MARIGOLD™

API Technical Guide: Batch Import

Cheetah Messaging

Table of Contents

1 Introduction

Purpose

The purpose of this document is to provide an overview of the BATCH IMPORT API endpoint within the Cheetah Messaging platform. This document discusses the intended use of the BATCH IMPORT endpoint, and provides technical details for how to implement the endpoint.

Overview

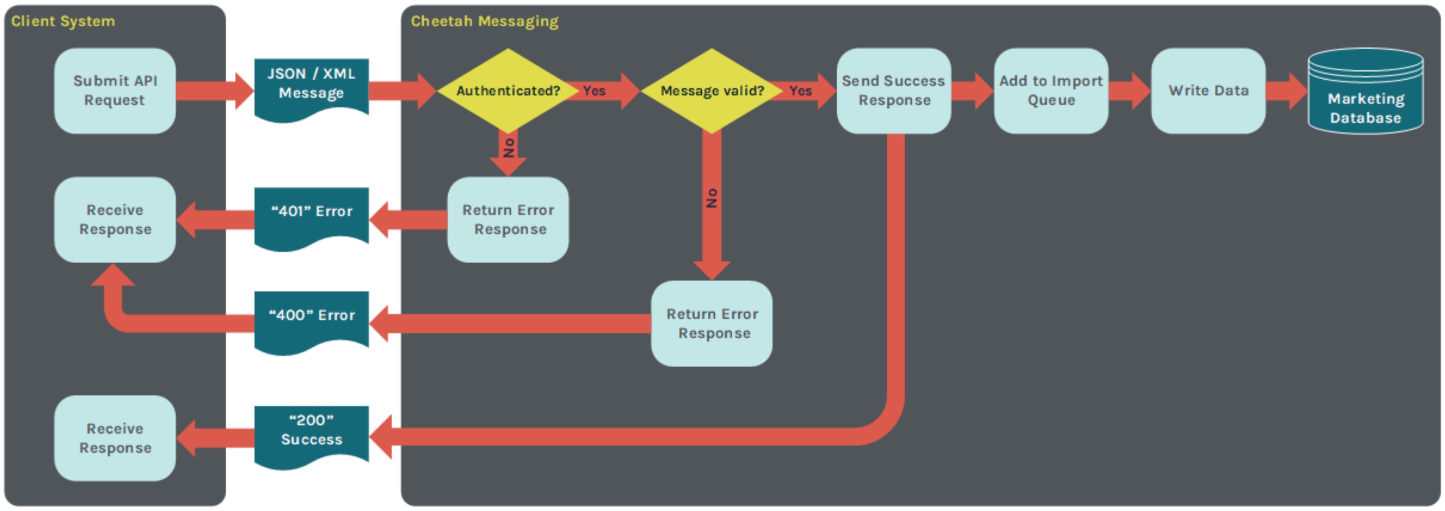
The BATCH IMPORT endpoint allows you to send Base64-encoded records to load into, or to update, your Messaging database. When the API message is received, the records are added to the import file queue, and are processed based on the priority indicated within the API request. You can also use this endpoint to view, edit, and delete existing Manual Imports.

This endpoint requires authentication using OAuth 2.0, and supports JSON and XML messages.

The URLs for this endpoint are:

- **North America**: https://api.eccmp.com/services2/api/Import
- **Europe**: https://api.ccmp.eu/services2/api/Import
- **Japan**: https://api.marketingsuite.jp/services2/api/Import

The following diagram depicts the basic processing flow for loading data via the BATCH IMPORT endpoint.



Pre-requisites

In order to use the BATCH IMPORT endpoint to write data to your database, the following assets must be defined within your Messaging account:

- **Table** -- The destination Table where you want to load your import data must be created, and must include all the necessary Fields, as well as any necessary Joins to other Tables. These Tables, Fields, and Joins can be created within the Messaging application.
- **Data Map** -- The Data Map provides data handling instructions, such as where to store the inbound data contained within the API request message, whether this data should be used to update existing records and / or create new records, and any optional formatting or special processing to perform on the inbound data. The Data Map can be created within the Messaging application, or through the use of the data map API endpoint.

In addition, the BATCH IMPORT endpoint supports the use of the following optional assets. If you intend to use these optional assets during the import process, they must be defined with your Messaging account:

- **Hierarchy Rule** -- Hierarchy Rules are typically utilized by clients who have a Parent / Child database architecture. This architecture is designed around a single database (referred to as the "Parent" system) with multiple brands residing in that database, separated into different views (each of which is referred to as a "Child" system). Hierarchy Rules are a top-down configuration feature set in the Parent system that determine which records a Child system can access. Hierarchy Rules can be created within the Messaging application.

- **NCOA Proce**ss -- NCOA Processes are typically utilized by clients running Print Campaigns. National Change Of Address (NCOA) is a process that submits a list of postal addresses to a service which maintains updated addresses for individuals who have filed a Change of Address with the U.S. Post Office. This process provides updated addresses for these individuals, thereby resulting in reduced postage costs as fewer pieces of mail are returned to the postal service. NCOA Processes can be created within the Messaging application.

Methods

The BATCH IMPORT endpoint supports the following HTTP methods:

- **POST**: Write the data in the request message to the database.
- **GET**: Retrieve information about an existing Manual Import.
- **PUT**: Update an existing Manual Import.
- **DELETE**: Delete a Manual Import.

Authentication

Access to the BATCH IMPORT endpoint requires that you first be authenticated within the platform. Within Messaging, authentication is handled by OAuth 2.0. To authenticate with OAuth 2.0, you must first obtain a "Consumer Key" and a "Consumer Secret." Both of these values are managed at the user level, and can be obtained from within the Messaging application.

Next, you'll use your Consumer Key and Consumer Secret to request a "token." A token is a text string that, when provided in a request message, will allow the user access to the requested service. Tokens are valid only for a certain period of time.

For more details on how to authenticate your API request, please see the *Messaging: API How-to Guide*.

2 Create a Manual Import

Overview

This section describes how to write to your database, via a POST request to the BATCH IMPORT endpoint.

Parameters

The options and parameters described in this section explain how to create a new Manual Import containing data that you want to write to your database.

cust_id

This integer parameter is required.

The **cust_id** parameter represents the Customer ID of your Messaging account. The Customer ID is a unique, system-generated identifier for every Messaging client account. This value isn't displayed anywhere within the Messaging application, so you must retrieve it by means of an API request, or speak to your Client Services Representative, who can proivde you with this value.

Example:

```
"cust_id": 394
```

prop_map_id

This integer parameter is required.

The **prop_map_id** parameter represents the Object Reference ID of the Data Map that you want to use to control how, and where, the inbound data is written.

Example:

```
"prop_map_id": 4569
```

type_id

This string parameter is required.

For a Manual Import, the value of the **type_id** parameter must be "FILE."

Example:

```
"type_id": "FILE"
```

import_file

This string parameter is optional.

The **import_file** value is used as a suffix to a system generated file name in the platform's back-end processing.

This value isn't displayed anywhere within the application. For audit and tracking purposes, the best practice is to populate this parameter with the original filename. In this manner, if any questions arise about the source of the import, you can more easily trace the request message back to the original file that contained the inbound data.

Example:

```
"import_file": "api_test_import"
```

encoding

This string parameter is required.

The **encoding** parameter instructs the system what character encoding method to use on the inbound data. The valid values for this parameter are:

- UTF8
- WesternEuropean_ISO88591
- Greek_ISO88597
- Japanese_ShiftJIS

Example:

```
"encoding": "UTF8"
```

file

This string parameter is required.

The **file** parameter is used to pass data through the API. This parameter must contain your entire import file as a single Base64-encoded string.

Note

You can use free, online Base64-encoding tools, such as:
https://www.base64encode.org.

Example:

```
"file":
"RW1haWwsRmlyc3ROYW1lLExhc3ROYW1lDQpqb2huLmRvZUBjaGVldGFoZGlnaXRhbC5jb
20sSm9obixEb2UNCm1hcnkuc21pdGhAY2hlZXRhaGRpZ2l0YWwuY29tLE1hcnksU21pdGg
NCmhvbWVyLnNpbXBzb25AY2hlZXRhaGRpZ2l0YWwuY29tLEhvbWVyLFNpbXBzb24NCg=="
```

hierarchy_id

This integer parameter is optional.

The **hierarchy_id** parameter represents the **Object Reference ID** of a Hierarchy Rule. Hierarchy Rules are used in Parent / Child database architectures to determine what fields each "Child" view can access.

Example:

```
"hierarchy_id": 21
```

ncoa_process_id

This integer parameter is optional.

The **ncoa_process_id** parameter represents the **Object Reference ID** of an NCOA Process. NCOA Processes are used by clients running Print Campaigns, to run postal addresses through the National Change of Address system.

Example:

```
"ncoa_process_id": 21
```

task_priority

This integer parameter is optional.

The **task_priority** parameter allows you to establish the priority for your import. The priority you select determines the order in which your import will be handled. Generally speaking, your imports with a higher priority will take precedence over your imports with a lower priority. If your imports have the same priority, they will be processed in the order in which they were added (older imports first). The valid values for this parameter are:

- "300:" Low
- "700:" Normal (this is the default value used if you don't provide this parameter)
- "800:" High
- "1000:" Urgent

obj

This object contains the name and location of the new Manual Import.

Example:

```
"obj":
  {
  "display_name": "Manual Import API",
  "parent_obj_id": 37249
  }
```

The parameters in **obj** are described below in more detail.

display_name

This string parameter is required.

The **display_name** parameter contains the name of the Manual Import, as it appears within the Messaging application. This name must be unique within the specified folder.

Example:

```
"display_name": "Manual Import API"
```

parent_obj_id

This integer parameter is required.

The **parent_obj_id** parameter represents the Folder ID of the folder where you want to save the new Manual Import.

Example:

```
        "parent_obj_id": 37249
```

3 Edit a Manual Import

Overview

This section describes the options for viewing, modifying, and deleting Manual Imports via the BATCH IMPORT endpoint.

Retrieve a Manual Import

The GET method is used to retrieve all of the information about a specified Manual Import. The response message provides details of the Manual Import, and shows the results of the import process, such as number of records parsed, cleansed, etc.

The platform parses the import data to ensure that the values are clean, valid, and usable. Depending on the Data Type for a field, various methods of parsing and cleansing are utilized. The platform may adjust the import value for a field according to rules set for that type of field. For example, if a numeric value is being imported into a field identified as a "Phone Number," the platform will verify that this value is the correct length, and will remove any unnecessary and invalid characters. For more details on the Data Parsing rules utilized by Messaging, please see the Messaging Online Help system.

When submitting a GET request to the BATCH IMPORT endpoint, the Object Reference ID must be provided as a query type parameter in the URL, not in the body. For example:

```
https://api.eccmp.com/services2/api/Import?id=3456
```

Delete a Manual Import

The DELETE method is used to delete a specified Manual Import.

When submitting a DELETE request to the BATCH IMPORT endpoint, the Object Reference ID must be provided as a query type parameter in the URL, not in the body. For example:

```
https://api.eccmp.com/services2/api/Import?id=3456
```

Edit a Manual Import

The PUT method allows you to submit modifications to an existing Manual Import. However, once a given import process is complete, and the inbound data has been written to the database, the options for updating the Manual Import are very limited -- you can change only the name of the Manual Import, but not the Data Map or the data itself.

When submitting a PUT request to the BATCH IMPORT endpoint, the **Object Reference ID** must be provided as a query type parameter in the URL, not in the body. For example:

```
https://api.eccmp.com/services2/api/Import?id=3456
```

import_id

This integer parameter is required.

In addition to being sent as a query type parameter in the URL, the **Object Reference ID** must also be included within the message body using the parameter **import_id**. The value in **import_id** must match the value provided in the URL.

For example:

```
"import_id": 3456,
```

The other parameters for the PUT method are the same as described in the **Create a Manual Import** section.

4 Response

This section describes the possible response messages sent back from the BATCH IMPORT endpoint.

Success

A successful response to a POST message to create a new Manual Import will generate a response code of "200," followed by the details of the new Manual Import contained within the body of the response message.

A successful response to a GET message to retrieve a Manual Import will generate a response code of "200," followed by the details of the specified Manual Import contained within the body of the response message. If the import file has completed processing, the response message also contains the details of the import parsing and database update steps. The message lists a variety of error conditions, along with the number of records that encountered that condition. These conditions are as follows:

- "FILE_ROWS" -- Total number of records in the file.
- "EMPTY_ROWS" -- No data in the row, nothing to import.
- "WARNINGS" -- The system identified missing, empty or unmapped columns, but the row will be imported.
- "BAD_EMAILS" -- Email field matches a cleansing rule but doesn't have a replacement. Email was set to null. Email column will not be imported.
- "FIXED_EMAILS" -- Email field matches a cleansing rule. Email field was replaced based on predefined rules.
- "BAD_PHONES" -- Phone Number column will not be imported.
- "FIXED_PHONES" -- Phone Numbers were cleaned (if possible).

- "ERRORS" -- The value provided in the file of a Unique ID or Primary Key field is empty or NULL.
- "EMAIL_BAN_HARD_LOCAL" -- Email previously encountered a hard bounce condition (client-specific).
- "EMAIL_BAN_HARD_COMMON" -- Email previously encountered a hard bounce condition (global).
- "EMAIL_BAN_SOFT_LOCAL" -- Emails previously encountered a soft bounce condition.
- "EMAIL_BAN_MASK_LOCAL" -- Email address is on the custom Banned Email list.
- "EMAIL_BAN_MASK_COMMON" -- Email address is on the Global Banned Email list.
- "BAD_AK_IDS" -- The Unique ID was missing information; the record will not be updated/created.
- "CONFLICT_AK_IDS" -- The Unique ID in the file is already associated with a different Primary Key for the Entity.
- "DUP_AK_IDS" -- The same Unique ID appeared on two or more records on the file; the first record will be updated/created, the remaining duplicate records will not be updated/created.
- "SWITCH_AK_IDS" -- The number of Unique IDs that were switched.
- "OLD_AK_IDS" -- The number of old Unique IDs found.
- "SUP_NEW_PK_IDS" -- The number of new recipient Primary Keys that were suppressed.
- "UPD_PK_IDS" -- The number of existing recipient Primary Keys.

A successful response to a PUT message to update a Manual Import will generate a response code of "200," followed by the details of the modified Manual Import contained within the body of the response message.

A successful response to a DELETE message will generate a response code of "204;" the body of the response message will be empty.

Errors

If Messaging encounters a problem with a BATCH IMPORT request message, the platform will send an error message with details of the problem. Below is a list of error codes and their descriptions.

Respons e Code	Error message	Description
400	"An Id is required when updating an object, Ids must match on the updated object."	For a PUT request, required parameter **import_id** is missing, or contains an invalid value, or doesn't match the **id** value in the URL.
400	"An Obj with this name already exists"	A Manual Import with this same name already exists in this folder; the **display_name** value must be unique within a folder.

5 Sample Messages

This section contains a sample request and response message for the BATCH IMPORT endpoint.

Request Message

This sample POST request message loads data into the client's Messaging database.

The user has already created the necessary Data Map that defines the table into which the data should be loaded, and how each column in the import maps to a column in the database table.

The data to be loaded consists of three records (and a header row). The user is importing three fields -- email address, first name, and last name.

```
Email,FirstName,LastName
john.doe@cheetahdigital.com,John,Doe
mary.smith@cheetahdigital.com,Mary,Smith
homer.simpson@cheetahdigital.com,Homer,Simpson
```

The user must Base64-encode this data, rather than sending the raw, unencoded data.

JSON Payload

```
{
  "cust_id": "394",
  "prop_map_id": "4593",
  "type_id": "FILE",
  "import_file": "api_test_import",
  "encoding": "UTF8",
  "file": "
RW1haWwsRmlyc3ROYW1lLExhc3ROYW1lDQpqb2huLmRvZUBjaGVldGFoZGlnaXRhbC5jb2
0sSm9obixEb2UNCm1hcnkuc21pdGhAY2hlZXRhaGRpZ2l0YWwuY29tLE1hcnksU21pdGgN
CmhvbWVyLnNpbXBzb25AY2hlZXRhaGRpZ2l0YWwuY29tLEhvbWVyLFNpbXBzb24NCg=="
  "Obj": {
    "display_name": "Batch Import API Test",
    "parent_obj_id": 37249
  }
}
```

Response Message

This sample response message shows the results of a GET request used to retrieve information about an existing Manual Import. The response message provides details of the Manual Import, and shows the results of the import process, such as number of records parsed, cleansed, etc.

JSON Payload

```
{
    "import_id": 6169,
    "import_name": "Batch Import API Test",
    "cust_id": 394,
    "prop_map_id": 4593,
    "type_id": "FILE",
    "status_id": "DONE",
    "import_file": "2018-02-07_20-20-50_9586527.api_test_import",
    "importPreviewFlags": {
        "import_id": 6169,
        "skip_loaded_preview": 1,
        "skip_parsed_preview": 1,
        "skip_stats_preview": 1
    },
    "importStat": {
        "import_id": 6169,
        "parsing_start_time": "2018-02-07T20:19:59.583",
        "parsing_finish_time": "2018-02-07T20:20:00.48",
        "update_start_time": "2018-02-07T20:20:07.633",
        "update_finish_time": "2018-02-07T20:20:11.177",
        "importStatMetrics": [
            {
                "metric_val": 0,
                "import_id": 6169,
                "type_id": 1000
            },
            {
                "metric_val": 3,
                "import_id": 6169,
                "type_id": "FILE_ROWS"
            },
            {
                "metric_val": 0,
                "import_id": 6169,
                "type_id": "EMPTY_ROWS"
            },
            {
                "metric_val": 0,
                "import_id": 6169,
                "type_id": "WARNINGS"
            },
            {
```

```
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "BAD_EMAILS"
    },
    {
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "FIXED_EMAILS"
    },
    {
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "BAD_PHONES"
    },
    {
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "FIXED_PHONES"
    },
    {
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "ERRORS"
    },
    {
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "EMAIL_BAN_HARD_LOCAL"
    },
    {
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "EMAIL_BAN_HARD_COMMON"
    },
    {
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "EMAIL_BAN_SOFT_LOCAL"
    },
    {
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "EMAIL_BAN_MASK_LOCAL"
    },
    {
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "EMAIL_BAN_MASK_COMMON"
    },
    {
        "metric_val": 0,
        "import_id": 6169,
        "type_id": "BAD_AK_IDS"
    },
    {
```

```
                    "metric_val": 0,
                    "import_id": 6169,
                    "type_id": "CONFLICT_AK_IDS"
                },
                {
                    "metric_val": 0,
                    "import_id": 6169,
                    "type_id": "DUP_AK_IDS"
                },
                {
                    "metric_val": 0,
                    "import_id": 6169,
                    "type_id": "SWITCH_AK_IDS"
                },
                {
                    "metric_val": 2,
                    "import_id": 6169,
                    "type_id": "OLD_AK_IDS"
                },
                {
                    "metric_val": 1,
                    "import_id": 6169,
                    "type_id": "SUP_NEW_PK_IDS"
                },
                {
                    "metric_val": 2,
                    "import_id": 6169,
                    "type_id": "UPD_PK_IDS"
                }
            ]
        },
        "obj": {
            "obj_id": 51675,
            "display_name": "Batch Import API Test",
            "type_id": "Import",
            "ref_id": 6169,
            "parent_obj_id": 37249,
            "eligibility_status_id": "READY"
        }
    }
```

6 Appendix -- Identifiers

Messaging uses several different types of IDs when referencing assets, such as tables, fields, folders, Filters, and so forth. This appendix describes these different types of IDs, and provides steps on how to look up the value of an ID.

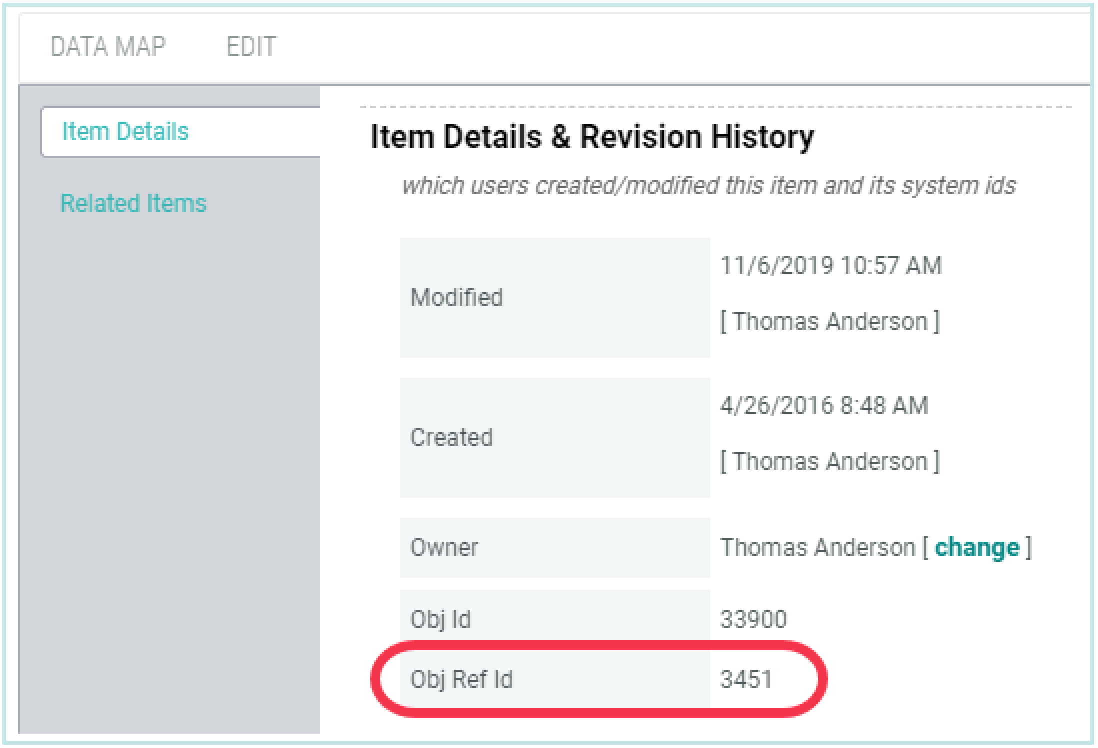
Object Reference ID

The Object Reference ID is a system-generated identifier for every item and asset in your account.

The value for this identifier can be found within the Messaging application, or by using the SEARCH endpoint, which will return the Object Reference ID in the response message.

To find an asset's Object Reference ID within the application:

1. From the System Tray, navigate to the desired screen for this asset type.
2. Select the desired asset. The asset's Details screen is displayed.
3. In the Tool Ribbon, click the the first tab; this tab is usually named after the asset type, such as "Data Map" for example. The "Item Details" screen is displayed. The Object Reference ID is listed on this screen.


DATA MAP EDIT

Item Details

Related Items

Item Details & Revision History

which users created/modified this item and its system ids

Modified	11/6/2019 10:57 AM [Thomas Anderson]
Created	4/26/2016 8:48 AM [Thomas Anderson]
Owner	Thomas Anderson [**change**]
Obj Id	33900
Obj Ref Id	3451

Optionally, for many asset types, you can use the SEARCH endpoint, and search for the desired asset:

1. Submit a GET request to the SEARCH API endpoint. The simplest method is to use the versions of the SEARCH endpoint that allow you to retrieve information based on either the asset's name or its type. For example, to retrieve information about all of your Data Maps:

```
https://api.eccmp.com/services2/api/Object?type=PropertyMap
```

2. The response message provides a list of all the assets in your system that match the search criteria. Find the desired asset in the response message.

3. As part of the API response message, the system provides the Object Reference ID, which is referred to as the **ref_id**. For example:

```
{
  "obj_id": 44737,
  "display_name": "Recipient Data Map",
  "type_id": "PropertyMap",
  "ref_id": 40329,
  "parent_obj_id": 43269,
  "eligibility_status_id": "READY"
}
```

Folder ID

The Folder ID is a unique, system-generated identifier for each folder and sub-folder in your system. This value is not displayed within the application user interface anywhere, so to get your Folder ID, you must retrieve it by means of the SEARCH API endpoint.

1. Submit a GET request to the SEARCH endpoint. The easiest method is to use the version that lets you search by object type -- use a type value of "Folder." For example:

```
https://api.eccmp.com/services2/api/Object?type=Folder
```

2. The response message provides a list of all the folders in your system. Find the desired folder in the response message.

3. As part of the API response message, the system provides the Folder ID, which is referred to as the **obj_id**.

Note

If this Folder is a sub-folder, the **parent_obj_id** is the Folder ID of the parent folder.

Sample Response:

```
{
  "obj_id": 37465,
  "display_name": "Content Block Folder",
  "type_id": "Folder",
  "ref_id": 37465,
  "parent_obj_id": 22817,
  "eligibility_status_id": "READY"
}
```