JavaScript SDK
Version 1.x.x

Zoho CRM

--zoho.com/crm--

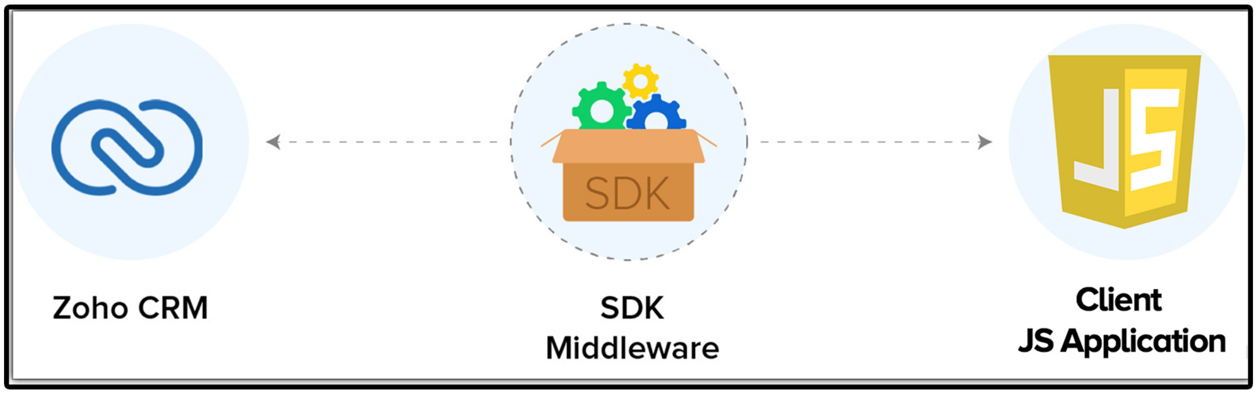
Table of contents

Overview

The JavaScript SDK offers a way to create client JavaScript 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 JS application.



Environmental Setup

You can install any browser as per your preference. JavaScript works on any web browser on any OS.

Get Our SDK

Download SDK

Note
It is mandatory for the client to have ZohoCRM.settings.fields.ALL to access all the record operations API. Otherwise, the system returns the OAUTH-SCOPE-MISMATCH error

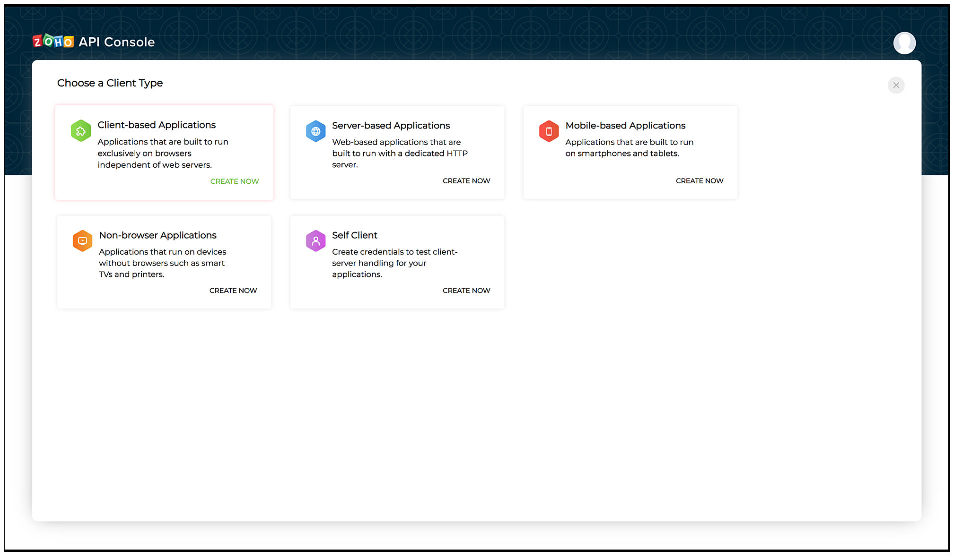
Register your Application

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

To register,

- Go to Zoho Developer Console.
- Choose a client type:
  - **Client-based**: Applications that are built to run exclusively on browsers independent of web servers.
  - **Server-based**: Web-based applications that are built to run with a dedicated HTTP server.
  - **Mobile**: Applications that are installed on smart phones and tablets.
  - **Non-browser Mobile Applications**: Applications for devices without browser provisioning such as smart TVs and printers.
  - **Self Client**: Stand-alone applications that perform only back-end jobs (without any manual intervention) like data sync.

For more details, refer to OAuth Overview.



Choose a Client Type

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

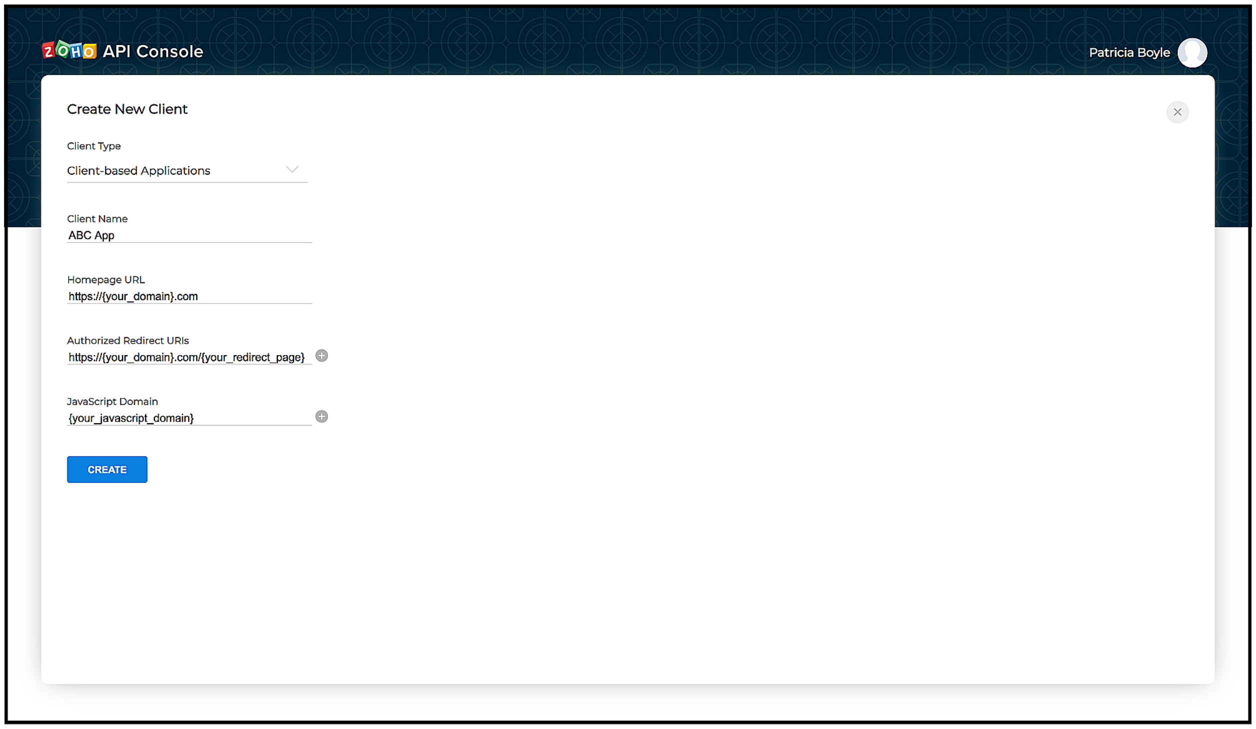
Server-based Applications
Web-based applications that are built to run with a dedicated HTTP server.
CREATE NOW

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

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

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

- Enter the following details:
    - **Client Name**: The name of your application you want to register with Zoho.
    - **Homepage URL**: The URL of your web page.
    - **Authorized Redirect URIs**: A valid URL of your application to which Zoho Accounts redirects you with a grant token(code) after successful authentication.



Create New Client

Client Type
Client-based Applications

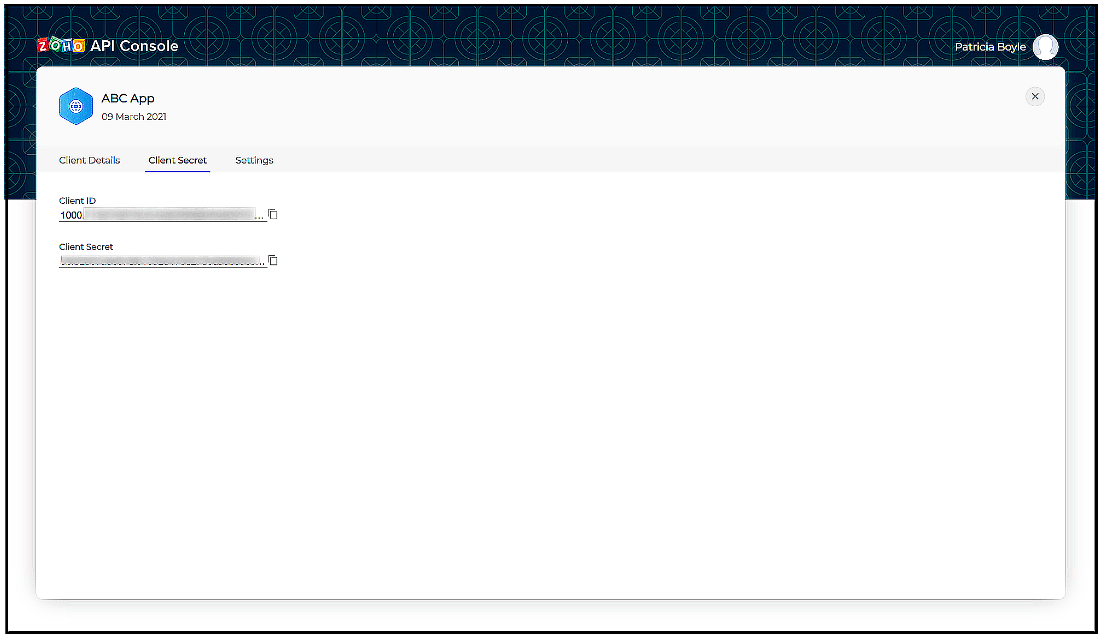
Client Name
ABC App

Homepage URL
https://{your_domain}.com

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

JavaScript Domain
{your_javascript_domain}

- Click **CREATE**.
- You will receive the following credentials:
    - **Client ID**: The consumer key generated from the connected app.
    - **Client Secret**: The consumer secret generated from the connected app.



Client ID
1000[redacted]

Client Secret
[redacted]

Note
If you don't have a domain name and a redirect URL, you can use dummy values in their place and register your client.

Using JS SDK in 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.

JavaScript SDK can be incorporated in two ways:
1. Integrating JS SDK via Webapps.
2. Using JS SDK on your own application.

1. **Integrating JS SDK via Webapps**
Follow the given steps for Integrating JS SDK:
- Register the client from CRM UI and note the client ID
- Create a new project using the command zet init via terminal/command line. Choose the option Catalyst and give the project name.
- A new folder will be created with the project name. Inside that, there will be a file

plugin_manifest.json. Update the client ID in that file and the required scopes to be used in the web app.

- Under the project folder, there will be another folder named app. This will act as the base.
- Include the zcrmsdk.js file (available in app folder) and use it in your HTML files.

a. **For Webapps Integration**:

- After the development, run the command zet pack from the project base folder and upload it in CRM UI.

> **Note** : Only one app can be uploaded for each client. While updating with the new app, the old one has to be deleted. Also, the redirect url will be changed.

- To know the redirect URL, Initializer.store.getToken(token) function has to be accessed from web app. It will redirect to accounts.zoho.com/oauth/v2/auth along with a parameter 'redirect_uri'. Configure it in https://api-console.zoho.com.

b. **To test it in local machine using web framework:**

- Create a redirect.html page within the app folder. The code for redirect.html has been provided below.
- Run it using the **zet run** via terminal/command line.>
- Enter **127.0.0.1:{your_port_number}** (for eg. 127.0.0.1:5000) in the browser's address bar and select the app_file.html
- It will redirect to **accounts.zoho.com/oauth/v2/auth** along with a **parameter redirect_uri**. Configure it in https://api-console.zoho.com/
- If the page successfully redirects to the redirect.html page then the app works as intended.
- Once the token is set for the first time, the page will be reloaded.

2. **Using JS SDK on your own application**.

- Create a redirect.html page for your application.
- Set up your own web server and authorize the SDK.
- You can then use the SDK in your own application.

Note:
The code for redirect.html

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Document</title>
    </head>
    <body></body>
    <script>
        function getPropertiesFromURL() {
            var props = {};

            var propertyString = window.location.hash || window.location.search;

            if( propertyString ) {

                propertyString = ( typeof propertyString === 'string' ) && propertyString.slice(1);

                if( propertyString ) {

                    propertyString
                        .split('&')
                        .forEach(function(prop) {

                            var key = prop.split('=')[0], value = prop.split('=')[1];

                            props[key] = value;
                        });
                }
            }

            return props;
        }

        function setAccessToken() {

            var hashProps = getPropertiesFromURL();

            if(hashProps) {

                for( var key in hashProps) {

                    if( hashProps.hasOwnProperty(key)) {

                        var value = ( key === 'api_domain' ) ? 
decodeURIComponent(hashProps[key]) : hashProps[key];

                        localStorage.setItem(key, value);
                    }
                }
            }

            setTimeout(function() { window.close(); }, 0);
        }
        setAccessToken();
    </script>
</html>
```

CDN Url : https://static.zohocdn.com/zohocrm/sdk/1.0.0/sdk.js

Configuration

Before you get started with creating your Javascript application, you need to register your client and authenticate the app with Zoho.

Follow the below steps to configure the SDK.

1. Create an instance of the **Logger** Class to log exception and API information.

```
/*
* Create an instance of Logger Class that takes parameter
* 1 -> Level of the log messages to be logged. Can be configured by typing Levels "." and choose any level from the list displayed.
*/
let logger = Logger.getInstance(Levels.ALL);
```

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

```
* Configure the environment
* which is of the pattern Domain.Environment
 * Available Domains: US, EU, IN, CN, AU
 * Available Environments: PRODUCTION(), DEVELOPER(), SANDBOX()
 */
let environment = DataCenter.US.PRODUCTION();
```

3. 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 redirect URL.
* 3 -> OAuth scope.
*/
let token = new OAuthToken("clientId", "redirectURL", "scope");
```

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

```
        /*
         * autoRefreshFields
         * if true - all the modules' fields will be auto-refreshed in the background, every hour.
         * if false - the fields will not be auto-refreshed in the background. The user can manually delete the cache or refresh the fields using methods from ModuleFieldsHandler
         *
         * cacheStore
         * A boolean field that allows or disallows the storage of module field information in cache.
         * True - the SDK stores all the modules' field information in cache, and refreshes every hour, if autoRefreshFields is true.
         * False - the SDK temporarily stores the modules' field information in a Map.
         *
         * if cacheStore true
         * pickListValidation
         * A boolean field that validates user input for a pick list field and allows or disallows the addition of a new value to the list.
         * True - the SDK validates the input. If the value does not exist in the pick list, the SDK throws an error.
         * False - the SDK does not validate the input and makes the API request with the user’s input to the pick list
         *
         * timeout
         * representing the number of milliseconds a request can take before automatically being terminated.
         */
let sdkConfig = new SDKConfigBuilder().setAutoRefreshFields(true).setPickListValidation(false).setCacheStore(true).timeout(1000).build();
```

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.

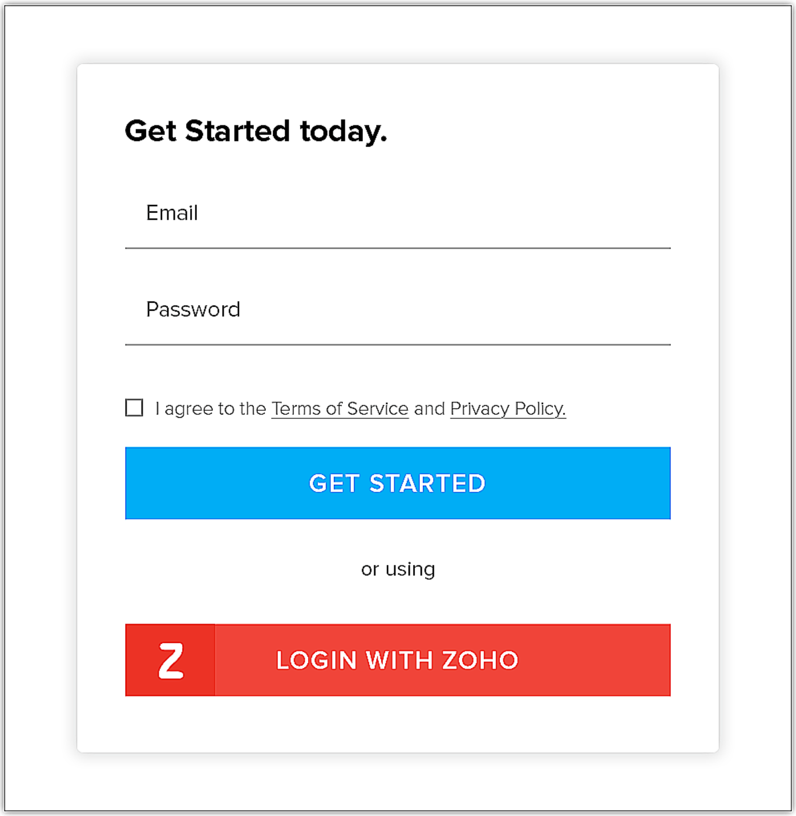
Note

- 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

Initialization

You must pass the following details to the SDK and initialize it before you can make API calls.

1. **environment** - The environment such as Production, Developer, or Sandbox from

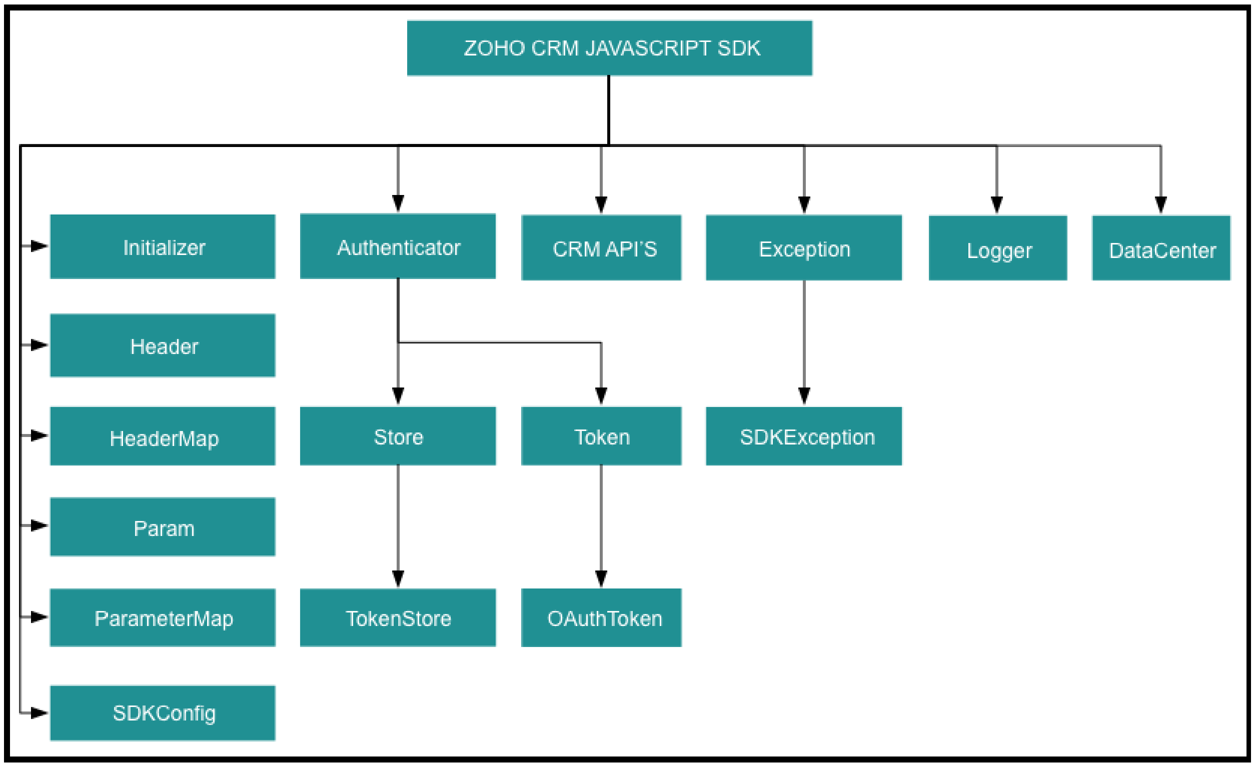
which you want to make API calls. This instance also takes the domain (data center) in which the tokens are generated. The format is *USDataCenter.PRODUCTION()*, *EUDataCenter.SANDBOX()* and so on.
2. **token** - The token must be specific to the user that makes the call, and specific to the org and the environment the token was generated in.
Besides the token, the token instance also takes the client ID, scope, and the redirect URI as its parameters.
3. **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.
4. **sdkConfig** - The class that contains the values of autoRefresh and pickListValidation fields.

Initialize the SDK using the following code.

```
class SDKInitializer{

    static async initializeSDK(){

        /*
         * Create an instance of Logger Class that takes parameter
         * 1 -> Level of the log messages to be logged. Can be configured by typing Levels "." and choose any level from the list displayed.
         */
        let logger = Logger.getInstance(Levels.ALL);

          /*
           * Configure the environment
           * which is of the pattern Domain.Environment
           * Available Domains: US, EU, IN, CN, AU
           * Available Environments: PRODUCTION(), DEVELOPER(), SANDBOX()
           */
          let environment = DataCenter.US.PRODUCTION();

          /*
           * Create a Token instance
           * 1 -> OAuth client id.
           * 2 -> OAuth redirect URL.
           * 3 -> OAuth scope.
           */
          let token = new OAuthToken("clientId", "redirectURL", "scope");


         /*
          * autoRefreshFields
          * if true - all the modules' fields will be auto-refreshed in the
background, every hour.
        * if false - the fields will not be auto-refreshed in the background. The user can manually delete the cache or refresh the fields using methods from ModuleFieldsHandler
        *
        * cacheStore
        * A boolean field that allows or disallows the storage of module field information in cache.
        * True - the SDK stores all the modules' field information in cache, and refreshes every hour, if autoRefreshFields is true.
        * False - the SDK temporarily stores the modules' field information in a Map.
        *
        * if cacheStore true
        * pickListValidation
        * A boolean field that validates user input for a pick list field and allows or disallows the addition of a new value to the list.
        * True - the SDK validates the input. If the value does not exist in the pick list, the SDK throws an error.
        * False - the SDK does not validate the input and makes the API request with the user’s input to the pick list
        *
        * timeout
        * representing the number of milliseconds a request can take before automatically being terminated.
        */
        let sdkConfig = new SDKConfigBuilder().setAutoRefreshFields(true).setPickListValidation(false).setCacheStore(true).build();

        /*
         * Call the static initialize method of Initializer class that takes the following arguments
         * 1 -> Environment instance
         * 2 -> SDKConfig instance
        * 3 -> Token instance
         * 4 -> Logger instance
         */
        await initializeSDK(environment, sdkConfig, token, logger);
    }
}
```

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 Javascript SDK is depicted in the following image.



Responses and Exceptions

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

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

Whenever the API returns an error response,the **getObject**() 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 getObject() returns an instance of one of the following classes, based on the return type.
- For application/json responses
- ResponseWrapper
- CountWrapper
- DeletedRecordsWrapper
- FileBodyWrapper
- MassUpdateResponseWrapper
- APIException

- For file download responses
  - FileBodyWrapper
  - APIException

For POST, PUT, DELETE Requests

- The getObject() 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 behaviors are thrown under the SDKException class.

SDK Sample code

```
class Record{

    static async call(){

        /*
         * Create an instance of Logger Class that takes parameter
         * 1 -> Level of the log messages to be logged. Can be configured by typing Levels "." and choose any level from the
```

```
list displayed.
         */
        let logger = Logger.getInstance(Levels.ALL);

        /*
         * Configure the environment
         * which is of the pattern Domain.Environment
         * Available Domains: US, EU, IN, CN, AU
         * Available Environments: PRODUCTION(), DEVELOPER(), SANDBOX()
         */
        let environment = DataCenter.US.PRODUCTION();

        /*
         * Create a Token instance
         * 1 -> OAuth client id.
         * 2 -> OAuth redirect URL.
         * 3 -> Oauth scope.
         */
        let token = new OAuthToken("clientId", "redirectURL", "scope");

/*
         * autoRefreshFields
         * if true - all the modules' fields will be auto-refreshed in the background, every hour.
         * if false - the fields will not be auto-refreshed in the background. The user can manually delete the cache or refresh the fields using methods from ModuleFieldsHandler
         *
         * pickListValidation
         * A boolean field that validates user input for a pick list field and allows or disallows the addition of a new value to the list.
         * True - the SDK validates the input. If the value does not exist in the pick list, the SDK throws an error.
         * False - the SDK does not validate the input and makes the API request with the user’s input to the pick list
         *
                * cacheStore
         * A boolean field that allows or disallows the storage
```

```
of module field information in cache.
         * True - the SDK stores all the modules' field information in cache, and refreshes every hour, if autoRefreshFields is true.
         * False - the SDK temporarily stores the modules' field information in a Map.
        * timeout - representing the number of milliseconds a request can take before automatically being terminated.
        */

        let sdkConfig = new SDKConfigBuilder().setAutoRefreshFields(true).setPickListValidation(false).setCacheStore(true).timeout(1000).build();

        /*
         * Call the static initialize method of Initializer class that takes the following arguments
         * 1 -> Environment instance
         * 2 -> SDKConfig instance
                * 3 -> Token instance
         * 4 -> Logger instance
         */
        await initializeSDK(environment, sdkConfig, token, logger);

        await Record.getRecords();
    }

    static async getRecords() {

        //Get instance of RecordOperations Class
        let recordOperations = new ZCRM.Record.Operations();

        //Get instance of ParameterMap Class
        let paramInstance = new ParameterMap();

        /* Possible parameters for Get Records operation*/
        await paramInstance.add(ZCRM.Record.Model.GetRecordsParam.APPROVED, "both");
        await
```

```
paramInstance.add(ZCRM.Record.Model.GetRecordsParam.CONVERTED, "both");

        await paramInstance.add(ZCRM.Record.Model.GetRecordsParam.SORT_BY, "Email");
        await paramInstance.add(ZCRM.Record.Model.GetRecordsParam.SORT_ORDER, "desc");
        await paramInstance.add(ZCRM.Record.Model.GetRecordsParam.PAGE, 1);
        await paramInstance.add(ZCRM.Record.Model.GetRecordsParam.PER_PAGE, 200);

        //Get instance of HeaderMap Class
        let headerInstance = new HeaderMap();

        /* Possible headers for Get Record operation*/
        await headerInstance.add(ZCRM.Record.Model.GetRecordsHeader.IF_MODIFIED_SINCE, new Date("2020-01-01T00:00:00+05:30"));

        //Call getRecords method that takes paramInstance, headerInstance and moduleAPIName as parameters
        let response = await recordOperations.getRecords("Leads");

        if(response != null) {

            //Get the status code from response
            console.log("Status Code: " + response.getStatusCode());

            if([204, 304].includes(response.getStatusCode())){
                console.log(response.getStatusCode() == 204? "No Content" : "Not Modified");

                return;
            }

            //Get the object from response
            let responseObject = response.getObject();
            if(responseObject != null){
                //Check if expected ResponseWrapper instance is received
                if(responseObject instanceof ZCRM.Record.Model.ResponseWrapper){
                    //Get the array of obtained Record instances
                    let records = responseObject.getData();
                    for (let index = 0; index < records.length; index++) {
                        let record = records[index];
                        //Get the ID of each Record
                        console.log("Record ID: " + record.getId());
                        //Get the createdBy User instance of each Record
                        let createdBy = record.getCreatedBy();
                        //Check if createdBy is not null
                        if(createdBy != null)
                        {
                            //Get the ID of the createdBy User
                            console.log("Record Created By User-ID: " + createdBy.getId());
                            //Get the name of the createdBy User
                            console.log("Record Created By User-Name: " + createdBy.getName());
                            //Get the Email of the createdBy User
                            console.log("Record Created By User-Email: " + createdBy.getEmail());
                        }
                        //Get the CreatedTime of each Record
                        console.log("Record CreatedTime: " + record.getCreatedTime());
                        //Get the modifiedBy User instance of each Record
                        let modifiedBy = record.getModifiedBy();
                        //Check if modifiedBy is not null
                        if(modifiedBy != null){
                            //Get the ID of the modifiedBy User
                            console.log("Record Modified By User-ID: " + modifiedBy.getId());
                            //Get the name of the modifiedBy User
                            console.log("Record Modified By User-Name: " + modifiedBy.getName());
                            //Get the Email of the modifiedBy User
                            console.log("Record Modified By User-Email: " + modifiedBy.getEmail());
                        }
                        //Get the ModifiedTime of each Record
                        console.log("Record ModifiedTime: " + record.getModifiedTime());
                        //Get the list of Tag instance each Record
                        let tags = record.getTag();
                        //Check if tags is not null
                        if(tags != null){
                            tags.forEach(tag => {
                                //Get the Name of each Tag
                                console.log("Record Tag Name: " + tag.getName());
                                //Get the Id of each Tag
                                console.log("Record Tag ID: " + tag.getId());
                            });
                        }
                        let keyValues = record.getKeyValues();
                        let keyArray = Array.from(keyValues.keys());
                        for (let keyIndex = 0; keyIndex < keyArray.length; keyIndex++) {
                            const keyName = keyArray[keyIndex];
                            let value = keyValues.get(keyName);

                            console.log(keyName + " : " + value);
```

```
                              }
                         }
                    }
               }
          }
     }
}
```

Sample Codes

All of Zoho CRM's APIs can be used through the Javascript 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(moduleApiName, recordId)	Creates an AttachmentsOperations class instance with the moduleAPIName and recordId.

Method	**Description**
getAttachments	To fetch the list of attachments of a record.
uploadAttachments	To upload attachments to a record.
deleteAttachments	To delete the attachments that were

	added to a record.
deleteAttachment	To delete an attachment that was added to a record.
downloadAttachment	To download an attachment that was uploaded to a record.
uploadLinkAttachments	To upload a link as an attachment to a record

Blueprint Operations

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

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

Bulk Read Operations

Method	**Description**
createBulkReadJob	To schedule a bulk read job to export records that match the criteria.
getBulkReadJobDetails	To know the status of the bulk read job scheduled previously.
downloadResult	To download the result of the bulk read job. The response contains a zip file. Extract it to get the CSV or ICS file depending on the "file_type" you specified while creating the bulk read job

Bulk Write Operations

Method	**Description**
uploadFile	To upload a CSV file in ZIP format. The response contains the "file_id". Use this ID while making the bulk write request.
createBulkWriteJob	To create a bulk write job to insert, update, or upsert records. The response contains the "job_id". Use this ID while getting the status of the scheduled bulk write job.
getBulkWriteJob_details	To know the status of the bulk write job scheduled previously.

downloadResult	To download the result of the bulk write job. The response contains a zip file. Extract it to get the CSV or ICS file depending on the "file_type" you specified while creating the write read job

Contact Roles Operations

Method	**Description**
getContactRoles	To get the list of all contact roles.
createContactRoles	To create contact roles.
updateContactRoles	To update contact roles.
deleteContactRoles	To delete contact roles.
getContactRole	To get specific contact role.
updateContactRole	To update specific contact role.
deleteContactRole	To delete specific contact role

Currencies Operations

Method	**Description**
getCurrencies	To get the list of all currencies available

	for your org.
addCurrencies	To add new currencies to your org.
updateCurrencies	To update the currencies' details of your org.
enableMultipleCurrencies	To enable multiple currencies for your org.
updateBaseCurrency	To update the base currency details of your org.
getCurrency	To get the details of specific currency.
updateCurrency	To update the details of specific currency

Custom View Operations

Constructor	**Description**
CustomViewsOperations(module)	Creates a CustomViewsOperations class instance with the moduleAPIName

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

getCustomView	in a module

Fields Metadata Operations

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

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

Files Operations

Method	**Description**
uploadFiles	To upload files and get their encrypted IDs.
getFile	To get the uploaded file through its encrypted ID

Layouts Operations

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

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

Modules Operations

Method	Description
getModules	To get the details of all the modules.
getModule	To get the details (metadata) of a specific module.
updateModuleByApiName	To update the details of a module by its module API name.
updateModuleById	To update the details of a module by its ID

Notes Operations

Method	Description
getNotes	To get the list of notes of a record.
createNotes	To add new notes to a record.
updateNotes	To update the details of the notes of a record.
deleteNotes	To delete the notes of a record.
getNote	To get the details of a specific note.
updateNote	To update the details of an existing note.
deleteNote	To delete a specific note

Notification Operations

Method	Description
enableNotifications	To enable instant notifications of actions performed on a module.
getNotificationDetails	To get the details of the notifications enabled by the user.
updateNotifications	To update the details of the notifications enabled by a user. All the provided details would be persisted and rest of the details

	would be removed.
updateNotification	To update only specific details of a specific notification enabled by the user. All the provided details would be persisted and rest of the details will not be removed.
disableNotifications	To stop all the instant notifications enabled by the user for a channel.
disableNotification	To disable notifications for the specified events in a channel

Organization Operations

Method	**Description**
getOrganization	To get the details of your organization.
uploadOrganizationPhoto	To upload a photo of your organization

Profile Operations

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

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

Query (COQL) Operation

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

Records Operations

Method	Description
getRecord	To get a specific record from a module.
updateRecord	To update a specific record in a module.
deleteRecord	To delete a specific record from a module.
getRecords	To get all records from a module.
createRecords	To insert records in a module.
updateRecords	To update records in a module.

deleteRecords	To delete records from a module.
upsertRecords	To insert/update records in a module.
getDeletedRecords	To get the deleted records from a module.
searchRecords	To search for records in a module that match certain criteria, email, phone number, or a word.
convertLead	To convert records(Leads to Contacts/Deals).
getPhoto	To get the photo of a record.
uploadPhoto	To upload a photo to a record.
deletePhoto	To delete the photo of a record.
massUpdateRecords	To update the same field for multiple records in a module.
getMassUpdateStatus	To get the status of the mass update job scheduled previously

Related List Operations

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

Method	Description
getRelatedLists	To get the details of all the related lists of a module.
getRelatedList	To get the details of a specific related list of a module

Related Records Operations

Constructor	Description
RelatedRecordsOperations(relatedListApiName, recordId, moduleApiName)	Creates a RelatedRecordsOperations class instance with the relatedListAPIName, recordId, and moduleAPIName

Method	Description
getRelatedRecords	To get list of records from the related list of a module.
updateRelatedRecords	To update the association/relation

	between the records.

Roles Operations

Method	**Description**
getRoles	To get the list of all roles available in your organization.

Shared Records Operations

Constructor	**Description**
ShareRecordsOperations(recordId, moduleApiName)	Creates a ShareRecordsOperations class instance with the recordId and moduleAPIName

Method	**Description**
getSharedRecordDetails	To get the details of a record shared with other users.
shareRecord	To share a record with other users in the organization.
updateSharePermissions	To • Update the sharing permissions of a record granted to users as Read-Write, Read-only, or grant full access.

	• Revoke access given to users to a shared record. • Update the access permission to the related lists of the record that was shared with the user.
revokeSharedRecord	To revoke access to a shared record

Tags Operations

Method	**Description**
getTags	To get the list of all tags in your organization.
createTags	To create tags.
updateTags	To update multiple tags.
updateTag	To update a specific tag.
deleteTag	To delete a specific tag from the module.
mergeTags	To merge two tags.
addTagsToRecord	To add tags to a specific record.
removeTagsFromRecord	To remove tags from a record.

addTagsToMultipleRecords	To add tags to multiple records.
removeTagsFromMultipleRecords	To remove tags from multiple records.
getRecordCountForTag	To get the record count for a tag.

Taxes Operations

Method	**Description**
getTaxes	To get the taxes of your organization.
createTaxes	To add taxes to your organization.
updateTaxes	To update the existing taxes of your organization.
deleteTaxes	To delete multiple taxes from your organization.
getTax	To get the details of a specific tax.
deleteTax	To delete a specific tax from your organization

Territory Operations

Method	**Description**

getTerritories	To get the list of all territories.
getTerritory	To get the details of a specific territory

Users Operations

Method	Description
getUsers	To get the list of users in your organization.
createUser	To add a user to your organization.
updateUsers	To update the existing users of your organization.
getUser	To get the details of a specific user.
updateUser	To update the details of a specific user.
deleteUser	To delete a specific user from your organization

Variable Groups Operations

Method	Description
getVariableGroups	To get the list of all variable groups available for your organization.

getVariableGroupById	To get the details of a variable group by its group ID.
getVariableGroupByApiName	To get the details of a specific variable group by its API name

Variables Operations

Method	**Description**
getVariables	To get the list of variables available for your organization.
createVariables	To add new variables to your organization.
updateVariables	To update the details of variables.
deleteVariables	To delete multiple variables.
getVariableById	To get the details of a specific variable by its unique ID.
updateVariableById	To update the details of a specific variable by its unique ID.
deleteVariable	To delete a specific variable.
getVariableForApiName	To get the details of a variable by its API name.

updateVariableByApiName	To update the details of a variable by its API name

Release Notes

Current Version

ZCRMSDK -VERSION 1.1.0

CD URL: https://static.zohocdn.com/zohocrm/sdk/1.1.0/sdk.js

Notes

- Supported Tag-CarryOver and External ID

Previous Versions

ZCRMSDK -VERSION 1.0.0

Install command: https://static.zohocdn.com/zohocrm/sdk/1.0.0/sdk.js

Notes

- Introducing Zoho CRM JavaScript SDK
- A new SDK that represents a significant effort to utilize the capabilities of JavaScript in managing your CRM data.
- The SDK is highly structured to ensure easy access to all the components.
- Each CRM entity is represented by an object, and each object 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.