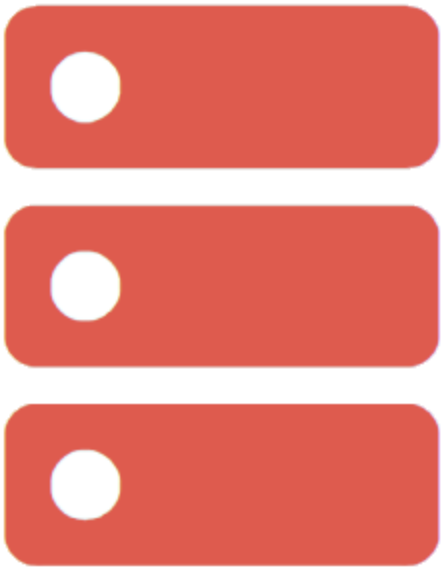
MARIGOLD™

API Technical Guide: Standard Data Load

Cheetah Messaging

Table of Contents

1 Introduction



Purpose

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

Overview

The STANDARD DATA LOAD endpoint is used to submit and write data, one record at a time, to one or more joined tables, in your Messaging database. The API request can also optionally be used to trigger the deployment of a Campaign.

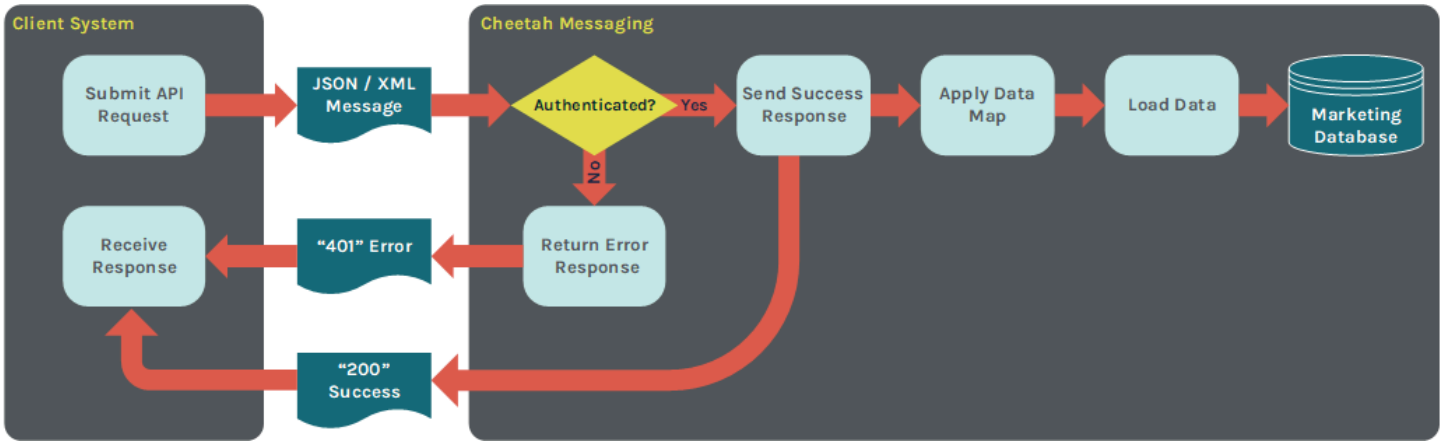
This endpoint requires authentication using OAuth 2.0, and supports JSON and XML messages.

The URLs for this endpoint are:

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

> **Note**
> This endpoint is sometimes referred to as the "Recipients" endpoint, but that name can be misleading. This endpoint can be used to load data into any table, not just into your "Recipient" table.

The following diagram depicts the basic processing flow for loading data via the STANDARD DATA LOAD endpoint.

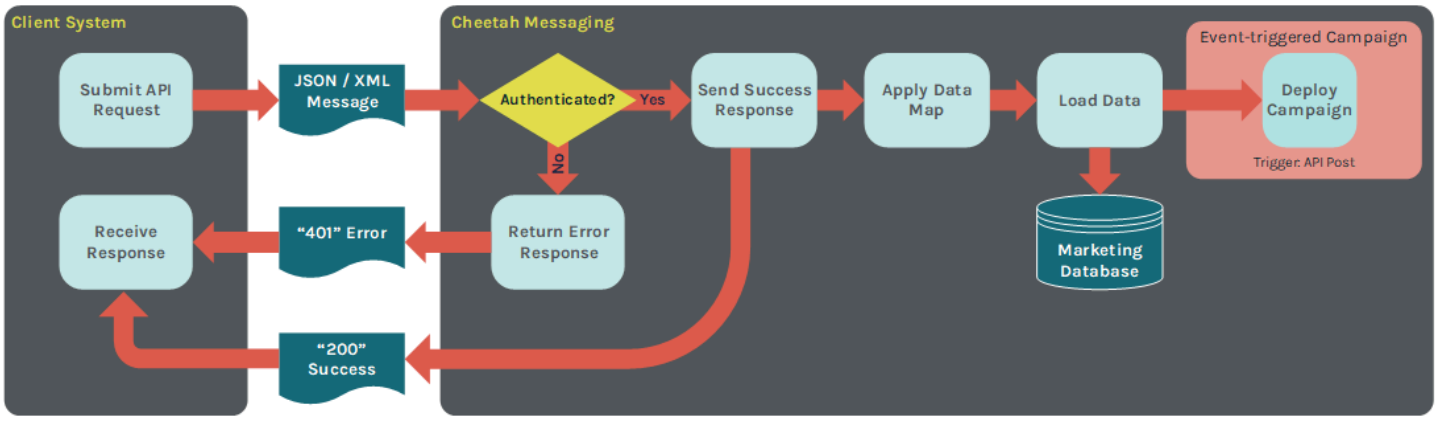


Campaign Trigger

The STANDARD DATA LOAD endpoint can optionally be used as a triggering mechanism for the deployment of an Event-triggered Campaign. The Campaign is deployed after the data in the request message is successfully written to the database, which allows you to use data from your database when building your message content.

When this endpoint is utilized in this fashion, it's typically referred to as the STANDARD EVENT TRIGGER.

The following diagram depicts the basic processing flow for loading data and deploying a Campaign via the STANDARD EVENT TRIGGER endpoint.



Pre-requisites

The STANDARD DATA LOAD endpoint requires that the following assets be defined within your Messaging account:

- **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 endpoint.
- **API Post** -- The API Post defines all the expected fields in the request message. The API Post must be created within the Messaging application. Please note that the API Post must have the option "REST API Only" checked, and the API Post must be published.

If you're using this endpoint to trigger the deployment of an Event-triggered Campaign, you must have the following asset defined as well:

- **Campaign** -- You must define and launch an Event-triggered Campaign that uses the above API Post as the event trigger.

Methods

The STANDARD DATA LOAD endpoint supports the following HTTP method:

- **POST**: Write the data in the request message to the database.

Authentication

Access to the STANDARD DATA LOAD 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 Load Data

Overview

This section describes how to write data to your database using a POST request to the STANDARD DATA LOAD endpoint.

Parameters

Parameters are the required or optional information contained within the API request message, and tell the system what data to write to the database, along with several other processing options.

apiPostId

This integer parameter is required.

The **apiPostId** parameter represents the **Form ID** of the API Post.

For example:

```
"apiPostId": "2691"
```

enable_validation

This Boolean parameter is optional. If not provided, the system will default this parameter to "false."

If **enable_validation** is set to "true," the system will validate the data in the request message (see **Data Validation** below for more details), prior to loading the data to the database. If this validation fails, the data will not be written to the database, and the response message will contain the appropriate error message.

If set to "false," the system will not perform this data validation step.

Cheetah Digital recommends setting this parameter to "true."

You can use this parameter instead of, or in conjunction with, the **validate_on_error** parameter described below.

For example:

```
"enable_validation": "true"
```

validate_on_error

This Boolean parameter is optional. If not provided, the system will default this parameter to "false."

If **validate_on_error** is set to "true," the system will attempt to load all data into the database without first validating it. If the system encounters any errors when loading the data, the system will then run the data validation process (see **Data Validation** below for more details), and the response message will contain the appropriate error message.

If set to "false," the system will not perform this data validation step.

Cheetah Digital recommends setting this parameter to "true."

You can use this parameter instead of, or in conjunction with, the **enable_validation** parameter described above.

For example:

```
"validate_on_error": "true"
```

data

This array parameter is required.

The **data** array contains the data you want to write to your database. The data is sent as key / value pairs consisting of a field's **Column Name** in the **name** parameter, and the corresponding value in the **value** parameter.

For example:

```
"data":[
    {
      "name": "email",
      "value": "john.doe@cheetahdigital.com"
    },
    {
      "name": "name_first",
```

```
    "value": "John"
  },
  {
    "name": "name_last",
    "value": "Doe"
  }
]
```

Data Validation

If you enable either of the data validation parameters described above, the system validates the data in the request message using the following rules:

Ambiguous or duplicate data validations:

- Check for duplicates in the incoming data.
- Check if the given database field names exist in the database.

Primary Key ID / Unique Identifier (i.e., "Alternate Key") validations:

- Check if the Primary Key ID or Unique Identifier field is provided.
- Check if the provided Primary Key ID or Unique Identifier for the main table is not empty.
- Check if the provided Primary Key ID or Unique Identifier for a joined table is not empty.

Data content validations:

- For fields of data type "String," "Email," "Push Registration ID," and "LINE Contact MID," the provided value is not longer than 255 bytes.
- For fields of data type "Long String," the provided value is not longer than 8000 bytes.
- For fields of data type "Money / Decimal," the provided value is between -9223372036854775808 through 9223372036854775807.
- For fields of data type "Date / Time," the provided value is within the valid date range.

- For fields of data type "Email," the provided value has the right format.
- For fields of data type "Phone," the provided value has the right format.
- For fields of data type "Big Integer," "Facebook ID," and "Twitter," the provided value is between -9223372036854775808 through 9223372036854775807.

3 Response

This section describes the possible response messages sent back from the STANDARD DATA LOAD endpoint.

Success

Upon successful completion of a STANDARD DATA LOAD service request, a "success" message is returned with a response code of '200.'

For example:

```
{
  "success": true
}
```

Errors

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

Response Code	Error message	Description
400	parameter '<field>' : unknown parameter	Provided field name is not defined in the database.
400	parameter '<field>' : ambiguous definition	Provided field name is duplicated within the message.
400	not found unique field(s) {<UniqueID>}	Unique Identifier is not found within the message.
400	parameter '<UniqueID>' : unique field(s) is not set"	Unique Identifier field found for main table, but value is empty.

Response Code	Error message	Description
400	parameter '<UniqueID>' : must not be empty	Unique Identifier field found for joined table, but value is empty.
400	parameter '<field>' : length is limited to 255 Bytes	Provided value for a "String," "Email," "Push Registration ID," or "LINE Contact MID" type field surpasses the 255-byte limit.
400	parameter '<field>' : length is limited to 8000 Bytes	Provided value for a "Long String" type field surpasses the 8000-byte limit.
400	parameter '<field>'='<value>' : value must be between -922337203685477.5808 and 922337203685477.5807	Provided value for a "Money / Decimal" type field is not within the valid range.
400	parameter '<field>'='<value>' : invalid datetime value	Provided value for a "Date / Time" field is not valid.
400	parameter '<field>'='<value>' : value must be between 1/1/1753 12:00:00 AM and 12/31/9999 11:59:59 PM"	Provided value for a "Date / Time" field is not within the valid range of dates.
400	parameter '<field>'='<value>' : wrong email format"	Provided value for an "Email" type field is not the right format.
400	parameter '<field>'='<value>' : wrong phone format"	Provided value for a "Phone" type field is not the right format.
400	parameter '<field>'='<value>' : Value was either too large or too small for an Int64	Provided value for a "Big Integer," "Facebook ID," or "Twitter" type field is not within the valid range.

4 Sample Messages

Request

This sample request message writes three fields to the client's database.

JSON Payload

```
{
  "apiPostId": "2515",
  "enable_validation": "true",
  "validate_on_error": "true",
  "data": [
    {
       "name": "email",
       "value": "mary.smith@cheetahdigital.com"
    },
    {
       "name": "name_first",
       "value": "Mary"
    },
    {
       "name": "name_last",
       "value": "Smith"
    }
  ]
}
```

XML Payload

```
<Recipient>
                                        <apiPostId>198</apiPostId>
                                                    <data>

<ColumnValue>

                                                <name>name_first</name>

                                                        <value></value>

</ColumnValue>

<ColumnValue>

                                    <name>weddingrewards_optin</name>

                                                        <value></value>
```

```
</ColumnValue>
<ColumnValue>
                                                    <name>lame_last</name>
                                                           <value></value>
</ColumnValue>
<ColumnValue>
                                                        <name>email</name>
                                                           <value></value>
</ColumnValue>
                                                                    </data>
</Recipient>
```

5 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.

Form ID

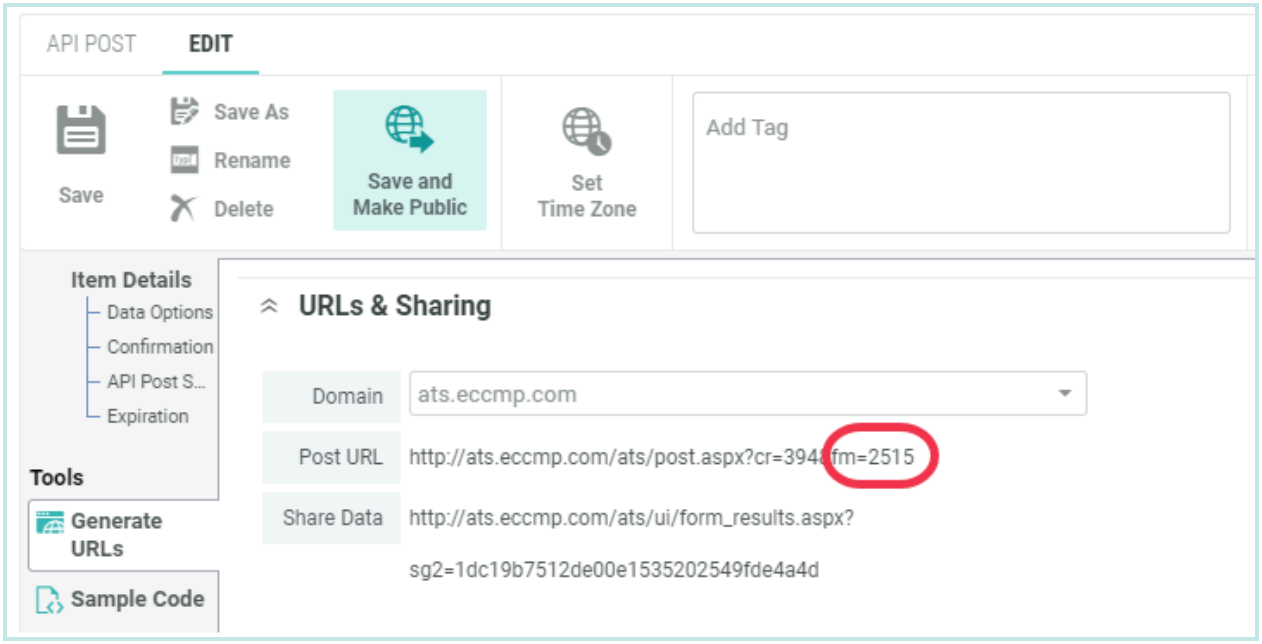
The Form ID is a system-generated identifier for every API Post in your account.

> **Note** API Post, the "Form ID" and the "Object Reference ID" have the same value. In common usage, when referring to the identifier for an API Post, you'll typically hear the term "Form ID" rather than "Object Reference ID."

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

To find the Form ID within the application:

1. From the System Tray, select *Data Integration* > *Processes* > *API Posts*. The system displays a list of all the API Posts in your account.
2. Select the desired API Post. The API Post Details screen is displayed.
3. From the Function Menu, select "Generate URLs." Within the "Post URL" field, the system displays the Form ID as the value of the "fm" parameter.



Optionally, you can use the SEARCH endpoint, and search for the desired API Post:

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 API Post name or its type. To retrieve information about all of your API Posts, use the asset type "ImportFormApi." For example:

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

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

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

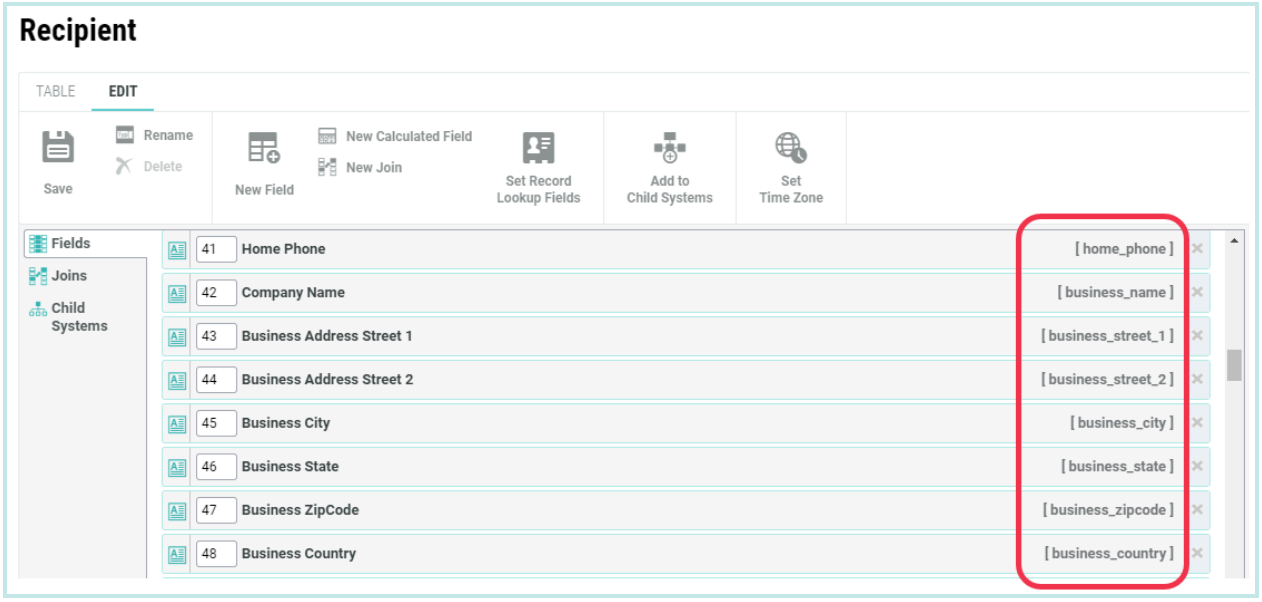
```
{
  "obj_id": 44737,
  "display_name": "Sequential Data Load API Post",
  "type_id": "ImportFormApi",
  "ref_id": 2515,
  "parent_obj_id": 37249,
  "eligibility_status_id": "READY"
}
```

Column Name

The Column Names for fields can be found on the Tables screen within the Messaging application, or by using the TABLE endpoint.

To look up the field name within the Messaging application:

1. From the System Tray, select *Data Management* > *Structures* > *Tables*. The system displays a list of all the tables in your account.
2. Select the desired table. The Table Details screen is displayed.
3. Within the list of fields in this table, the Column Name is displayed on the far-right of the screen.



To retrieve the Column Name for a field using the TABLE endpoint:

1. Submit a GET request to the TABLE API endpoint. The simplest method is to use the version of the TABLE endpoint that allows you to retrieve table information based on the table's name. For example:

```
https://api.eccmp.com/services2/api/Table?tableName=recipient
```

2. Within the API response message, the system lists every field in this table. As part of that field definition, the response includes the Column Name (referred to as the **columnName**).

Sample Response:

```
{
 "viewId": 1002,
 "entityId": 100,
 "displayName": "First Name",
 "propId": 1030,
 "columnName": "name_first"
}
```