Node JS
Version 0

Zoho CRM

-zoho.com/crm-

Table of Contents

Overview

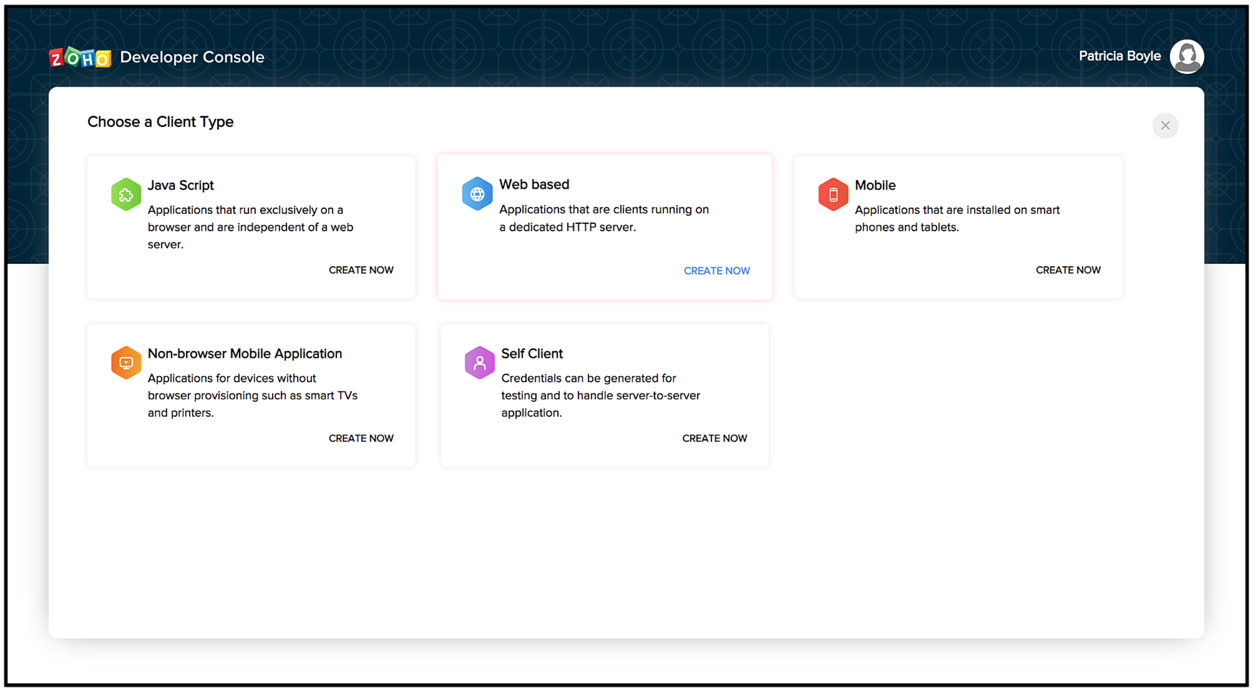
Node SDK is a wrapper for Zoho CRM APIs. Hence invoking a Zoho CRM API from your Node application is just a function call which provide the most appropriate response.

This SDK supports both single user as well as multi user authentication.

Registering a Zoho Client

Since Zoho CRM APIs are authenticated with OAuth2 standards, you should register your client app with Zoho. To register your app:

1. Go to Zoho Developer Console.
2. Click ADD CLIENT.
3. Choose the client as Web based and click CREATE NOW.



Developer Console

Patricia Boyle

Choose a Client Type

Java Script
Applications that run exclusively on a browser and are independent of a web server.
CREATE NOW

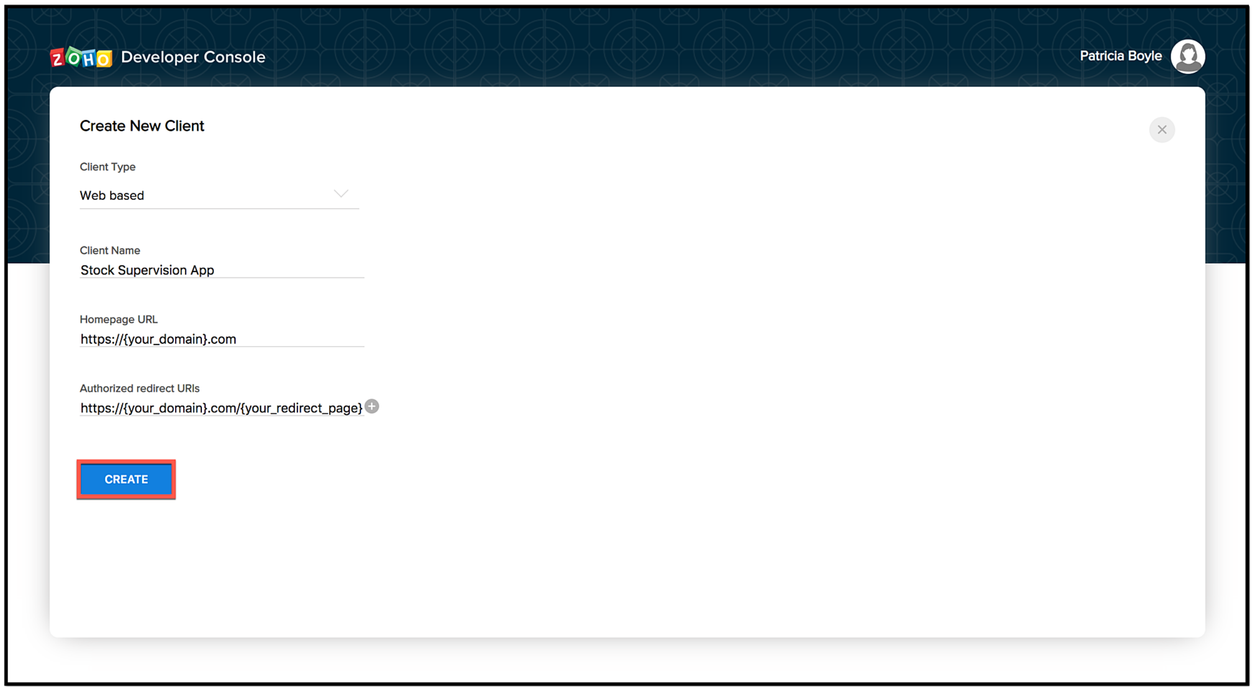
Web based
Applications that are clients running on a dedicated HTTP server.
CREATE NOW

Mobile
Applications that are installed on smart phones and tablets.
CREATE NOW

Non-browser Mobile Application
Applications for devices without browser provisioning such as smart TVs and printers.
CREATE NOW

Self Client
Credentials can be generated for testing and to handle server-to-server application.
CREATE NOW

4. Specify the client name, homepage URL of your application's UI, and a redirect URI to which you want to redirect the users after they grant consent to your application.



Developer Console

Patricia Boyle

Create New Client

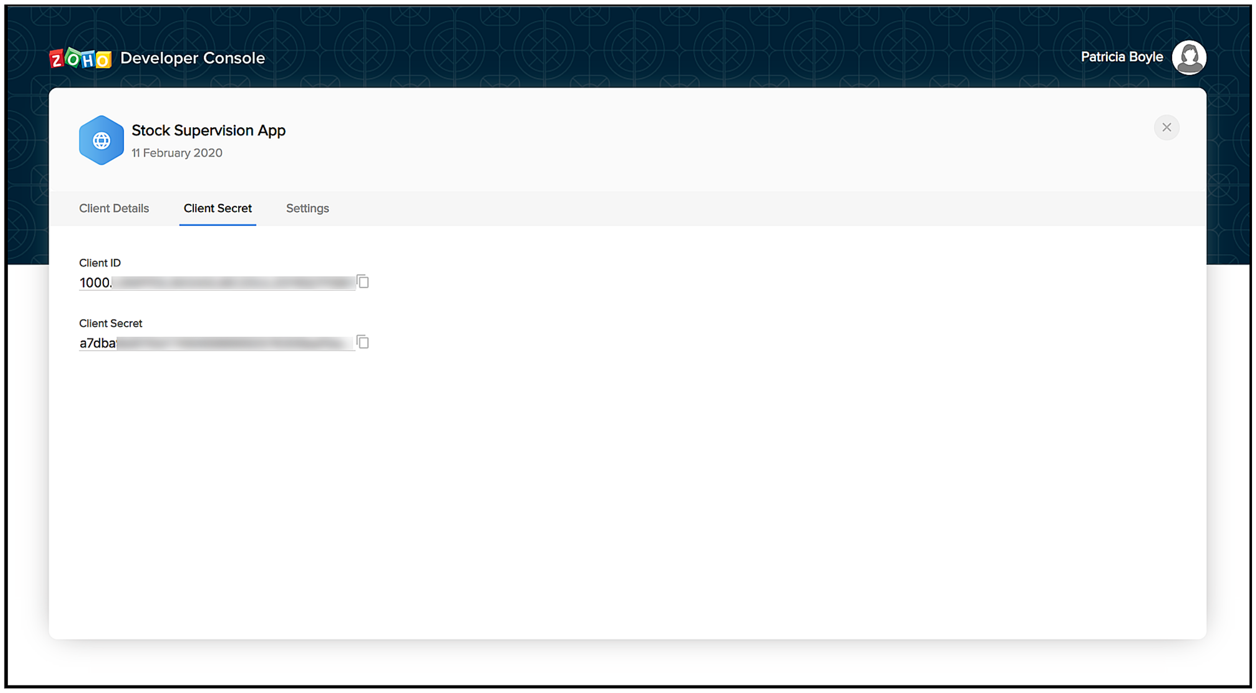
Client Type: Web based

Client Name: Stock Supervision App

Homepage URL: https://{your_domain}.com

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

5. Click Create.
6. Your Client ID and Client Secret will be displayed.



Developer Console

Patricia Boyle

Stock Supervision App

11 February 2020

Client ID: 1000.[redacted]

Client Secret: a7dba[redacted]

Installation of Node CRM SDK

Node JS SDK will be installed and a package named 'zcrmsdk' will be created in the local machine.

The package can be added using the following code:

```
var ZCRMRestClient = require('zcrmsdk')
```

Installing the SDK

Here's how you install the Node JS SDK

- Run the command below:

```
npm install zcrmsdk
```

Another method to install the SDK is to add it in dependencies to the **package.json** of the node server with the latest version (recommended) and to run npm install in the directory which installs all the dependencies mentioned in package.json.

Upgrade the SDK

- Run this command to upgrade the Node JS SDK to the latest version.

```
npm install --upgrade zcrmsdk
```

Configurations

Your OAuth Client details should be given to the SDK as a property file or must be passed while initializing the SDK as JSON key-value pairs. In the SDK, you need to configure a file named oauth_configuration.properties. Please place the respective values in that file. You can place it under resources folder from where the SDK is used.

Please fill the values for the following keys alone.

Based on your domain(EU,CN, IN, or AU), please change the value of crm.iamurl. Default value is set as US domain.

In **oauth_configuration.properties** file:

```
[zoho]
crm.iamurl=
crm.clientid=
crm.clientsecret=
crm.redirecturl=
```

- crm.clientid, crm.clientsecret and crm.redirecturl are your OAuth client's configurations that you get after registering your Zoho client.
- crm.iamurl is the accounts URL. It may be accounts.zoho.com or accounts.zoho.eu. If the crm.iamurl is not specified, by default the URL will be accounts.zoho.com.

In **configuration.properties** file:

```
[crm]
api.url=
api.user_identifier=
api.tokenmanagement=
[mysql]
username=
password=
```

- **api.url** is the URL used to call APIs. By default, the URL is www.zohoapis.com.
- **api.user_identifier** will be empty by default. For single user authentication, this key can be filled with the respective email id, so that all calls happens by using this user's authentication.
- **api.tokenmanagement** is given as a measure to manage and maintain tokens. If tokenmanagement is not provided, then the SDK's default implementation(mysql) will be followed.
- **username** and **password** can be given here if you already have one created for your MySQL.

The above keys specified in configuration.properties file are all optional.

Token Storage Mechanism

MySQL Persistence

To use the default token storage provided by the SDK, the following are to be done:

MySQL should be running at the **default port** in **localhost**.

Database with name **zohooauth** should be created and a table with below configurations should be present in the database. Table, named "**oauthtokens**", should have the columns "**useridentifier**" (varchar) "**accesstoken**" (varchar), "**refreshtoken**" (varchar) and "**expirytime**" (bigint).

Once the configuration is set, storage and retrieval of tokens will be handled by the SDK.

Custom Persistence

If the user wants to utilize their own mechanism, they can mention it in **configuration.properties** by providing the respective module in **api.tokenmanagement**.

This module should contain the below methods,

1. saveOAuthTokens(token_obj)
2. updateOAuthTokens(token_obj)
    - Irrespective of the response, the next execution happens. So care should be taken by the user in handling their module.
3. getOAuthTokens()
    - The expected response for this method : JSON array containing json response with expirytime, refreshtoken and accesstoken fields.

> Note:
> All methods should return promise.

saveOAuthtoken & **updateOAuthTokens** will be called with JSON parameters, which contain fields given below,

```
access_token
refresh_token
expires_in
```

Sample code for custom token management methods

saveOAuthTokens()

```
const fs = require('fs');
var path = require('path');
var file_persistence = {};
file_persistence.saveOAuthTokens = function(tokenobject){
 return new Promise(function(resolve,reject){
file_persistence.updateOAuthTokens(tokenobject).then(function(){
   resolve();
  })
 })
}
```

getOAuthTokens()

```
file_persistence.getOAuthTokens = function(user_identifier){
 return new Promise(function(resolve,reject){
  var found = 0;

  if (fs.existsSync(path.dirname(require.main.filename)+'/zcrm_oauthtokens.txt')){
  var token_as_string = fs.readFileSync(path.dirname(require.main.filename)+'/zcrm_oauthtokens.txt', 'utf8');
  tokens = JSON.parse(token_as_string);

  for(token in tokens.tokens){
   if(tokens.tokens[token].user_identifier == user_identifier){
    found = 1;
    resolve(tokens.tokens[token]);
   }
  }

  if(found == 0){
   resolve();
  }
  }
 })
}
```

updateOAuthTokens()

```
file_persistence.updateOAuthTokens = function(tokenobject){
 return new Promise(function(resolve,reject){
  var tokens = {}
  if (fs.existsSync(path.dirname(require.main.filename)+'/zcrm_oauthtokens.txt')){
  var token_as_string = fs.readFileSync(path.dirname(require.main.filename)+'/zcrm_oauthtokens.txt', 'utf8');
  tokens = JSON.parse(token_as_string);

  for(token in tokens.tokens){
   if(tokens.tokens[token].user_identifier == tokenobject.user_identifier){
    tokens.tokens.splice(token)
  }
 }
 tokens.tokens.push(tokenobject);
 }
```

```
  else{
   tokens.tokens = [tokenobject]
  }
 token_as_string = JSON.stringify(tokens);


fs.writeFile(path.dirname(require.main.filename)+'/zcrm_oauthtokens.txt', token_as_string, function (err) {
   if (err) throw err;
  resolve();
  });
 })
}

module.exports = file_persistence;
```

Custom persistence will create the token file named zcrm_oauthtokens.txt and this file will contain the below keys with their values.

```
{"tokens":
 [{"access_token":"1000.xxxxx",
 "expires_in":1582804396157,
 "user_identifier":"Patricia.boyle@zohocorp.com",
 "refresh_token":"xxx"}]}
```

Configuration array

The Node JS SDK also accepts the configuration details as a JSON array. You can pass this array when you initialize the SDK, as below (after the 'require' statement).

```
var configJson={
 "client_id":"{client_id}",//mandatory
 "client_secret":"{client_secret}",//mandatory
 "redirect_url":"{redirect_url}",//mandatory
 "user_identifier":"{user_identifier}",
 "mysql_username":"{mysql_username}",//optional ,"root" is default value
 "mysql_password":"{mysql_password}",//optional ,"" is default value
 "base_url":"{api_base_url}",//optional ,"www.zohoapis.com" is default value
 "iamurl":"{accounts_url}",//optional ,"accounts.zoho.com" is default value
 "version":"{api_version}",//optional ,"v2" is default value
 "tokenmanagement":"{token_management_module}"//optional ,mysql module is default
}
```

Initialization

Whenever the app is started, the below code snippet is to be called for initialization. Note that the snippet mentioned below is only when you specify the configuration details in the properties files.

```
var ZCRMRestClient = require('zcrmsdk');
ZCRMRestClient.initialize().then(function()//Use ZCRMRestClient.initialize(configJson).then(function() for configuration array
{
//do whatever required after initialize
})
```

Generating self-authorized grant and refresh token

For self client apps, the self authorized grant token should be generated from the Zoho

Developer Console (https://accounts.zoho.com/developerconsole)

1. Go to Zoho Developer Console.
2. Select your Self Client.
3. Provide the necessary scopes separated by commas, along with the scope aaaserver.profile.READ.
4. Select the *Time Duration* from the drop-down. This is the time the grant token is valid for.
5. Provide a description for the scopes. Click *GENERATE*.
6. Copy the grant token.

Please note that 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.

All functions return promises in zcrm node sdk.

Getting access and refresh tokens from grant token through method calls

```
ZCRMRestClient.generateAuthTokens(user_identifier,grant_token).then(function(auth_response){
console.log("access token :"+auth_response.access_token);
console.log("refresh token :"+auth_response.refresh_token);
console.log("expires in :"+auth_response.expires_in);
});
```

The access and refresh tokens are generated. In case the access token is expired, the SDK will refresh it automatically.

If the user has refresh token and need to generate access token below function can be used,

```
ZCRMRestClient.generateAuthTokenfromRefreshToken(user_identifier,refresh_token).then(function(auth_response){
```

```
console.log("access token :"+auth_response.access_token);
console.log("refresh token :"+auth_response.refresh_token);
console.log("expires in :"+auth_response.expires_in);
});
```

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.

Sample API call for getting Leads:

```
var input ={};
input.module = "Leads";
var params = {};
params.page = 0;
params.per_page = 5;
input.params = params;
zcrmsdk.API.MODULES.get(input).then(function(response){
    var result = "<html><body><b>Leads</b>";
    var data = response.body;
    data = JSON.parse(data);
    data = data.data;
    for (i in data){
        var record = data[i];
        var name = record.Full_Name;
        result+="<span>"+name+"</span>";
    }
    result+="</body></html>";
  })
```

Hierarchy

zcrmsdk

```
API
  ORG
    get
  MODULES
    get
    post
    put
    delete
    getAllDeletedRecords
    getRecycleBinRecords
    getPermanentlyDeletedRecords
    search
  SETTINGS
    getFields
    getLayouts
    getCustomViews
    updateCustomViews
    getModules
    getRoles
    getProfiles
    getRelatedLists
  ACTIONS
    convert
  USERS
    get
  ATTACHMENTS
```

```
      uploadFile
      deleteFile
      downloadFile
      uploadLink
      uploadPhoto
      downloadPhoto
      deletePhoto
    FUNCTIONS
      executeFunctionsInGet
      executeFunctionsInPost
```

As appearing in the hierarchy, **zcrmsdk** entity module has variables to fetch its own and other modules properties.

For example, to call an API to get module data, the request should be **zcrmsdk.API.MODULES.{operation_type}**. The operation types can be GET, POST, PUT, DELETE or CREATE.

Response Handling

All API calls will give the actual API response given by Zoho APIs, except file download.

For file download, the response will contain an extra field filename.

Error Handling

All errors will be thrown explicitly and care should be taken in catching the same.

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.

Organization & User Operations

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

Methods	**Description**
crmclient.API.ORG.get(params)	To fetch information about your CRM account organization.
crmclient.API.USERS.get(input)	To fetch information about all the users in your CRM account.
crmclient.API.USERS.get(input)	To fetch information about a particular user in your CRM account. [Specify the id of the user].

Settings Operations

These methods involve actions that can be performed in your application, to fetch information regarding the resources available in Zoho CRM. In other words, you can get the meta data of all the resources.

Methods	**Description**
crmclient.API.SETTINGS.getModules(input)	To fetch the list of all the modules available in your CRM account.
crmclient.API.SETTINGS.getModules(input)	To fetch information about a particular module in your CRM account. [Specify the name of the module]
crmclient.API.SETTINGS.getRoles(input)	To fetch the list of all the roles that were created in your CRM account.
crmclient.API.SETTINGS.getRoles(input)	To fetch information about a particular role in your CRM account. [Specify the id of the role]
crmclient.API.SETTINGS.getProfiles(input)	To fetch the list of all the profiles that are available in your CRM account.
crmclient.API.SETTINGS.getProfiles(input)	To fetch information about a particular role available in your CRM account. [Specify the id of the profile]
crmclient.API.SETTINGS.getFields(input)	To fetch the list of all the fields available in a module in your CRM account.
crmclient.API.SETTINGS.getLayouts(input)	To fetch the list of all the layouts of a module in your CRM account.
crmclient.API.SETTINGS.getLayouts(input)	To fetch information about a particular layout of a particular module in your

	CRM account. [Specify the id of the layout]
crmclient.API.SETTINGS.getRelatedLists(input)	To fetch the list of all the related lists of a particular module in your CRM account.
crmclient.API.SETTINGS.getCustomViews(input)	To fetch the list of all the custom views of a particular module in your CRM account.
crmclient.API.SETTINGS.getCustomViews(input)	To fetch information about a particular custom view of a particular module in your CRM account. [Specify the id of the custom view]

Modules Operations

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

Methods	**Description**
crmclient.API.MODULES.get(input)	To fetch the list of all the records that are present in a module.
crmclient.API.MODULES.get(input)	To fetch information about a particular record present in a module. [Specify the id of the record]

crmclient.API.MODULES.post(input)	To create records in a module.
crmclient.API.MODULES.put(input)	To update existing records in a module.
crmclient.API.MODULES.put(input)	To update a particular record in a module. [Specify the id of the record]
crmclient.API.MODULES.delete(input)	To delete a particular record in a module.
crmclient.API.MODULES.delete(input)	To delete records in a module.
crmclient.API.MODULES.getAllDeletedRecords(input)	To fetch the list of all the deleted records of a module.
crmclient.API.MODULES.getRecycleBinRecords(input)	To fetch the list of all the records that are deleted and are in the Recycle Bin.
crmclient.API.MODULES.getPermanentlyDeletedRecords(input)	To fetch the list of all the permanently deleted records of a module.
crmclient.API.MODULES.search(input)	To search for records in a module.

Actions & Attachments Operations

These methods involve actions that can be performed in your application, to perform certain actions such as converting a lead, uploading attachments, photos, etc.

Methods	**Description**
crmclient.API.ACTIONS.convert(input)	To convert a record(Leads to Contacts/Deals).
crmclient.API.ATTACHMENTS.uploadFile(input)	To upload files(attachments) to a record in a module.
crmclient.API.ATTACHMENTS.downloadFile(input)	To download files(attachments) that were attached to a record in a module.
crmclient.API.ATTACHMENTS.deleteFile(input)	To delete files(attachments) that were attached to a record in a module.
crmclient.API.ATTACHMENTS.uploadPhoto(input)	To upload photos/images to a record in a module.
crmclient.API.ATTACHMENTS.downloadPhoto(input)	To download photos/images that were attached to a record in a module.
crmclient.API.ATTACHMENTS.deletePhoto(params)	To delete photos/images that were attached to a record in a module.

Release Notes

Current Version:

1. ZCRMSDK -VERSION 0.0.20

- Install command

```
npm install zcrmsdk@0.0.20
```

Notes

Fixed the mandatory check for redirect_url in configuration JSON.

Previous Versions

1. ZCRMSDK -VERSION 0.0.19

- Install command

```
npm install zcrmsdk@0.0.19
```

Notes

- Support provided for the getPersistenceModule() in ZCRMRestClient.js.

2. ZCRMSDK -VERSION 0.0.18

- Install command

```
npm install zcrmsdk@0.0.18
```

Notes

Custom token management.

3. ZCRMSDK -VERSION 0.0.17

- Install command

```
npm install zcrmsdk@0.0.17
```

Notes

- Configuration details can be sent as JSON while initializing the SDK.
- Handled the error Response.

4. ZCRMSDK -VERSION 0.0.16
Install command

```
npm install zcrmsdk@0.0.16
```

Notes
Handled the change in the OAuth token response.

5. ZCRMSDK -VERSION 0.0.15
- Install command

```
npm install zcrmsdk@0.0.15
```

Notes
- User identifier default value handled in get method.

6. ZCRMSDK -VERSION 0.0.14
- Install command

```
npm install zcrmsdk@0.0.14
```

Notes
- Default user identifier set internally in case user doesn't set it.

7. ZCRMSDK -VERSION 0.0.13
- Install command

```
npm install zcrmsdk@0.0.13
```

Notes
- Custom headers support added.
- Unwanted code removed.

8. ZCRMSDK -VERSION 0.0.12
- Install command

```
npm install zcrmsdk@0.0.12
```

Notes

- Issue fixes

9. ZCRMSDK -VERSION 0.0.11

- Install command

```
npm install zcrmsdk@0.0.11
```

Notes

- Refresh token key added in result.

10. ZCRMSDK -VERSION 0.0.10

- Install command

```
npm install zcrmsdk@0.0.10
```

Notes

- Header added.

11. ZCRMSDK -VERSION 0.0.7 - 0.0.9

- Install command

```
npm install zcrmsdk@0.0.x
```

Notes

- Issue fixes.

12. ZCRMSDK -VERSION 0.0.6

- Install command

```
npm install zcrmsdk@0.0.6
```

Notes

- Token management taken from oauth_configuration.properties fixed.

13. ZCRMSDK -VERSION 0.0.1 - 0.0.5

Install command

```
npm install zcrmsdk@0.0.x
```

Notes

- Issue fixes.

14. ZCRMSDK -VERSION 0.0.0

Install command

```
npm install zcrmsdk@0.0.0
```

Notes

- Initial release.