details of another user **user2** is given in the variables **user2, token2, environment2**.
4. When the **start()** is called which in-turn invokes the run(), the details of user1 are passed to the **switch_user** method through the **MultiThread object**. Therefore, this creates a thread for **user1**.
5. Similarly, When the **start()** is invoked again, the details of **user2** are passed to the **switch_user** function through the **MultiThread** object. Therefore, this creates a thread for **user2**.

Multi-threading in a Single User App

Here is a sample code to depict multi-threading for a single-user app.
import threading

```
1  from zcrmsdk.src.com.zoho.crm.api.user_signature import
   UserSignature
2  from zcrmsdk.src.com.zoho.crm.api.dc import USDataCenter
3  from zcrmsdk.src.com.zoho.api.authenticator.store import DBStore
4  from zcrmsdk.src.com.zoho.api.logger import Logger
5  from zcrmsdk.src.com.zoho.crm.api.initializer import Initializer
6  from zcrmsdk.src.com.zoho.api.authenticator.oauth_token import
   OAuthToken, TokenType
7  from zcrmsdk.src.com.zoho.crm.api.sdk_config import SDKConfig
8  from zcrmsdk.src.com.zoho.crm.api.record import *
9
10
11 class MultiThread(threading.Thread):
```

```
12
13      def __init__(self, module_api_name):
14          super().__init__()
15          self.module_api_name = module_api_name
16
17      def run(self):
18          try:
19              print("Calling Get Records for module: " +
    self.module_api_name)
20
21              response =
    RecordOperations().get_records(self.module_api_name)
22
23              if response is not None:
24
25                  # Get the status code from response
26                  print('Status Code: ' +
    str(response.get_status_code()))
27
28                  if response.get_status_code() in [204, 304]:
29                      print('No Content' if
    response.get_status_code() == 204 else 'Not Modified')
30                      return
31
32                  # Get object from response
33                  response_object = response.get_object()
34
35                  if response_object is not None:
36
37                      # Check if expected ResponseWrapper instance
    is received.
38                      if isinstance(response_object,
    ResponseWrapper):
39                          # Get the list of obtained Record
    instances
40                          record_list = response_object.get_data()
41
42                          for record in record_list:
43                              for key, value in
    record.get_key_values().items():
```

```
                                print(key + " : " + str(value))

                    # Check if the request returned an exception
                    elif isinstance(response_object, APIException):
                        # Get the Status
                        print("Status: " + response_object.get_status().get_value())

                        # Get the Code
                        print("Code: " + response_object.get_code().get_value())

                        print("Details")

                        # Get the details dict
                        details = response_object.get_details()

                        for key, value in details.items():
                            print(key + ' : ' + str(value))

                        # Get the Message
                        print("Message: " + response_object.get_message().get_value())

        except Exception as e:
            print(e)

    @staticmethod
    def call():
        logger = Logger.get_instance(level=Logger.Levels.INFO, file_path="/Users/user_name/Documents/python_sdk_log.log")

        user = UserSignature(email="abc@zoho.com")

        token = OAuthToken(client_id="clientId", client_secret="clientSecret",
                           token="GRANT Token", token_type=TokenType.GRANT, redirect_url="redirectURL")

```

```
        environment = USDataCenter.PRODUCTION()

        store = DBStore()

        sdk_config = SDKConfig()

        resource_path = '/Users/user_name/Documents/python-app'

        Initializer.initialize(user=user,
  environment=environment, token=token, store=store,
  sdk_config=sdk_config, resource_path=resource_path,
  logger=logger)

        t1 = MultiThread('Leads')
        t2 = MultiThread('Quotes')

        t1.start()
        t2.start()

        t1.join()
        t2.join()
MultiThread.call()
```

1. The program execution starts from **call()**, where the SDK is initialized with the details of the user.
2. When the **start()** is called which in-turn invokes the **run()**, the **module_api_name** is switched through the MultiThread object. Therefore, this creates a thread for the particular MultiThread instance.
3. Similarly, the details of another user **user2** is given in the variables **user2, token2, environment2**.

SDK Sample Code

```
from datetime import datetime
from zcrmsdk.src.com.zoho.crm.api.user_signature import UserSignature
from zcrmsdk.src.com.zoho.crm.api.dc import USDataCenter
from zcrmsdk.src.com.zoho.api.authenticator.store import DBStore
from zcrmsdk.src.com.zoho.api.logger import Logger
from zcrmsdk.src.com.zoho.crm.api.initializer import Initializer
from zcrmsdk.src.com.zoho.api.authenticator.oauth_token import OAuthToken, TokenType
from zcrmsdk.src.com.zoho.crm.api.record import *
from zcrmsdk.src.com.zoho.crm.api import HeaderMap, ParameterMap
from zcrmsdk.src.com.zoho.crm.api.sdk_config import SDKConfig

class Record(object):

    def __init__(self):
        pass

    @staticmethod
    def get_records():

        """
        Create an instance of Logger Class that takes two parameters
        1 -> Level of the log messages to be logged. Can be configured by typing Logger.Levels "." and choose any level from the list displayed.
        2 -> Absolute file path, where messages need to be logged.
        """
        logger = Logger.get_instance(level=Logger.Levels.INFO, file_path="/Users/user_name/Documents/python_sdk_log.log")

        # Create an UserSignature instance that takes user Email as parameter
        user = UserSignature(email="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 = USDataCenter.PRODUCTION()

        """
        Create a Token instance that takes the following
parameters
        1 -> OAuth client id.
        2 -> OAuth client secret.
        3 -> REFRESH/GRANT token.
        4 -> token type.
        5 -> OAuth redirect URL.
        """
        token = OAuthToken(client_id="clientId",
client_secret="clientSecret",
                           token="REFRESH/ GRANT Token",
token_type=TokenType.REFRESH / TokenType.GRANT,
redirect_url="redirectURL")

        """
        Create an instance of TokenStore
        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"
        """
        store = DBStore()

        """
        auto_refresh_fields
            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(zcrmsdk/src/com/zoho/crm/api/util/module_fiel
    ds_handler.py)
64
65          pick_list_validation
66              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.
67              if True - the SDK validates the input. If the value
    does not exist in the pick list, the SDK throws an error.
68              if False - the SDK does not validate the input and
    makes the API request with the user's input to the pick list
69          """
70          config = SDKConfig(auto_refresh_fields=True,
    pick_list_validation=False)
71
72          """
73          The path containing the absolute directory path (in the
    key resource_path) to store user-specific files containing
    information about fields in modules.
74          """
75          resource_path = '/Users/user_name/Documents/python-app'
76
77          """
78          Call the static initialize method of Initializer class
    that takes the following arguments
79          1 -> UserSignature instance
80          2 -> Environment instance
81          3 -> Token instance
82          4 -> TokenStore instance
83          5 -> SDKConfig instance
84          6 -> resource_path
85          7 -> Logger instance
86          """
87          Initializer.initialize(user=user,
    environment=environment, token=token, store=store,
    sdk_config=config, resource_path=resource_path, logger=logger)
88
89          try:
```

```
90              module_api_name = 'Leads'
91
92              param_instance = ParameterMap()
93
94              param_instance.add(GetRecordsParam.converted, 'both')
95
96              param_instance.add(GetRecordsParam.cvid,
   '12712717217218')
97
98              header_instance = HeaderMap()
99
100
   header_instance.add(GetRecordsHeader.if_modified_since,
   datetime.now())
101
102              response =
   RecordOperations().get_records(module_api_name, param_instance,
   header_instance)
103
104             if response is not None:
105
106                 # Get the status code from response
107                 print('Status Code: ' +
   str(response.get_status_code()))
108
109                 if response.get_status_code() in [204, 304]:
110                     print('No Content' if
   response.get_status_code() == 204 else 'Not Modified')
111                     return
112
113                 # Get object from response
114                 response_object = response.get_object()
115
116                 if response_object is not None:
117
118                     # Check if expected ResponseWrapper instance
   is received.
119                     if isinstance(response_object,
   ResponseWrapper):
120                         # Get the list of obtained Record
```

```
    instances
121                         record_list = response_object.get_data()
122
123                         for record in record_list:
124
125                             # Get the ID of each Record
126                             print("Record ID: " +
    record.get_id())
127
128                             # Get the createdBy User instance of
    each Record
129                             created_by = record.get_created_by()
130
131                             # Check if created_by is not None
132                             if created_by is not None:
133                                 # Get the Name of the created_by
    User
134                                 print("Record Created By - Name:
    " + created_by.get_name())
135
136                                 # Get the ID of the created_by
    User
137                                 print("Record Created By - ID: "
    + created_by.get_id())
138
139                                 # Get the Email of the
    created_by User
140                                 print("Record Created By -
    Email: " + created_by.get_email())
141
142                             # Get the CreatedTime of each Record
143                             print("Record CreatedTime: " +
    str(record.get_created_time()))
144
145                             if record.get_modified_time() is not
    None:
146                                 # Get the ModifiedTime of each
    Record
147                                 print("Record ModifiedTime: " +
    str(record.get_modified_time()))
```

```
148
149                                 # Get the modified_by User instance
   of each Record
150                                 modified_by =
   record.get_modified_by()
151
152                                 # Check if modified_by is not None
153                                 if modified_by is not None:
154                                     # Get the Name of the
   modified_by User
155                                     print("Record Modified By -
   Name: " + modified_by.get_name())
156
157                                     # Get the ID of the modified_by
   User
158                                     print("Record Modified By - ID:
   " + modified_by.get_id())
159
160                                     # Get the Email of the
   modified_by User
161                                     print("Record Modified By -
   Email: " + modified_by.get_email())
162
163                                 # Get the list of obtained Tag
   instance of each Record
164                                 tags = record.get_tag()
165
166                                 if tags is not None:
167                                     for tag in tags:
168                                         # Get the Name of each Tag
169                                         print("Record Tag Name: " +
   tag.get_name())
170
171                                         # Get the Id of each Tag
172                                         print("Record Tag ID: " +
   tag.get_id())
173
174                                 # To get particular field value
175                                 print("Record Field Value: " +
   str(record.get_key_value('Last_Name')))
```

```
176
177                             print('Record KeyValues: ')
178
179                             for key, value in
   record.get_key_values().items():
180                                 print(key + " : " + str(value))
181
182                     # Check if the request returned an exception
183                     elif isinstance(response_object,
   APIException):
184                         # Get the Status
185                         print("Status: " +
   response_object.get_status().get_value())
186
187                         # Get the Code
188                         print("Code: " +
   response_object.get_code().get_value())
189
190                         print("Details")
191
192                         # Get the details dict
193                         details = response_object.get_details()
194
195                         for key, value in details.items():
196                             print(key + ' : ' + str(value))
197
198                         # Get the Message
199                         print("Message: " +
   response_object.get_message().get_value())
200
201         except Exception as e:
202             print(e)
203
204 Record.get_records()
```

Sample Codes

All of Zoho CRM's APIs can be used through the Python 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(module_api_name, record_id)	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.
uploadLink_attachments	To upload a link as an attachment to a

	record.

Blueprint Operations

Constructor	Description
BluePrintOperations(record_id, module_api_name)	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_write_job_details	To know the status of the bulk write job scheduled previously.
download_result	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 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.
	To update the currencies' details of your

update_currencies	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
CustomViewsOperations(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
FieldsOperations(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
LayoutsOperations(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
ProfilesOperations(if_modified_since)	Creates a ProfilesOperations class 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

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

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
RelatedRecordsOperations(related_list_api_name, record_id, module_api_name)	Creates a RelatedRecordsOperations class instance with the relatedListAPIName, recordId, and

Constructor	Description
	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
ShareRecordsOperations(record_id, module_api_name)	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

Release Notes

Current Version:

1. ZCRMSDK - VERSION 3.1.0

- Install command

```
1  pip install zcrmsdk==3.1.0
```

Notes

- This version supports External ID.