REST API Documentation

USER GUIDE

September 2019

MemberSuite™
EMPOWER YOUR FUTURE

Table of Contents

Overview

The MemberSuite REST API allows the client to take their user specific credentials and make calls to their environment to retrieve, update, add, and remove data. The MemberSuite REST API was built on the open source project, Swagger UI, to give the client the smoothest experience while making calls to their environment. REST architecture allows the Swagger UI to provide multiple data formats rather than the sole option of xml, provided by SOAP. JSON has been set as the default format for the parameters used to make a call, but that format can be changed to html, xml, yaml, and other options listed within the parameter content type drop down.

Three main endpoints have been exposed for client use: Platform, Security, and GraphQL. MemberSuite has exposed many objects within the three endpoints each containing various operations for editing the specified object. The operations available to the user are: GET, POST, PUT, and DELETE. The available operations for each object vary depending upon the security surrounding that object. For example, the association endpoint within platform only contains a GET call, because the other calls would be useless or possibly damaging to the data within the client's environment. In contrast, the CustomFields endpoint within Platform contains each of the available calls listed above allowing the user to obtain or change the custom fields object within their environment.

Credentials

Authentication

Credentials are necessary to create an AuthToken which will be used to access the customers environment. The user will need to navigate to the platform endpoint (blue URL below) with their username, password, and clientID. The tenantid also needed to obtain the user specific token will be provided by MemberSuite.

- Navigate to the AuthToken segment of the platform endpoint (as pictured below) within the REST API, then expand the POST call operation.
- Double click the model parameters to input them into the value field prior to changing the values to your own data.
- Once you have determined your credentials to be accurate, click "Try it out!" to proceed. This will output a bearer token which must be taken to the security endpoint (as pictured below) where the token will be used under the PortalUsers tab.

- Expand the PortalUsers tab and place the bearer token obtained from the previous page into the parameter value box labeled authorization.
- Once the token has been entered, the user will click "Try it out!" to receive the Base64 encoded auth token needed to access the user specific environment through the REST endpoint.
- The Auth token will then need to be taken to an API development environment such as Postman, so the user may create and obtain the URL for their environment.

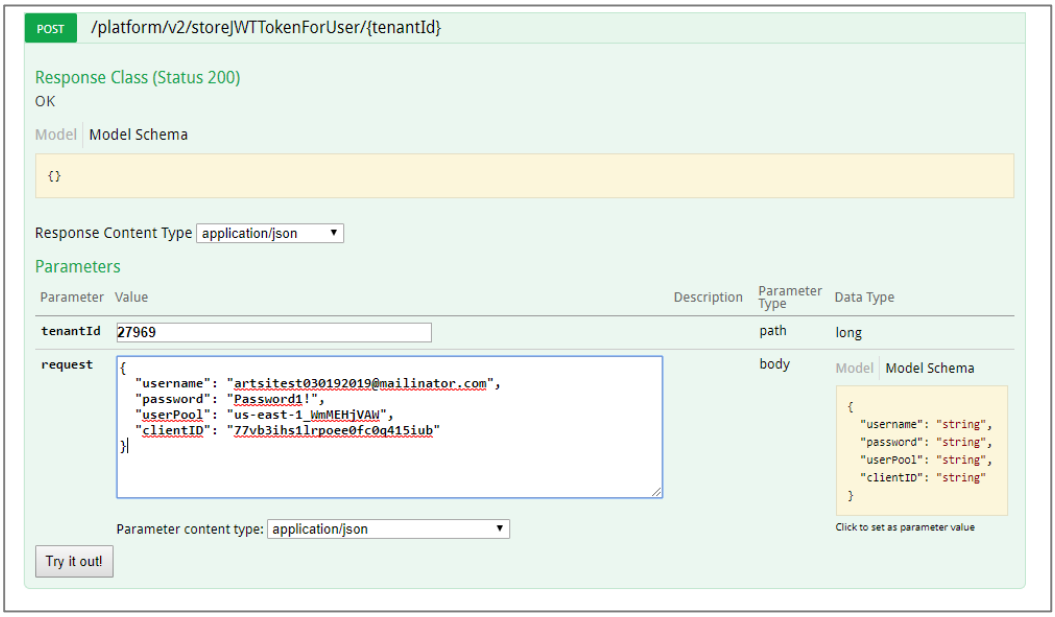
Single Sign-On Using the JWT Token:

Single Sign-on is used with MemberSuite to provide a seamless transition for members from an association's password protected environment into the MemberSuite Portal without the user having to re-enter a password or a completely different set of credentials.

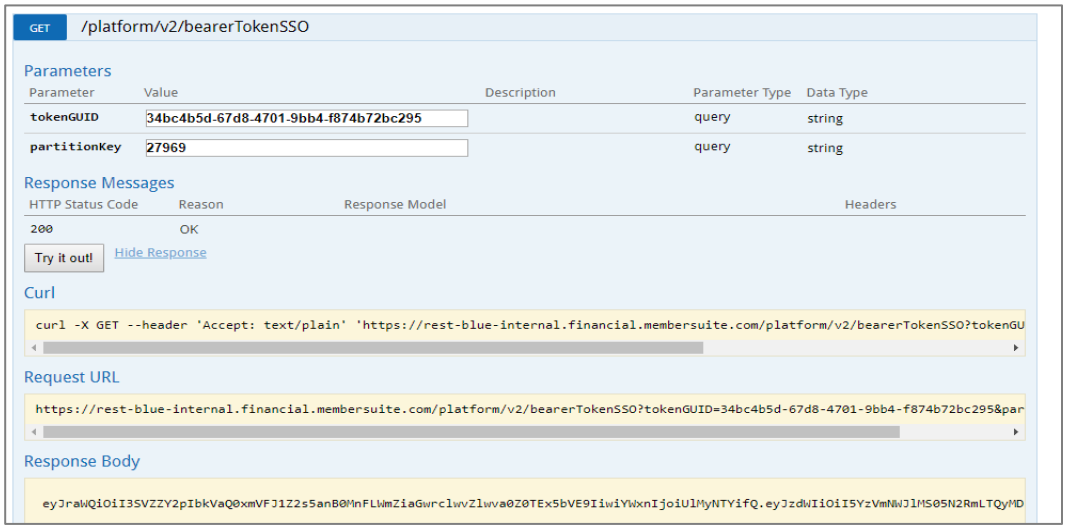
Existing User:

In order to implement Single Sign on we need to obtain a JWT token that identifies the user.

1. The third-party provider will have to send the username, password, client id, and user pool to obtain the JWT token for an already existing User. This may be done via the REST API calls on the Platform Endpoint located at http://rest.membersuite.com/swagger/platform/v2/storeJWTTokenForUser/27969
- Note: the number at the end is a sample tenant Id. If creds are incorrect there will be a 401 response.



2. Once the POST is made with correct credentials there will be a redirect directly to the MemberSuite URL that includes a query parameter bearerTokenGUID=4e0d6891-6efb-4c16-9348-2edd1d3e3bb9. This will sign in the user and allow them to look around the site using the JWT token. To verify the token:



New User:

In order to properly obtain a JWT token for New Users:

1. New users will sign into their third-party client website using their MemberSuite credentials.

2. They will then be redirected to the Cognito UI for user creation and then must proceed to step 2 of the enrollment processes. If they do not proceed to step 2 of enrollment, they are not considered a valid user.
3. The JWT token used for SSO is only valid for a half hour in which case they will have to revalidate.
4. The JWT token after initial SSO is valid for an hour on the site itself.

Single Sign-On using the SOAP Token:

Single Sign-on is used with MemberSuite to provide a seamless transition for members from an association's password protected environment into the MemberSuite Portal without the user having to re-enter a password or a completely different set of credentials.

Any word in `` is a variable name: e.g.: `eventId` stands for the actual id of the event. This id will vary from event to event.

xxxx gets replaced by the host/client name.

Event Flow:

Step - 1: As a non-logged in user, user hits the URL:

https://xxxx.users.membersuite.com/event/`eventId`/details?entry=0

Step - 2: The user is redirected to the wordpress login URL with query parameters:

nextUrl=events, entry=1, id=`eventId`

Step - 3: If the user logs in: The user is redirected to

https://xxxx.users.membersuite.com/auth/sso?tokenGUID=`guidToken`&nextUrl=events&id=`eventId`&entry=1

Next the user is navigated to event registration flow

Step - 4: If the user signs up: The user is redirected to

https://xxxx.users.membersuite.com/auth/sso?nextUrl=events&id=`eventId`&entry=1&signup=1

Next the user is navigated to Cognito sign up and then event registration flow

*Note:

-- In case of signup, user won't be logged in the wordpress site.

-- In case of signup, tokenGUID should not be there

Normal Flow:

If the user logs in: The user is redirected to

https://xxxx.users.membersuite.com/auth/sso?tokenGUID=`guidToken`&nextUrl=`nextUrl`

If the user signs up: The user is redirected to

https://xxxx.users.membersuite.com/auth/sso?nextUrl=`nextUrl`&signup=1

*Note: If the nextUrl is a complete URL including https protocol, then it would be treated as a wordpress URL and the user would be redirected to the nextUrl in the wordpress site

*Note: If the nextUrl is a partials URL (e.g.: store/browse), then it would be treated as a portal URL and the user would be redirected to the nextUrl in the portal site

Logout:

If the user logs out from wordpress website, he is redirected to:

https://xxxx.users.membersuite.com/auth/sso?logOut=1.

The user logs out of portal and is redirected back to wordpress logout URL

If the user logs out from portal, he is redirected to the wordpress logout URL, where he would be logged out of wordpress site as well.

apisUsed:

/regularSso: POST Method: Wordpress site submits the token to this REST API. The response header of this API redirects the user to portal along with tokenGuid

/RegularSSO: GET Method: Portal takes the tokenGuid from the query params and hits this API and gets the actual SSO token which is used as the access token going forward.

Get/Put

The combination of the Get/Put SSO endpoints is used to temporarily store and retrieve a token, sent from a third-party application trying to connect with the MemberSuite p5 portal. Due to the length of the token and the design of the updated portal the token cannot be passed directly to the portal environment.

Post

The call takes in a token and tenantId to return a temporary GUID to be used in the subsequent call.

Endpoint: platform/RegularSSO

Input: key, value pair of the token

Output: token GUID

Get

The Guid returned by the previous call is used to retrieve the token and gain access to the updated portal.

Endpoint: platform/RegularSSO

Input: tokenGuid and the tenantId

Output: portal redirect URL

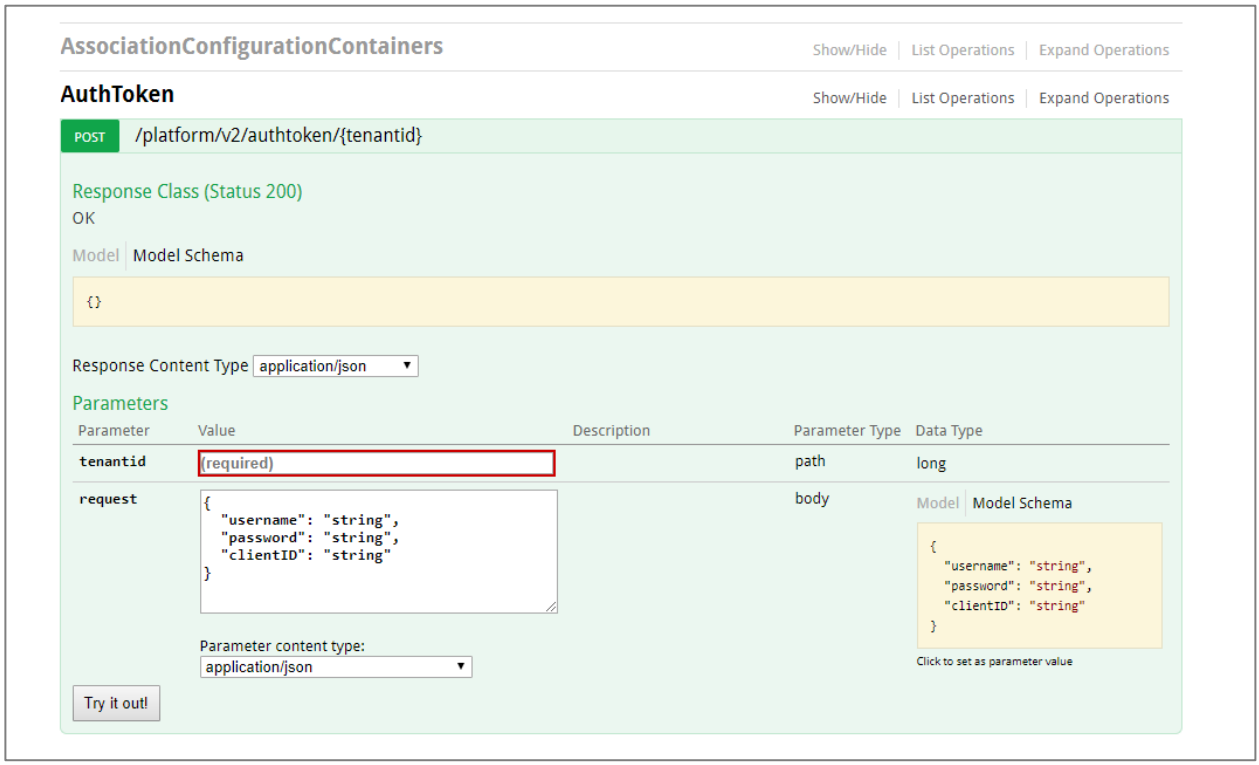
-which will take you to the updated portal.

How To: Locate Your Association ID

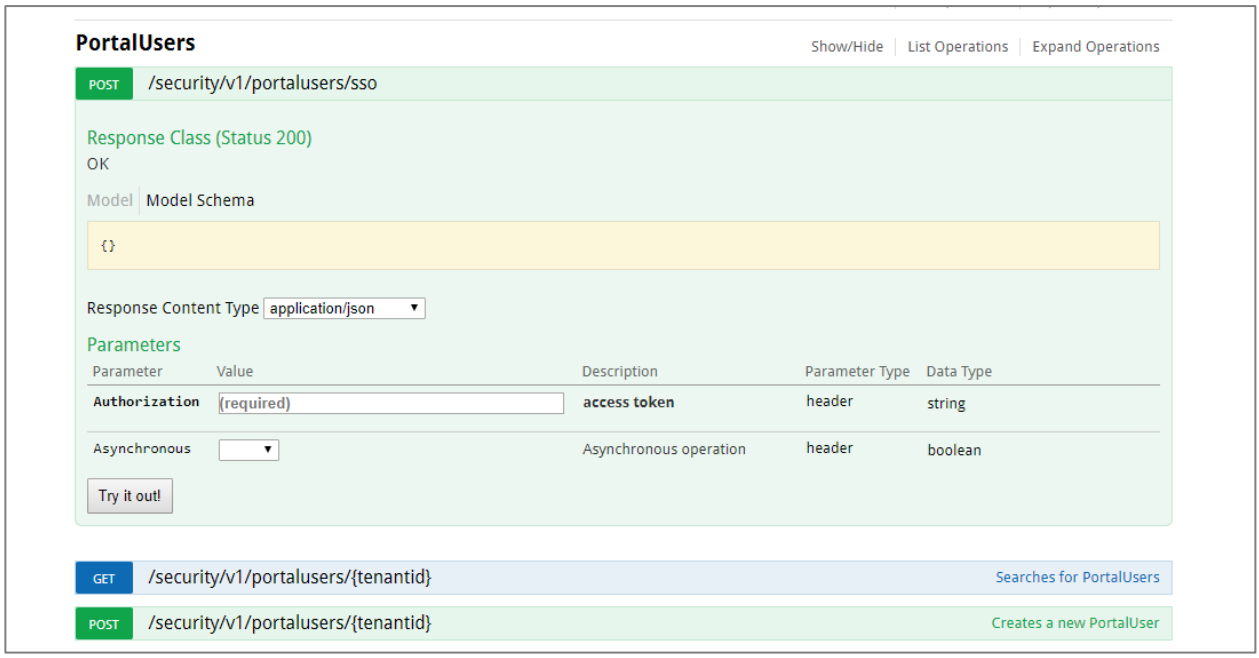
You must specify your Association ID in the header of each request to the Concierge API. You can determine your Association ID by logging into the console and navigating to the Association Settings screen.

1. Click on Setup in the Console toolbar
2. Click on Association Settings
3. Make a note of your specific Association's ID

https://rest.membersuite.com/platform/swagger/ui/index#!



https://rest.membersuite.com/security/swagger/ui/index#!



Platform

The platform endpoint of the MemberSuite Rest API contains an extensive list of operations to choose from when making a call to the user's environment. The user can use these operations to make a POST, GET, PUT, or DELETE call. The types of calls available vary depending upon the chosen operation. The examples shown below describe how to use the REST API to make such calls.

Platform: Association

https://rest.membersuite.com/platform/swagger/ui/index#/Association

For security purposes, the association segment of the platform endpoint only contains a GET call that will return the response data for the specified tenantid. The model schema above the value box for the tenantid parameter specifically shows what the customer can expect from the GET call. Once the tenantid has been entered, a response body containing the data for the association will be produced in the output.

Input Parameters:

tenantId: 27969

Response Body:

{
"id": "8902ce52-0004-ca0d-7305-0b3dd2c24826",
"userPoolClient": "77vb3ihs1lrpoee0fc0q415iub",
"userPool": "us-east-1_WmMEHjVAW",
"baseCurrency": "USD",
"timeZone": null,
"bluePayAPISignature": "46e23d957b30a550a426e751f340e43a",
"bluePayMerchantLoginID": "100355734958",
"navigation": {
"store": true,
"events": true,
"directory": true
},
"priorityPaymentsMerchantLoginID": null,

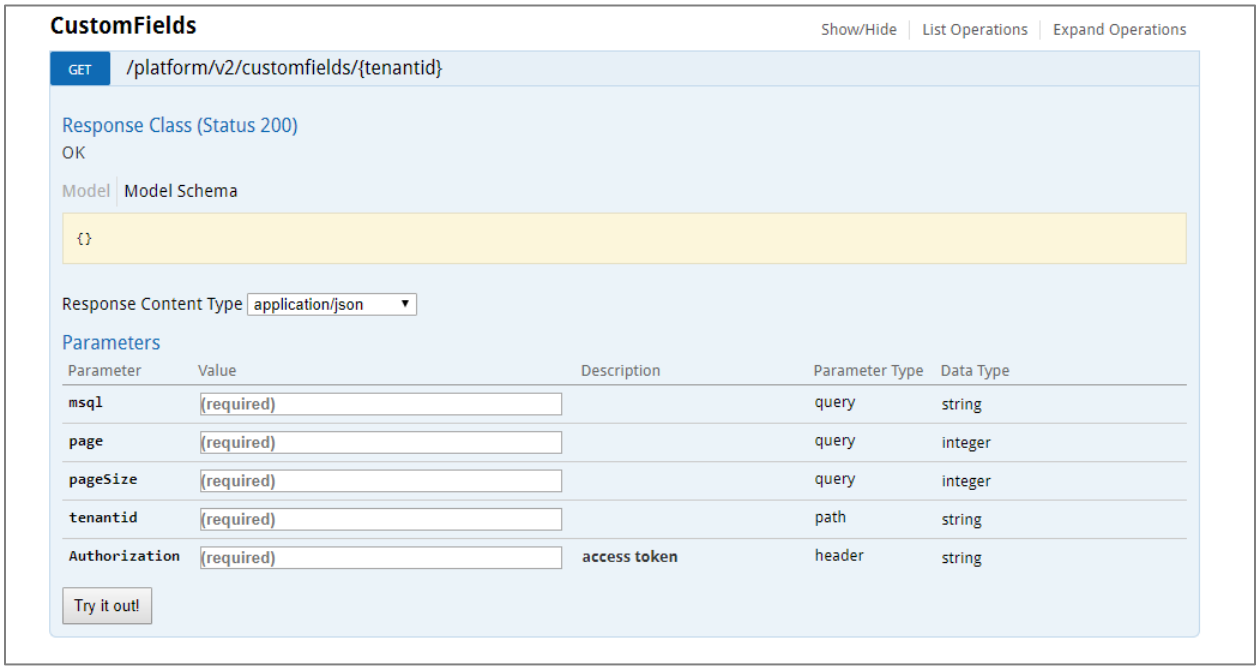
```
  "priorityPaymentsPartner": null,
  "restrictModules": [
    "Accreditations",
    "Advertising",
    "Awards",
    "Billing",
    "CareerCenter",
    "Certifications",
    "Committees",
    "CRM",
    "Documents",
    "EMarketing",
    "Engagement",
    "Events",
    "Exhibits",
    "Financial",
    "Fundraising",
    "Analytics",
    "LegislativeAffairs",
    "Membership",
    "Orders",
    "Prospects",
    "Realtors",
    "ReportStudio",
    "Subscriptions",
    "Volunteers",
    "Discussions",
    "Reports"
  ]
}
```

Platform: CustomFields

https://rest.membersuite.com/platform/swagger/ui/index#/CustomFields

The CustomFields segment within the platform endpoint contains three separate GET calls, as well as PUT, POST, and DELETE options.

The tenantid GET operation call requires more parameters than the calls previously stated. As shown below, you will need parameter values for msql, page, pagesize, tenantid, and the authorization token created in the PortalUsers segment of the security endpoint within the REST API.



Input Parameters:

msql: SELECT id, name FROM CustomField

page: 0

pageSize: 0

tenantId: 27969

Authorization: Base64 Encoded JWT.

Response Body: List of the asked values in the msql query.

[
{
"roW_NUMBER": 1,
"id": "8902ce52-000e-c056-3c43-0b3991c4decd",
"name": "Doyouhaveanyfoodallergies___q",
"iD1": "8902ce52-000e-c056-3c43-0b3991c4decd"
},

.

.

.

{
"roW_NUMBER": 318,
"id": "8902ce52-000e-ceb1-cc09-0b3f689f1e51",
"name": "ListTestNew__c",
"iD1": "8902ce52-000e-ceb1-cc09-0b3f689f1e51"
}
]

The description/tenantid GET operation call requires less values to be passed, and outputs a response body

describing the details of what data is accepted and what data will be null for the platform custom fields.

Input Parameters:

tenantId: 27969

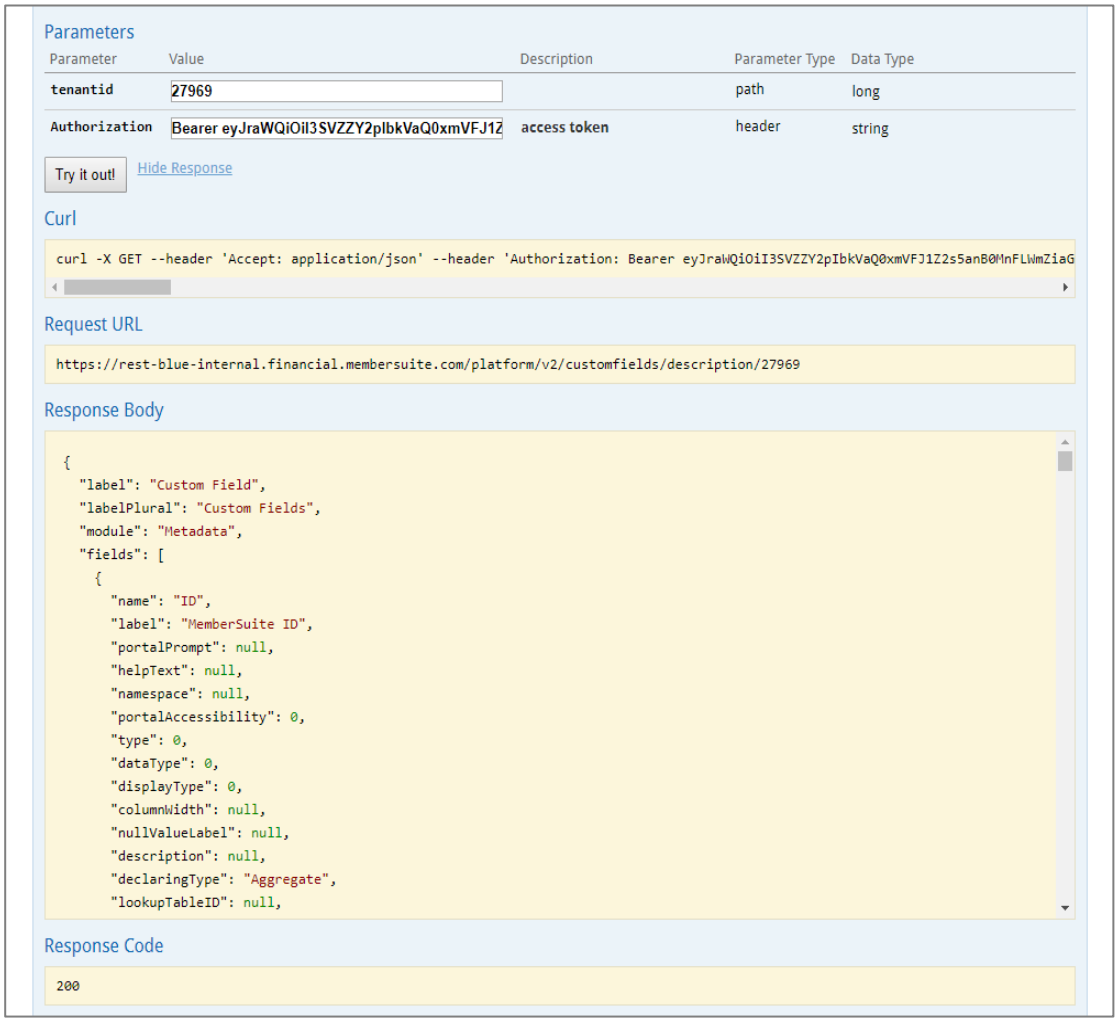
Authorization: Base64 Encoded JWT.

Response Body: List of all the available custom fields with all the possible sub-values that describes the field.

{
"label": "Custom Field",
"labelPlural": "Custom Fields",
"module": "Metadata",
"fields": [
{
"name": "ID",
"label": "MemberSuite ID",
"portalPrompt": null,
"helpText": null,

"namespace": null,
"portalAccessibility": 0,
"type": 0,
"dataType": 0,
"displayType": 0,
"columnWidth": null,
"nullValueLabel": null,
"description": null,
"declaringType": "Aggregate",
"lookupTableID": null,
"extensionServiceID": null,
"relationshipTypeID": null,
"sortable": true,
"displayable": true,
"defaultValue": null,
"minimumValue": null,
"maximumValue": null,
"pickListEntries": [],
"referenceType": null,
"isRequired": false,
"isRequiredInPortal": false,
"isSealed": false,
"isReadOnly": true,
"accessLevel": 20,
"doNotDescribe": false,
"isOverriden": false,
"customFieldID": null,
"referenceTypeContext": null,
"metadataType": null,

"doNotConvertTimeToUTC": false,
"precision": 2,
"suppressDefaultValue": false,
"startingYear": null,
"cert20": false,
"endingYear": null,
"applicableType": null
},
.
.
.
]



Parameters

Parameter	Value	Description	Parameter Type	Data Type
tenantid	27969		path	long
Authorization	Bearer eyJraWQiOiI3SVZZY2pIbkVaQ0xmVFJ1Z	access token	header	string

Try it out! Hide Response

Curl

```
curl -X GET --header 'Accept: application/json' --header 'Authorization: Bearer eyJraWQiOiI3SVZZY2pIbkVaQ0xmVFJ1Z2s5anB0MnFLWmZiaG
```

Request URL

```
https://rest-blue-internal.financial.membersuite.com/platform/v2/customfields/description/27969
```

Response Body

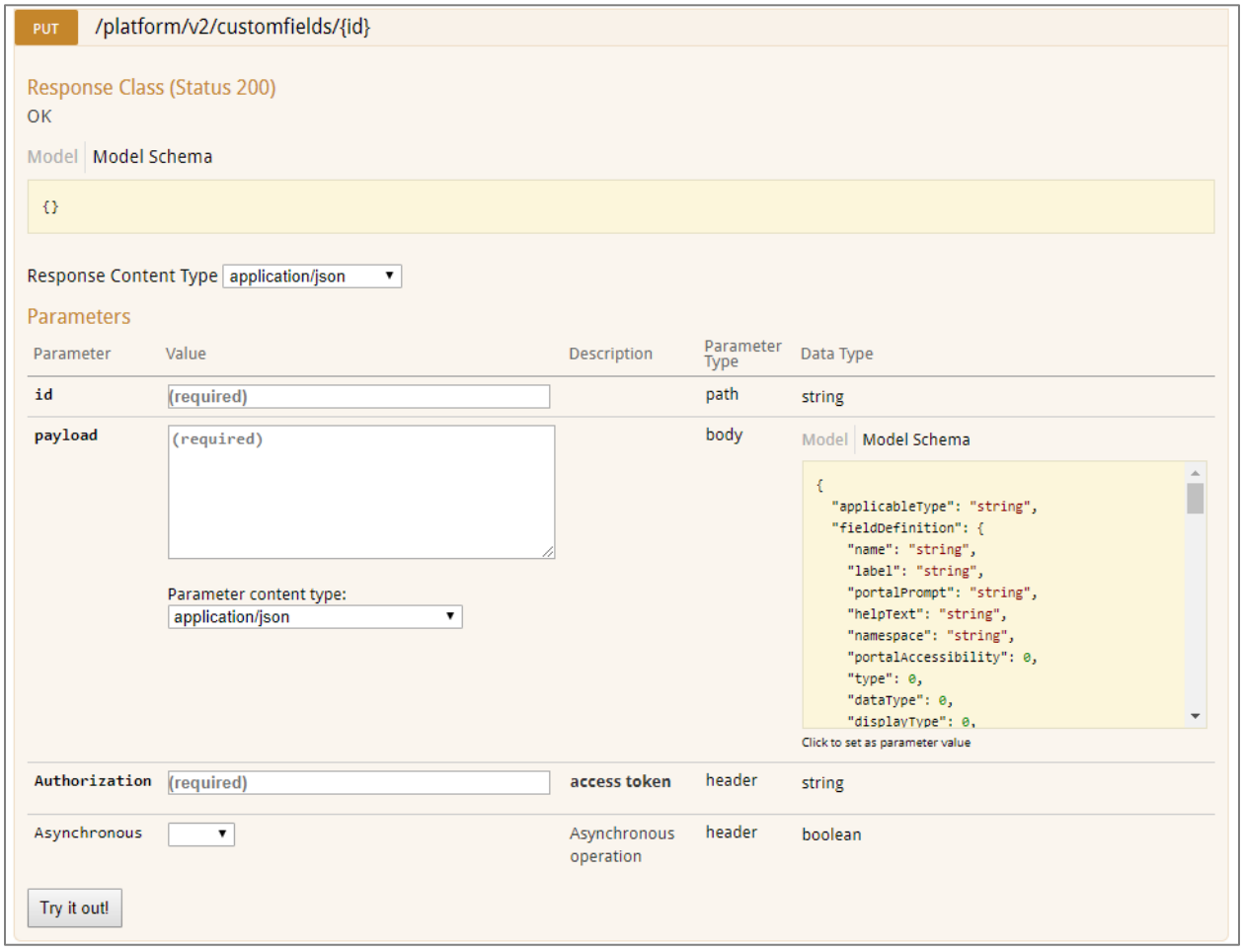

```
{
  "label": "Custom Field",
  "labelPlural": "Custom Fields",
  "module": "Metadata",
  "fields": [
    {
      "name": "ID",
      "label": "MemberSuite ID",
      "portalPrompt": null,
      "helpText": null,
      "namespace": null,
      "portalAccessibility": 0,
      "type": 0,
      "dataType": 0,
      "displayType": 0,
      "columnWidth": null,
      "nullValueLabel": null,
      "description": null,
      "declaringType": "Aggregate",
      "lookupTableID": null,
```


Response Code

```
200
```

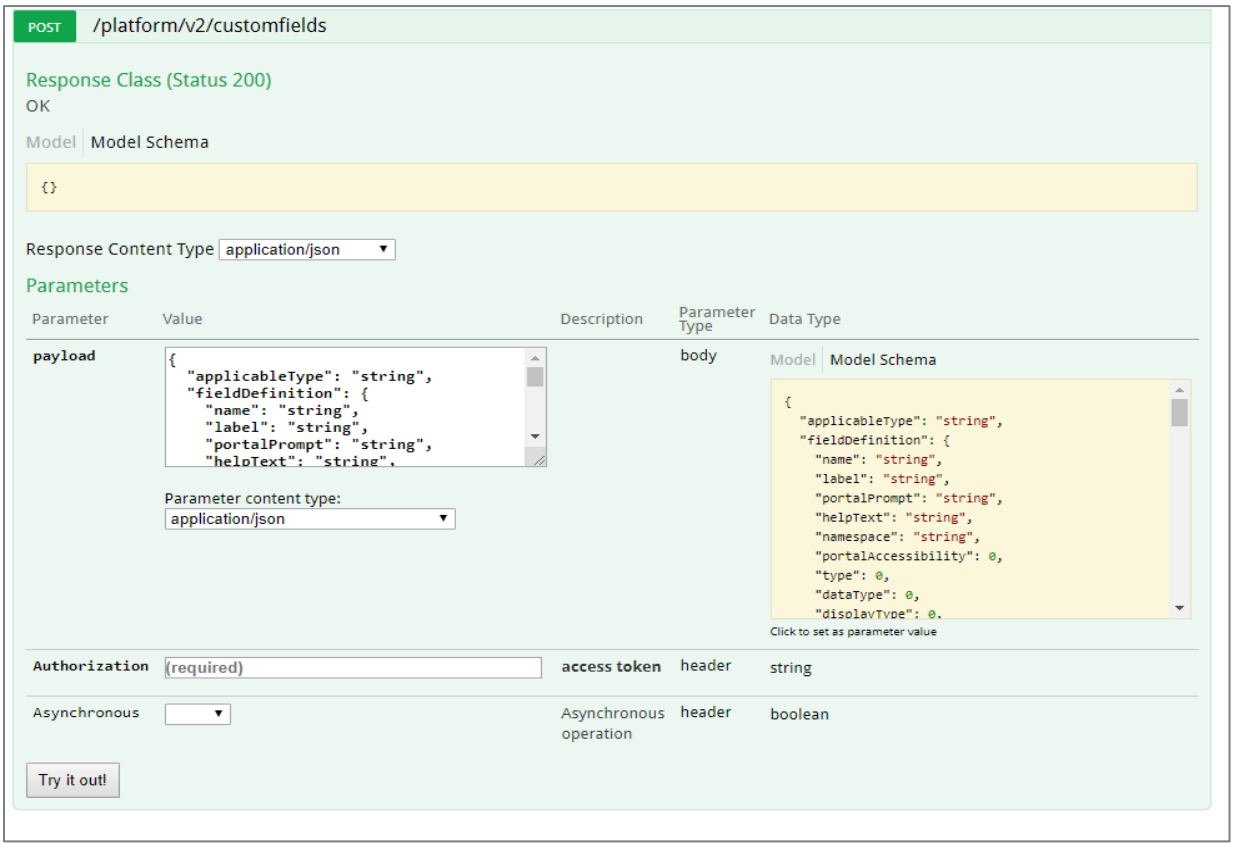
PUT

The PUT operation call requires an extensive payload to be passed as well as an authorization token. Together, these values will be used to change the customfields content within the user's environment.



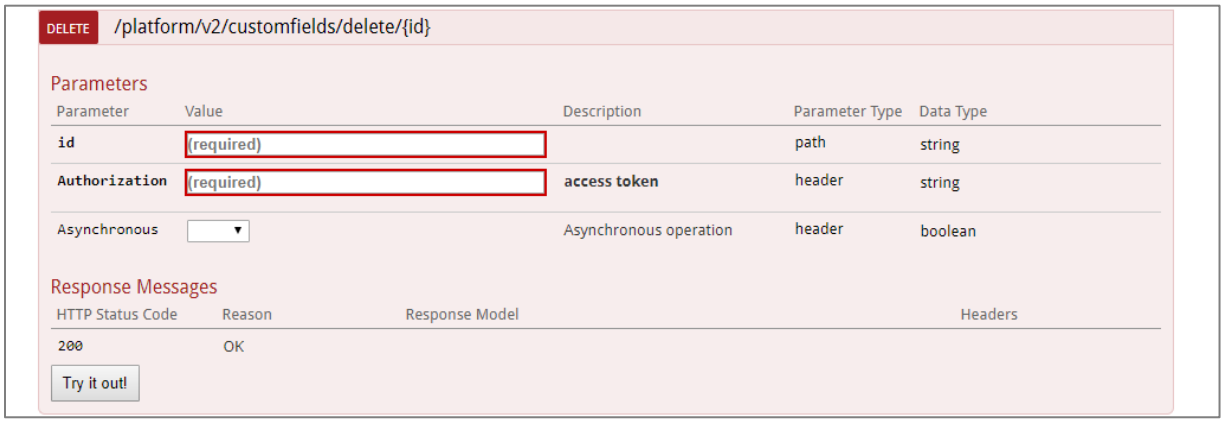
POST

The POST call for custom fields requires a payload containing the information to be passed as the value as well as an access token.



DELETE

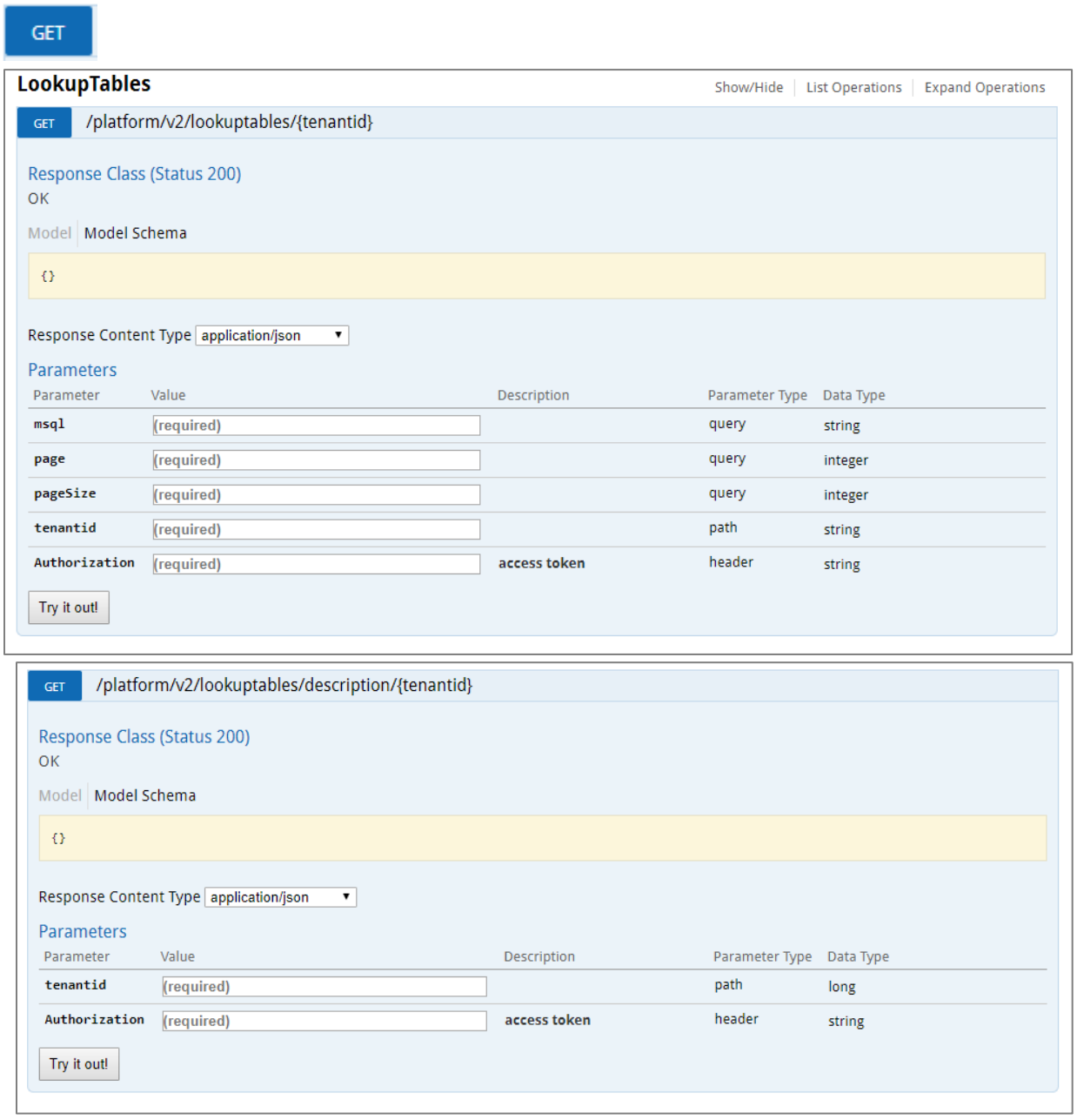
The DELETE call requires the id and authorization parameters to be passed. This call is used to delete an id out of the custom fields segment of the environment



Platform: LookupTables

https://rest.membersuite.com/platform/swagger/ui/index#/LookupTables

The lookuptables object of the platform endpoint contains three separate GET calls, as well as a PUT, POST, and DELETE call like the custom fields example described above.



GET /platform/v2/lookuptables/{id}

Response Class (Status 200)
OK

Model | Model Schema


```
{
  "description": "string",
  "rows": [
    {
      "name": "string",
      "value": "string"
    }
  ],
  "isConfiguration": true,
  "securityLock": {
```


Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	(required)		path	string
Authorization	(required)	**access token**	header	string

Try it out!

PUT

PUT /platform/v2/customfields/{id}

Response Class (Status 200)
OK

Model | Model Schema


```
{}
```


Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	(required)		path	string
payload	(required) Parameter content type: application/json		body	Model | Model Schema { "applicableType": "string", "fieldDefinition": { "name": "string", "label": "string", "portalPrompt": "string", "helpText": "string", "namespace": "string", "portalAccessibility": 0, "type": 0, "dataType": 0, "displayType": 0, Click to set as parameter value
Authorization	(required)	**access token**	header	string
Asynchronous		Asynchronous operation	header	boolean

Try it out!

POST

POST /platform/v2/customfields

Response Class (Status 200)

OK

Model | Model Schema

```
{}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
payload	"namespace": "string", "portalAccessibility": 0, "type": 0, "dataType": 0, "displayType": 0, "columnWidth": 0, Parameter content type: application/json		body	Model Schema { "applicableType": "string", "fieldDefinition": { "name": "string", "label": "string", "portalPrompt": "string", "helpText": "string", "namespace": "string", "portalAccessibility": 0, "type": 0, "dataType": 0, "displayType": 0, Click to set as parameter value
Authorization	(required)	access token	header	string
Asynchronous		Asynchronous operation	header	boolean

Try it out!

DELETE

DELETE /platform/v2/customfields/delete/{id}

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	(required)		path	string
Authorization	(required)	access token	header	string
Asynchronous		Asynchronous operation	header	boolean

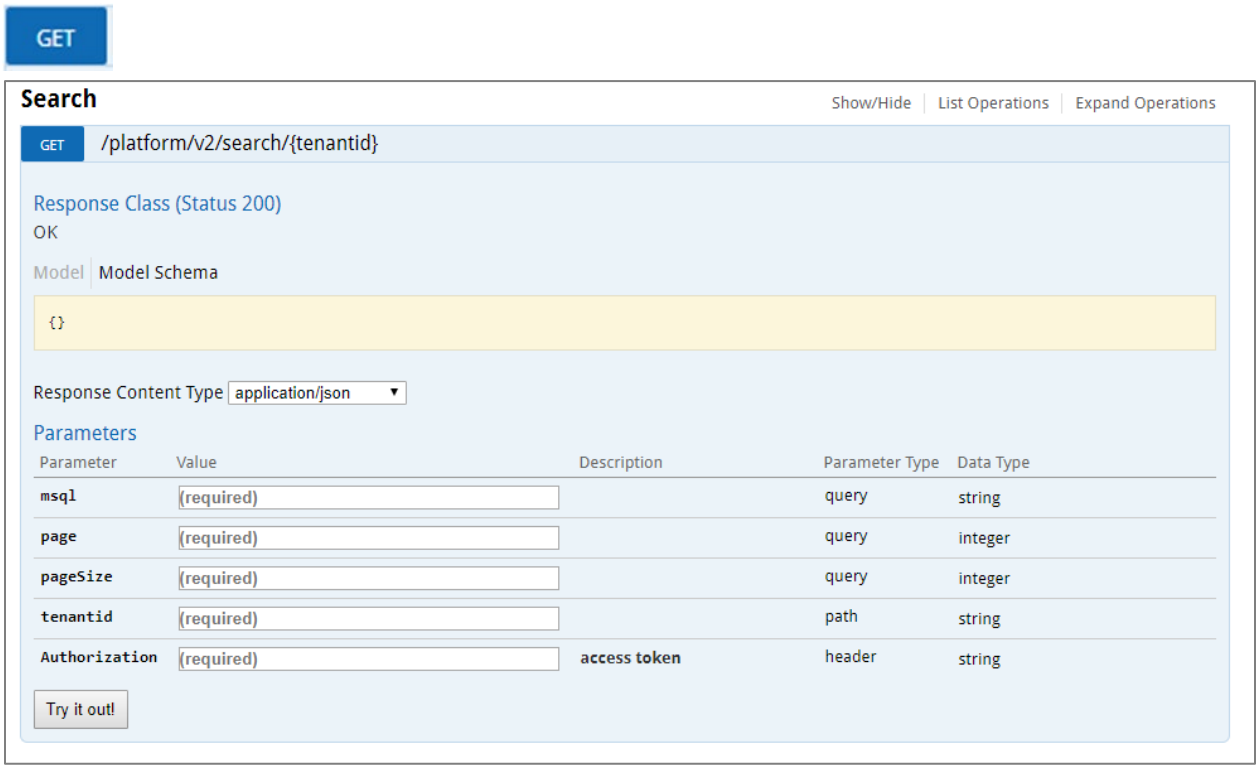
Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	OK		

Try it out!

Platform: Search

The Search segment of the Platform endpoint within the REST API contains only a GET operation. The necessary parameters pictured below will be used to output the search results specific to those values.



Platform: WhoAmI

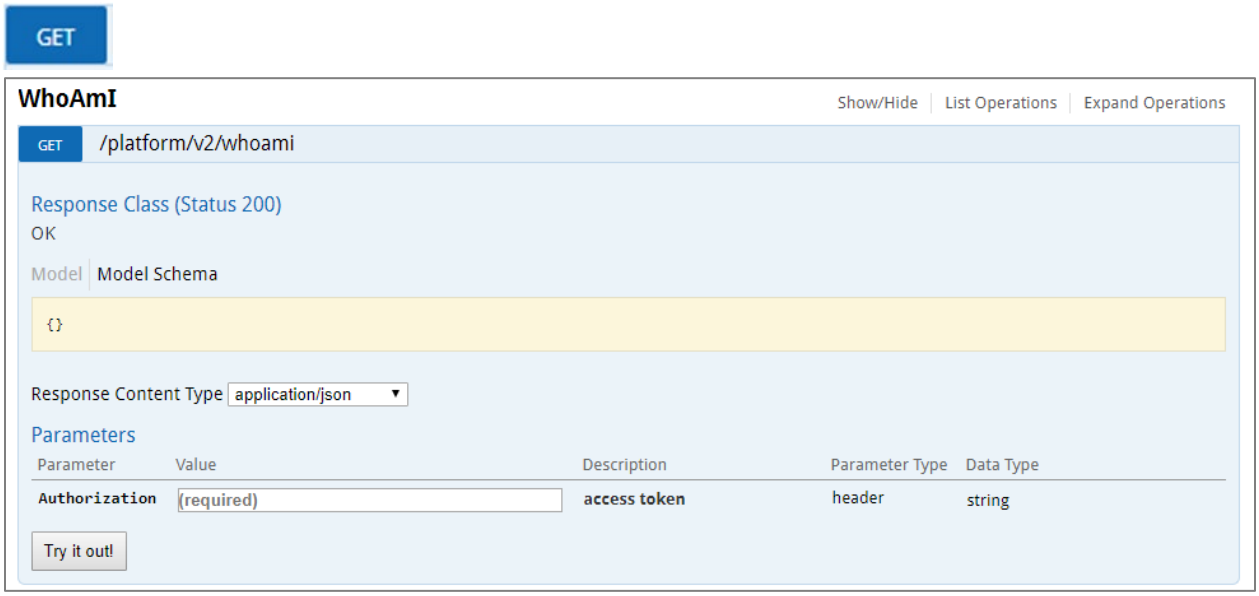
The WhoAmI object of the Platform endpoint within the REST API contains only a GET operation. An authorization token obtained through the security endpoint as described earlier in the credentials section is all that is required to complete the GET call within the WhoAmI segment of the platform endpoint.

Input Parameters:

Authorization: Base64 Encoded JWT.

Response Body:

{
"tenantId": 27969,
"associationId": "8902ce52-0004-ca0d-7305-0b3dd2c24826",
"userId": "00000000-0032-c171-d271-0b3f653a5faa",
"email": "dude00701@yopmail.com",
"firstName": "fghd",
"lastName": "sfgsd",
"ownerId": "8902ce52-0006-c5ea-4962-0b3f653a5fa1",
"membershipId": null,
"receivesMemberBenefits": false,
"paymentProcessor": "BluePay",
"businessUnit": "8902ce52-0034-c836-158e-0b3c94884698",
"merchantAccount": "8902ce52-01f8-c76e-c7d2-0b3c948886cd",
"baseCurrency": "USD"
}



Platform: AuthToken_alt

The alternate AuthToken is specific to only certain organizations attempting to integrate with the MemberSuite REST API. The example query below is different than the previously mentioned information regarding platform authorization and should be disregarded unless told otherwise.

The AuthToken object of the Platform endpoint within the REST API contains only a POST operation.

https://rest.membersuite.com/platform/swagger/ui/index#!/AuthToken_v1/AuthToken_v1_PostAlternateAuth

{

"username": "xxxxx@mailinator.com",

"password": "xxxxxxx",

"userPool": "xxxxxxxxx",

"clientID": "xxxxxxxxxxxxxxxx"

}

Response Body:

{

"access_token":
"eyJraWQiOiI2dXJtS0tDMU9mQnc4dWNEUzQxYWtvK09JeDJoWXVNMVVFS2lERnhMYUxnPSIsImFsZyI6IlJTMjU2In0.eyJzdWIiOiI5Y2YyMzc0ZC1lMDE0LTRiMWYtYmUyYi0yOTg0Njc5MjJlNDQiLCJhdWQiOiI2M3I4Z2JkanVjc2QyY2FnbGg0OHN1rts2MyIsImV2ZW50X2lkIjoiNTA1Yzk2MjMtYjJiNS00MTJjLWI5NzctMGMwMTJhMWRhNzZkIiwidG9rZW5fdXNlIjoiaWQiLCJhdXRoX3RpbWUiOjE1Njc2MTA5NjAsImlzcyI6Imh0dHBzOlwvXC9jb2duaXRvLWlkcC51cy1lYXN0LTEuYW1hem9uYXdzLmNvbVwvdXMtZWFzdC0xXzFBR2k5cjB2VCIsImNvZ25pdG86dXNlcm5hbWUiOiJyZXN0YXBpdGVzdEBtYWlsaW5hdG9yLmNvbSIsImV4cCI6MTU2NzYxNDU2MCwiaWF0IjoxNTY3NjEwOTYwLCJlbWFpbCI6InJlc3RhcGl0ZXN0QG1haWxpbmF0b3IuY29tIn0.ShooTde47kloVK3JIo4z0PaveY-KGSuFZkp4o88M3qy_k75956WwuoElTAJ7T6AmIorwLclnbR7-V4BEcHogE1vWPaANRYg0aeYkouUhMSA8__qudFNoDnHEeADvxRwK2j16vilT65-2aBZVN8CM7jxh-TrStyQCT8J6mi0l5WkXJYEKjHbwReCjQjOwG_LMasdcjzuqCj1hVCw

pq73MElpzIY0oV_GpICA4POt_SkzVSAq2gRh6An6C_PSzLgCOti3wbqvO5ZdljZd0LpMlJzCHufYlyK7ouNLUNIAm-gc9-GvPgFoHqSDq6nRnjCnVPODNtRI0Al2v28gUdd5w",

"token_type": "bearer",

"expires_in": "3600",

"refresh_token": "",

"refresh_token_server_uri": "",

"refresh_server_base_uri": "",

"scope": "RealTimeApi AdminApi AuthenticationApi PatronApi AgentApi CustomApi",

"agent_id": "0",

"team_id": "0"

}

Search in MemberSuite

Overview

The most common thing you will do in MemberSuite is to pull data out of the system. The mechanism for querying data in MemberSuite is called Search. There are two primary methods for search:

- MemberSuite Query Language (MSQL) – this is a SQL-like language that allows you to select, insert, and update records
- Search object – this is an object that represents a search, including search criteria, output columns, and sorting

MemberSuite provides an easy service for converting a MSQL string to a Search object, because internally, MemberSuite uses Search objects for everything.

MemberSuite Query Language

The MemberSuite Query Language is a SQL-like language that lets you SELECT/INSERT/UPDATE records. The basic syntax:

- SELECT columns from <searchType> where <criteria
- UPDATE <record> SET <fieldsToSet> WHERE ID = id
- INSERT INTO <record> (columns) VALUES <valuesToInsert>

You can execute MSQL via the ExecuteMSQL method in the API. Some examples of MSQL queries:

- Select FirstName, LastName from Individual where Email like '%gmail%' or Age> 15 and Gender_c= 'Male' order by CreatedDate
- Insert into Individual (FirstName, LastName) values ('Andrew', Ryan')
- Update Individual set LastName= 'Ryan' where ID = 594811f7-0006-4feb-b7c1-ae841b42a9c2'

The MemberSuite Object

Overview

In MemberSuite's system, domain objects are core objects that contain properties, methods, etc. But when you integrate with MemberSuite via the API, you need to be able to get at objects in a language-agnostic, serializable way. For this, we utilize the MemberSuiteObject.

The MemberSuite object is an implementation of the Data Transfer Object design pattern. Instead of being a core in-memory object, it is a collection of name/value pairs that represent the object. The MemberSuiteObject is designed to be serializable, meaning that it can travel across the API boundary without a problem. Here again we see the usefulness of aggregates; the aggregate boundary can now be considered a whole MemberSuiteObject for the purposes of transport.

Whenever you request an object from the API, the API goes through and converts the response to a MemberSuiteObject via a process called serialization. The MemberSuiteObject gives us several benefits:

- Its name/value pair structure can easily travel across the Internet, and be consumed by any language
- An association's custom fields can be placed in the name/value pair collection, making the MemberSuite object a direct representation of the association's object structure
- Entirely new objects can be created/transported, like Custom Objects, because the MemberSuiteObject is not constrained to any in-memory object

That second point is important. Associations have the ability to create custom fields for their objects. Those will always come across as fields that end in __c. So, you can always tell a custom field by looking for keys in the MemberSuiteObject ending in "__c".

System Fields

Since every object in MemberSuite derives from Aggregate, there are some base fields that each object has. They are:

- **ID** – this is the GUID identifier that was described in the previous article
- **Name** – by rule, all objects in MemberSuite must have a name. This name need not necessarily be unique, but there must be a name

- **SystemTimestamp** – this is a binary field that tracks each time a record was updated. When you save an object, the timestamp is compared with the one in the database, and if the record has changed, you'll need to get a new copy before saving again

Other Common Fields

In addition to the system fields, here are fields you see frequently on objects:

- **LocalID** – this is the numeric ID assigned to the object. This is unique to the association specifically. Think of this like an identity key in a traditional database
- **Code** – Configuration objects often use a code in addition to a name. This makes it easy to reference the object by Code, so that if the Name changes, references to the object do not break

Object Description

Since each association can have their own custom fields, and even their own custom objects, there needs to be a way to determine what fields are available for an object. This process is called object description. This result of description is class metadata. Class metadata incudes field names, types, default values, visibilities, and several other properties.

You can call the DescribeObject API method to describe an object.

The Search Object

Usually, you will find it easier to use MSQL for querying data. It's simpler and easier to use. However, there may be times when you want to construct your own search object, particularly if you're designing a search programmatically.

At the base of a search object is a SearchOperation, which is a single operation/criterion from a search. A SearchOperationGroup is a child item of SearchOperation, as it is a special type of criterion that has child criteria. A Search is the principle object, which contains:

- Type – the search type. This is usually the object type, like Individual or Organization
- Context – if the search has a context (for instance, searching for event registrations for an event)
- Criteria – a collection of SearchOperations that represent the search criteria

- Output Columns – a collection of columns that represent the output of the search
- Sort Columns – a collection of columns that represent the sort order

Because Criteria can contain SearchOperationGroups, you have the ability to design parenthetical queries. Consider the following MSMQL statement:

- Select FirstName, LastName from Individual where Email like '%gmail%' or Age> 15 and Gender_c= 'Male' order by CreatedDate

Note that we have a custom field in our MSMQL, which is perfectly legal. We know this because it ends in "__c". The corresponding search object would be created like this:

```
Search s =  new  Search();
s.AddOutputColumn( "FirstName" );
s.AddOutputColumn( "LastName" );

s.AddCriteria(Expr.Equals( "Email" ,  "gmail" ));

 // now, here's our parenthesis, which is its own group
SearchOperationGroup sog =  new  SearchOperationGroup();
sog.Criteria.Add(Expr.IsGreaterThan( "Age" ,  15 ));
sog.Criteria.Add(Expr.Equals( "Gender__c" , "Male" ));

s.AddCriteria(sog);  // add the parenthetical statement

 //  add the sort column
s.AddSortColumn( "CreatedDate" );
```

Objects in MemberSuite

Tenancy in MemberSuite

In MemberSuite, there is a difference between a customer and an association. A customer is a billing entity – we have a contractual relationship with the organization represented by this record. An Association is a "tenant" in our system; it's a repository of objects that comprise an association. One customer usually has one association, but there are exceptions to this, like when a customer is in the business of managing multiple associations (Association Management Companies), or when sandbox associations are created under the customer.

Thus, the 99% of all objects in MemberSuite live in an association.

How Objects Work in MemberSuite

MemberSuite is inherently an object-oriented design; we use object to encapsulate state and behavior. There is a special class of objects that correspond to the functionality of the software called domain objects. Domain objects borrow from Eric Evan's Domain Driven Design – by definition, these objects should represent pieces of the domain that a domain expert (or subject matter expert) can understand. Consider the following:

We can see the concept of an Individual and an organization are both types of Entities. An Entity can own an order – which means either an individual or an organization can define an order.

These concepts are easily readable and verifiable by a non-programmer domain expert, which is the very point of domain objects. Domain objects represent the key concepts of the problem an application is trying to solve.

MemberSuite has over 200 domain objects, spanning over 20 different modules, or namespaces. Examples of these objects:

- Individual
- Invoice
- Order
- Event
- Competition

Object Identification

Each object has a globally unique identifier, or GUID. A GUID is a 32-byte value that is designed to be unique across all time and space. For more information on GUIDs, look at:

http://en.wikipedia.org/wiki/Globally_unique_ident...

The GUID ensures that we can locate an object across multiple databases, and across time and space. The GUID has a specific format that allows us, from a GUID, to determine the association that the object belongs to and the type of objects.

The first 4 bytes of the GUID are called the *association hint*. This identifies the association the object belongs to. As a rule, these 4 bytes match the 4 bytes of the target association. In this example, MemberSuite determines what association this object belongs to so by finding an association whose ID belongs with **594811f7**.

The second two bytes are called the *GUID hint*. This tells MemberSuite what type of object it is. In this case, **0006** indicates that the object is of type **Individual**. The Data Dictionary reference included in this documentation will list the 2-byte GUID hint for each object.

Aggregates

Borrowing further from Domain Driven Design, we define an aggregate as an object boundary. There are several important reasons for this. Consider an object, Invoice. An invoice has multiple InvoiceLineItem objects. But invoices have rules; one invoice must have at least one-line item. Additionally, line items must add up to the invoice total.

If you can modify InvoiceLineItem records directly, how would one enforce these rules? It would quickly become difficult, as you would have to write logic to check for other invoice line items when saving individual line items.

Conversely, we can make a rule for ourselves; you cannot save InvoiceLineItems directly, you must save them within the context of an Invoice. This solves our problem; since invoice line items can't be referenced or saved directly, we can rely on the Invoice object to apply any validation rules necessary. Thus, the Invoice, and the associated InvoiceLineItems, form an aggregate boundary. The Invoice is known as an aggregate. The InvoiceLineItem is known as an aggregate child.

Aggregates have the following properties:

1. All aggregates in MemberSuite have IDs that can be referenced directly

2. Aggregates can contain non-aggregates but cannot contain direct links to other aggregates. If they point to other aggregates (i.e., the Order.BillTo pointing to an Entity), they must reference the ID, instead of the whole object itself
3. Aggregates can perform their own validations
4. Aggregates must travel together across the API boundaries – so when you pull an Invoice, you must get its invoice line items.

AssociationDomainObjects vs. The Others

Now, consider this object model:

As you can see, the fundamental concept of MemberSuite is a DomainObject. From there, we have aggregates and aggregate children. Under aggregate, we've got objects that live in an association (remember that 99% of objects do), and then objects that live outside of an association.

Think of MemberSuite as your apartment building. You live in an apartment. So does your neighbor. So does his. Almost everyone lives in an apartment building, except for the super-intendent. The super has his own office that isn't an apartment, but instead supports all the other apartments.

In MemberSuite, each association is like an apartment. Almost everything lives in an association, or an apartment. A few things don't – they live like the super, and in MemberSuite the super's office is called the catalog. Catalog objects are global across associations and live in a special database called – you guessed it – the Catalog.

Examples of catalog objects:

- Customer
- User

Platform: WhoAmI

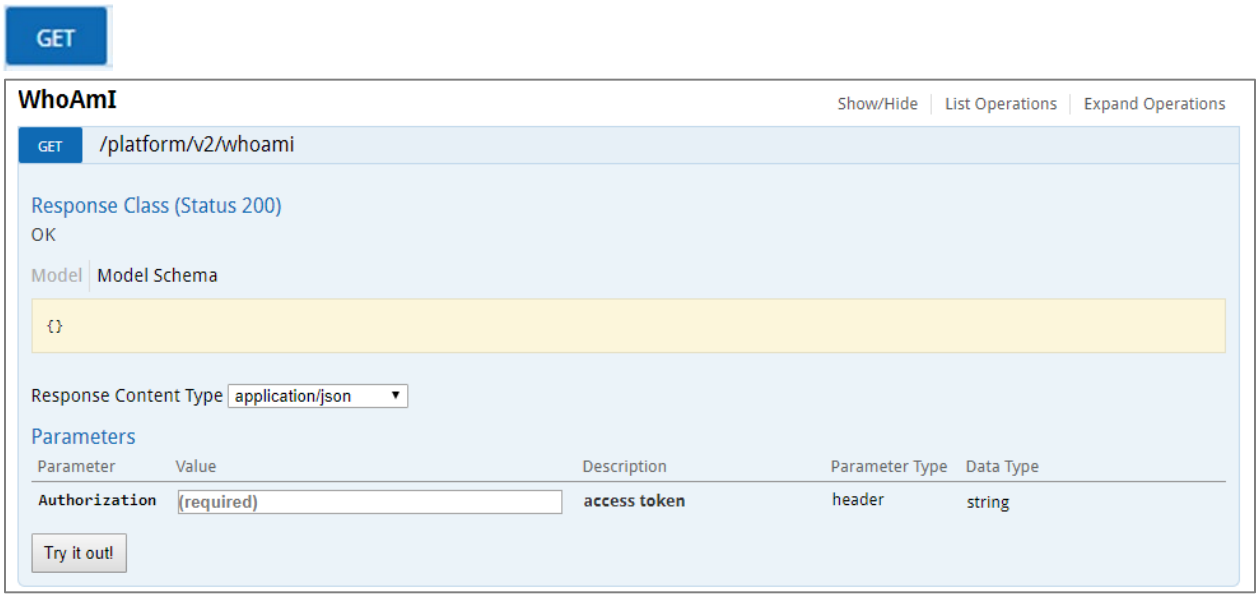
The WhoAmI object of the Platform endpoint within the REST API contains only a GET operation. An authorization token obtained through the security endpoint as described earlier in the credentials section is all that is required to complete the GET call within the WhoAmI segment of the platform endpoint.

Input Parameters:

Authorization: Base64 Encoded JWT.

Response Body:

{
"tenantId": 27969,
"associationId": "8902ce52-0004-ca0d-7305-0b3dd2c24826",
"userId": "00000000-0032-c171-d271-0b3f653a5faa",
"email": "dude00701@yopmail.com",
"firstName": "fghd",
"lastName": "sfgsd",
"ownerId": "8902ce52-0006-c5ea-4962-0b3f653a5fa1",
"membershipId": null,
"receivesMemberBenefits": false,
"paymentProcessor": "BluePay",
"businessUnit": "8902ce52-0034-c836-158e-0b3c94884698",
"merchantAccount": "8902ce52-01f8-c76e-c7d2-0b3c948886cd",
"baseCurrency": "USD"
}



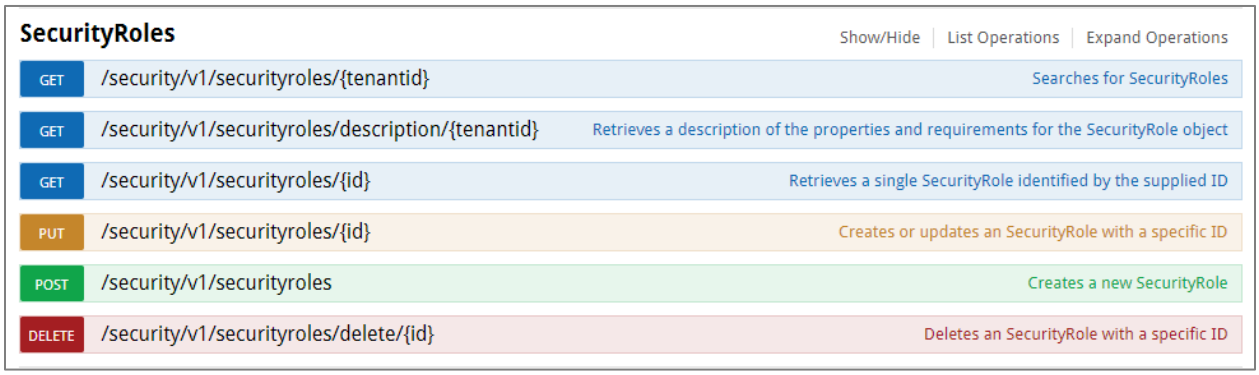
Security

The security endpoint contains an extensive list of operations to choose from when making a call to the user's environment. Like the platform endpoint, the user can use these operations to make a POST, GET, PUT, or DELETE call to the environment. The types of calls available varies depending upon the chosen operation. Each call within the security endpoint includes a description of what a correctly entered call will return. The security endpoint will be used to access and change the roles and policies of the user's environment. A few examples of what can be done using the security endpoint have been included below.

https://rest.membersuite.com/security/swagger/ui/index#/

Security: SecurityRole

A security role defines how different users, access different types of records. To control access to data, you can modify existing security roles, create new security roles, or change which security roles are assigned to each user. Each user can have multiple security roles.



Security: PortalUsers

https://rest.membersuite.com/security/swagger/ui/index#/PortalUsers

One of the diverse endpoints that combines the security with the portal users. This endpoint fulfills various task, few of the important ones are as follows:

- Creating a new portal user.
- Getting the attributes related to user.
- Retrieving a description of the properties and requirements for the portal user.
- Retrieving a single portal user identified by the supplied ID
- Searching a portal user.

First Searching Id's for all the available portal users.

GET

Input Parameters:

Msql: SELECT id FROM PortalUser

Page: 0

pageSize: 0

tenantId: 27969

Authorization: Base64 Encoded JWT.

Response Body: List of the asked values in the msql query.

[
 {
  "roW_NUMBER": 1,
  "id": "8902ce52-0032-c328-401f-0b38ce976fbf",
  "iD1": "8902ce52-0032-c328-401f-0b38ce976fbf"
 }

.

.

]

Now, after getting all the listed portal user. One can retrieve all the details specific to the supplied Id.

GET

Input Parameters:

id: GUID representing the individual Id.

Authorization: Base64 Encoded JWT.

Response Body:

{
"classType": "PortalUser",
"parentTypes": [
"User",
"SystemDomainObject",
"CatalogAggregate",
"Aggregate",
"DomainObject"
],
"owner": "8902ce52-0006-c88b-d57f-0b38ce976f65",
"lastLoggedInAs": "8902ce52-0006-c88b-d57f-0b38ce976f65",
"securityLock": null,
"firstName": "Sample",
"lastName": "Portal User",
"emailAddress": "apak@membersuite.com",
"passwordHash": "Gdnm3qhi9XWaMIy+WFTu2LsIgV8f29cPLXQETjbJPZ3eCQRS",
"title": null,

"department": null,
"isActive": true,
"isSuperUser": false,
"timeZone": "Eastern Standard Time",
"lastAssociation": null,
"lastLoggedIn": "2019-02-07T03:29:08",
"firstLoginDate": "2016-06-20T14:14:16",
"loginFailures": 0,
"phoneNumber": null,
"notes": null,
"securityQuestion": null,
"securityAnswer": null,
"mustChangePassword": false,
"locale": null,
"baseCurrency": null,
"id": "8902ce52-0032-c328-401f-0b38ce976fbf",
"name": "5ive",
"keywords": null,
"lastModifiedBy": "00000000-0003-cbee-31c4-0b3e14a575f9",
"lastModifiedByName": "asawyerr",
"lastModifiedDate": "2019-02-08T11:38:27",
"createdDate": "2014-07-25T14:04:01",
"createdBy": "00000000-0003-42c0-b8c7-523d782a761a",
"createdByName": "bwest",
"lockedForDeletion": false,
"isConfiguration": false,
"isSealed": false,
"systemTimestamp": "AAAAAAWbifQ=",
"existsInNoSQL": false}

CRM

CRM stands for Customer Relationship Management. At its simplest definition, a CRM system allows businesses to manage business relationships and the data and information associated with them.

With CRM, you can store customer and prospect contact information, accounts, leads, and sales opportunities in one central location, so the information is accessible by many, in real time. A CRM can expand to include sophisticated features to help teams collaborate with colleagues and customers, send customized emails, gather insights etc.

An example showing the updating and retrieval of the updated value for the individual.

PATCH

Updates the specified field for the specified individual.

Input Parameters:

id: GUID representing the individual Id.

payload:

```
{
        "Work_PhoneNumber": "0123456787"
}
```

Response Body: 200 Status Code

GET

Fetches all the data related to the provided individual id. Here we can see that the above updated work phone number is updated and reflected in real time.

Input Parameters:

id: GUID representing the individual Id.

Authorization: Base64 Encoded JWT.

Response Body:

{
"classType": "Individual",
"firstName": "sdf",

.

.

.

"work_PhoneNumber": "(012) 345-6787",

.

.

. }