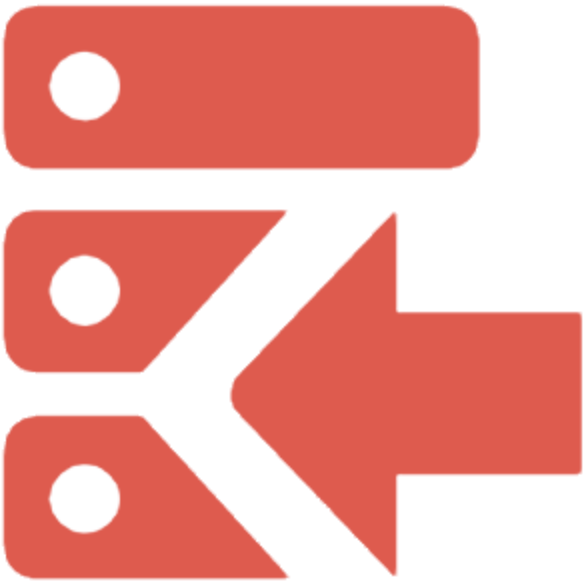
API Technical Guide: Sequential Data Load

Cheetah Messaging

Table of Contents

1 Introduction



Purpose

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

This document also describes using the SEQUENTIAL DATA LOAD endpoint to trigger the deployment of a Campaign. When this endpoint is utilized in this fashion, it's referred to as the SEQUENTIAL EVENT TRIGGER.

Overview

The SEQUENTIAL DATA LOAD endpoint (sometimes referred to as the "XML API" or "XML POST" endpoint) allows you to send a single request message, with data to be loaded into one or more relational tables. For example, you could send a single message that contains one record to be loaded into a ‘Recipient’ table, two related records to be loaded to an ‘Order’ table, and five related records to be loaded to an ‘Order Item’ table. The system will load or update data to these tables, one table at a time, in a user-defined sequence.

Please note that if you're loading data into multiple tables, you can send only ONE record at the parent level (you can optionally write multiple records into the joined tables). Continuing the above example where you're loading data into three tables, each API call can reference only one recipient, but potentially multiple orders, and potentially multiple items per order.

The format of the request message must be executed using the layout described within this document, which is largely driven by your database architecture within the

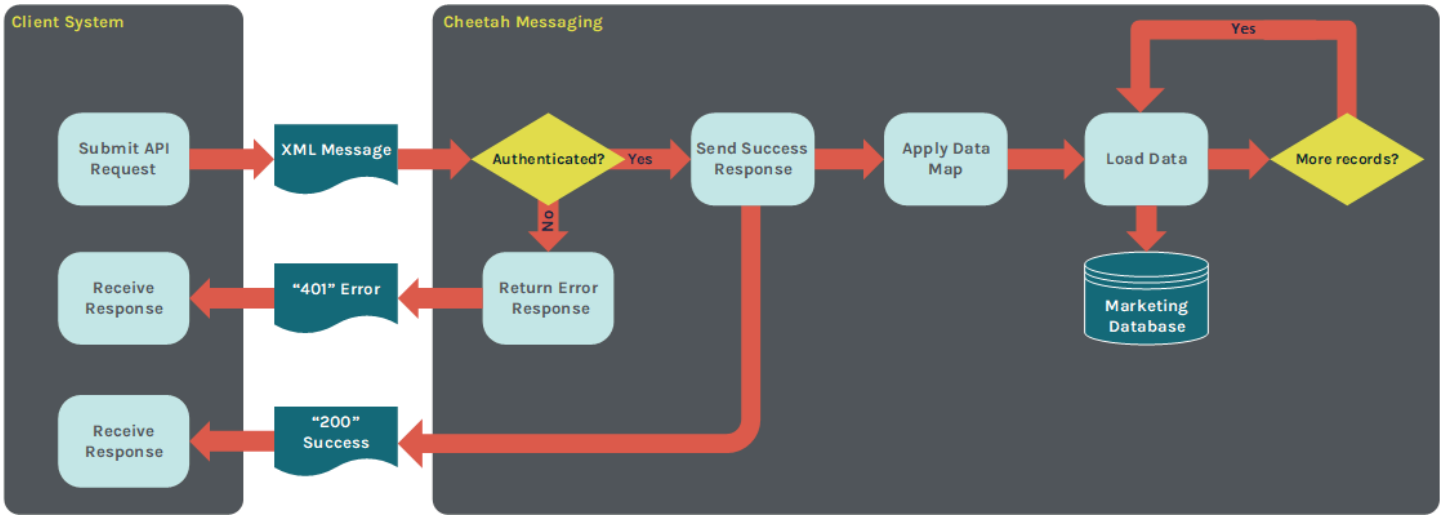
Messaging platform. You must have a strong understand of this architecture, including the tables, joins, and fields in your database, in order to construct this message correctly.

This endpoint requires authentication using OAuth 2.0, and supports only XML messages (not JSON).

The URLs for this endpoint are:

- **North America:** https://ats.eccmp.com/ats/XmlPost/PostSecureAuth2
- **Europe:** https://ats.ccmp.eu/ats/XmlPost/PostSecureAuth2
- **Japan:** https://ats.marketingsuite.jp/ats/XmlPost/PostSecureAuth2

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



Campaign Trigger

The SEQUENTIAL DATA LOAD endpoint can optionally be used as a triggering mechanism for the deployment of an Event-triggered Campaign. The platform will load or update the database prior to deploying the Campaign, which allows you to use data from your database when building your message content.

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

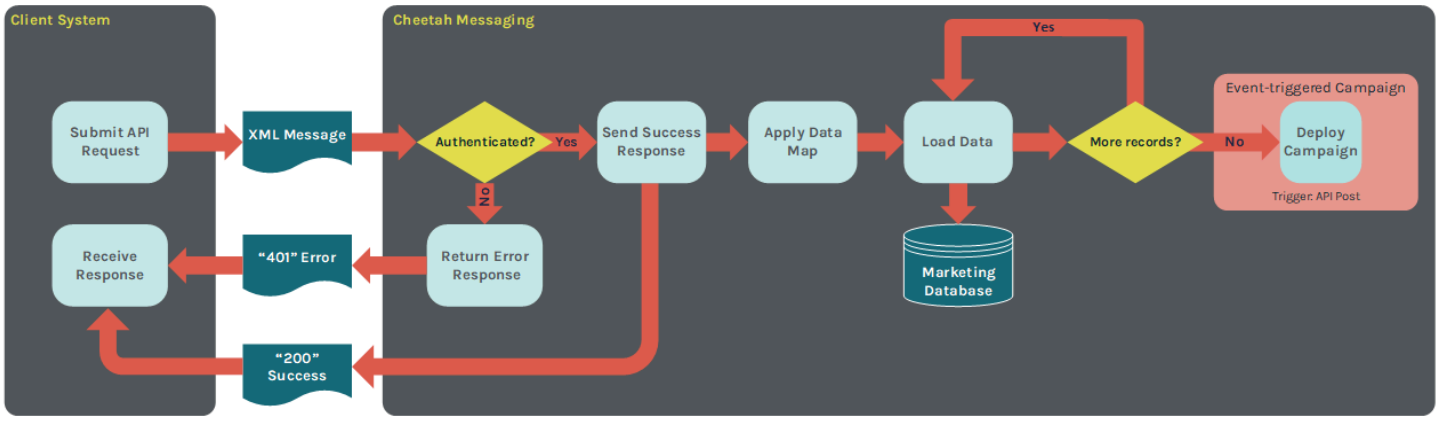
The advantages to using the SEQUENTIAL EVENT TRIGGER are as follows:

- Allows you to load data into multiple joined relational database tables via a single API request message.
- Allows you to populate the email message content with Personalization values pulled from your database.
- Typically, the audience in an Event-triggered Campaign is "all triggered records." The SEQUENTIAL EVENT TRIGGER allows you to optionally assign a Filter to the Event-triggered Campaign. This Filter applies additional restrictions, or exclusions, to the Campaign audience.

The key drawback to the SEQUENTIAL EVENT TRIGGER is speed, because the platform has to load the data from the API message into the database first, before deploying the Campaign. The time it takes to send the email is hard to predict, as it depends on many variables, such as the complexity and volume of the data structure in the API message, and the current load on the database.

If your business requirements demand a faster, more predictable send time, you may want to consider one of the other Messaging trigger APIs, such as ADVANCED EVENT TRIGGER or EMAIL CAMPAIGN TRIGGER.

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



Pre-requisites

The SEQUENTIAL 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, and should use the Data Map described above. Please note that the API Post must have both the "REST API Only" and the "Triggered" options unchecked, and the API Post must be published.

If you're writing data to multiple tables, you must have a Data Map and an API Post defined for EACH table. As an example, let's say you're using the SEQUENTIAL DATA LOAD endpoint to load data into three tables -- 'Recipient,' 'Order,' and 'Order Item.' You would need to define the following assets:

- 'Recipient' table:
- A Data Map built off the 'Recipient' table that defines the fields being loaded / updated in this table.
- An API Post built off the 'Recipient' table that uses the above Data Map.
- 'Order' table:
- A Data Map built off the 'Order' table that defines the fields being loaded / updated in this table. This Data Map must also include a join to the above 'Recipient' table.
- An API Post built off the 'Order' table that uses the above Data Map.
- 'Order Item' table:
- A Data Map built off the 'Order Item' table that defines the fields being loaded / updated in this table. This Data Map must also include a join to the above 'Order' table.
- An API Post built off the 'Order Item' table that uses the above Data Map.

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 an API Post as the event trigger type. If you're writing data to multiple tables, the API request message will define the sequence in which these tables are loaded / updated. Typically, you want to use the API Post for the last, final table in the sequence as the trigger mechanism for the Campaign. In this manner, you can be certain that all of the necessary data contained in the API message has been loaded into the database, prior to building and sending the email message.

Methods

The SEQUENTIAL DATA LOAD endpoint supports the following HTTP method:

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

Authentication

Access to the SEQUENTIAL 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 Request

Overview

This section describes how to write data to your database using a POST request to the SEQUENTIAL 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.

Each request message is represented by an **<ApiSubmission>** tag. Within the **<ApiSubmission>** tag, the message contains one or more **<Records>** tags, each of which has one or more **<Record>** tags, each of which has one or more **<Field>** tags.

The basic structure of the SEQUENTIAL DATA LOAD message is as follows:

```
<ApiSubmission>
  <Records>
    <Record>
      <Field>
      </Field>
    </Record>
  </Records>
</ApiSubmission>
```

If you're writing data to multiple tables, you'll need a separate **<Records>** tag for each table. As an example, let's say you're using the SEQUENTIAL DATA LOAD endpoint to load data into three tables -- 'Recipient,' 'Order,' and 'Order Item.' The basic structure of your API message would be as follows:

```
<ApiSubmission>
                    <!--order item table info-->
  <Records>
    <Record>
      <Field>
```

```
        </Field>
      </Record>
    </Records>
                    <!--order table info-->
    <Records>
      <Record>
        <Field>
        </Field>
      </Record>
    </Records>
                    <!--recipient table info-->
    <Records>
      <Record>
        <Field>
        </Field>
      </Record>
    </Records>

</ApiSubmission>
```

ApiSubmission

The **<ApiSubmission>** tag includes several high-level parameters.

For example:

```
<ApiSubmission CustId="394" SubmissionTrackingCode="Testing"
FormGroupId="1" ApiVersion="1">
```

These parameters are described below in more detail.

CustId

This integer parameter is required.

The **CustId** parameter represents the **Customer ID** of your Messaging account. The Customer ID is a unique, system-generated identifier for every Messaging client account.

Example:

```
CustId="394"
```

SubmissionTrackingCode

This string parameter is optional.

If used, the **SubmissionTrackingCode** parameter allows you to provide a value that Messaging will store as part of every record created in this API message. This feature is conceptually similar to a Globally Unique ID (GUID); it lets you link together records that

all came in together on the same API message. This parameter is primarily intended for auditing and logging purposes.

To use this feature, you must have a field named "SubmissionTrackingCode" defined in every table into which you're loading data. The value you provide in the API message will then be written into this field in every impacted table.

Example:

```
SubmissionTrackingCode="Testing"
```

FormGroupId

This integer parameter is optional.

If used, the **FormGroupId** parameter should be set to "1."

Example:

```
FormGroupId="1"
```

ApiVersion

This integer parameter is optional.

If used, the **ApiVersion** parameter should be set to "1."

Example:

```
ApiVersion="1"
```

Records

The **<Records>** tag defines the table to which you're loading or updating data. You may have one or more **<Records>** tags in your message -- one for each table.

Example:

```
<Records TableName="recipient" FormId="2674" SubmissionSequence="1">
```

These parameters are described below in more detail.

TableName

This string parameter is required.

The **TableName** parameter represents the **Table Name** for the table to which you're writing data.

You can use either the system name of the table, or the user-friendly display name. For example, let's say you have a table with a display name of "Order Item Table." By default, the platform will automatically generate the system name for this table as "order_item_table." When you're submitting the SEQUENTIAL DATA LOAD message, you can use either value.

Example:

```
TableName="recipient"
```

FormId

This integer parameter is required.

The **FormId** parameter represents the **Form ID** of the API Post that you defined for this table.

Example:

```
FormId="2674"
```

SubmissionSequence

This integer parameter is required.

If you're writing data to multiple tables, the SEQUENTIAL DATA LOAD endpoint will write data to one table at a time (thus the name "sequential") rather than simultaneously. The **SubmissionSequence** parameter controls the order in which these tables are loaded or updated.

For example, let's say you're loading data into three tables: 'Recipient,' 'Order,' and 'Order Item.' Each table must have its own **<Records>** tag, each with its own **SubmissionSequence** parameter that defines which table gets loaded first, which gets loaded second, and which gets loaded last.

Example:

```
<Records TableName="order_item" FormId="2677" SubmissionSequence="1">
      <Record>
```

```
                                                                    <Field> ...
</Field>
      </Record>
</Records>

<Records TableName="order" FormId="2676" SubmissionSequence="2">
      <Record>
                                                                    <Field> ...
</Field>
      </Record>
</Records>

<Records TableName="recipient" FormId="2674" SubmissionSequence="3">
      <Record>
                                                                    <Field> ...
</Field>
      </Record>
</Records>
```

In the above example, the platform will write data to the 'Order Item' table first, as it has the lowest **SubmissionSequence** value. Next, the system will write data to the 'Order' table, then lastly to the 'Recipient' table.

If you're using this API message to trigger the deployment of an Event-triggered Campaign, you typically want to use API Post for the final table in the sequence as the trigger mechanism. In the above example, this would be the API Post with the **FormId** value of "2674." In this manner, you can be certain that all the necessary data (Order details, Order Item details) is loaded to the database, before building and sending the email message.

Record

The **<Record>** tag defines a single record, or row. You may have one or more **<Record>** tags for a **<Records>** tag within your message.

Example:

```
<Record>
      <Field FieldName="email">johndoe@cheetahdigital.com</Field>
      <Field FieldName="first_name">John</Field>
      <Field FieldName="last_name">Doe</Field>
</Record>
```

These parameters are described below in more detail.

Field

The **<Field>** tag defines a single field, or column in this table. You may have one or more **<Field>** tags for a **<Record>** tag within your message.

Example:

```
<Field FieldName="email">johndoe@cheetahdigital.com</Field>
```

These parameters are described below in more detail.

FieldName

This string parameter is required.

The **FieldName** parameter represents a single field, or column, along with its corresponding value. For the field names, you must use the field's **Column Name**, and not the viewer-friendly "Display Name."

3 Response

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

Success

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

For example:

```
{
  Data saved and pending processing.
}
```

Errors

If Messaging encounters a problem with a SEQUENTIAL 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
200	FormId is not specified for element with the sequence number <x>	The required **FormId** parameter is missing for one of the tables in your message.
200	Invalid XML	XML code is malformed, such as a missing bracket, for example.
200	Please specify the customer ID with the top level attribute 'CustId'	The required **CustId** parameter is missing.

Response Code	Error message	Description
200	Error saving data. Please check that your Customer ID is correct, and that Form IDs are correct.	The **FormID** and / or **CustId** parameter is invalid.
200	There was an error. Contact administrator.	You may have posted the XML to the wrong page on the correct server of the endpoint URL. Verify that your URL is correct.

4 Sample Messages

This section contains several sample XML payloads to the SEQUENTIAL DATA LOAD endpoint.

Request #1

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

XML Payload

```
<ApiSubmission CustId="394">
  <Records TableName="recipient" FormId="2674" SubmissionSequence="1">
    <Record>
      <Field FieldName="email">johndoe@cheetahdigital.com</Field>
      <Field FieldName="name_first">John</Field>
      <Field FieldName="name_last">Doe</Field>
    </Record>
  </Records>
</ApiSubmission>
```

Request #2

This sample requests writes data to three different tables -- 'Recipient,' 'Order,' and 'Order item.' In this particular order, the customer has purchased two items, a Red Shirt and Blue Socks.

XML Payload

```
<ApiSubmission CustId="394">
  <Records TableName="order_item" FormId="2677" SubmissionSequence="1">
    <Record>
      <Field FieldName="order_id">12347</Field>
      <Field FieldName="order_item_id">123</Field>
      <Field FieldName="prodname">Red Shirt</Field>
    </Record>
    <Record>
      <Field FieldName="order_id">12347</Field>
```

```
      <Field FieldName="order_item_id">456</Field>
      <Field FieldName="prodname">Blue Socks</Field>
    </Record>
  </Records>
  <Records TableName="order" FormId="2676" SubmissionSequence="2">
    <Record>
      <Field FieldName="order_id">12347</Field>
      <Field FieldName="order_date">04/06/18 15:01:00</Field>
      <Field FieldName="email">johndoe@cheetahdigital.com</Field>
    </Record>
  </Records>
  <Records TableName="recipient" FormId="2674" SubmissionSequence="3">
    <Record>
      <Field FieldName="email">johndoe@cheetahdigital.com</Field>
      <Field FieldName="name_first">John</Field>
      <Field FieldName="name_last">Doe</Field>
    </Record>
  </Records>
</ApiSubmission>
```

5 Appendix -- Identifiers

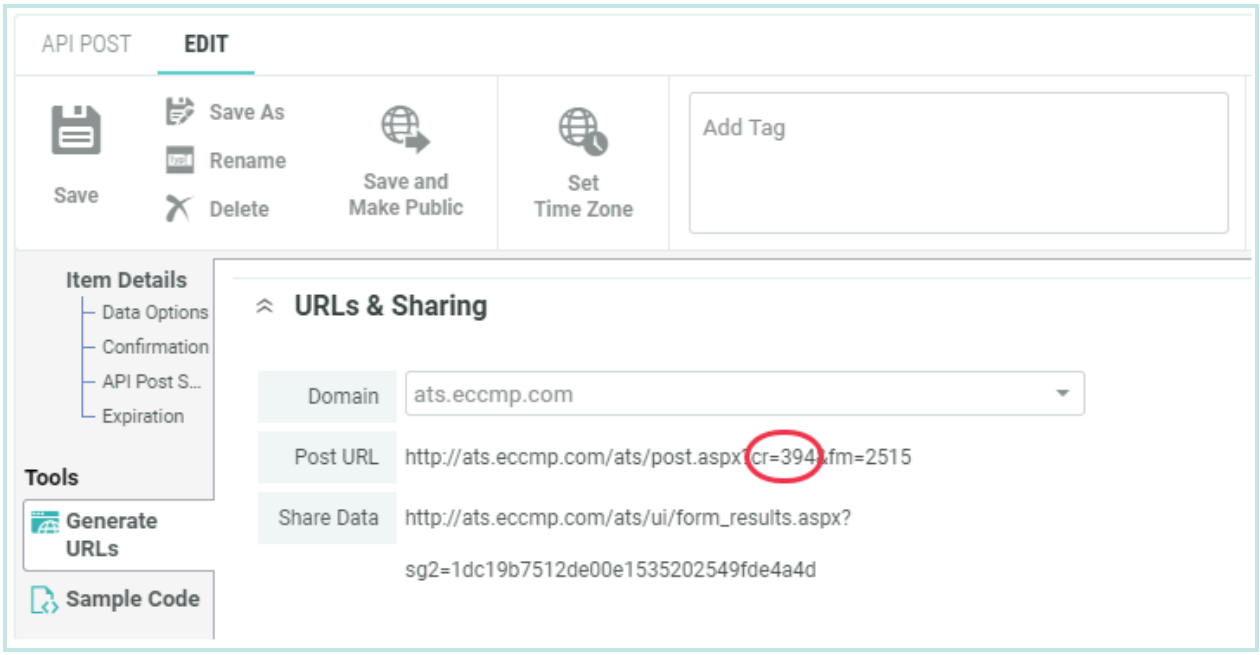
Messaging uses several different types of IDs when referencing assets, such as tables, fields, folders, Filters, and so forth. This appendix describes these different types of IDs, and provides steps on how to look up the value of an ID.

Customer ID

The Customer ID is a system-generated identifier for your client account.

To find your Customer 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 your Customer ID as the value of the "cr" parameter.



Form ID

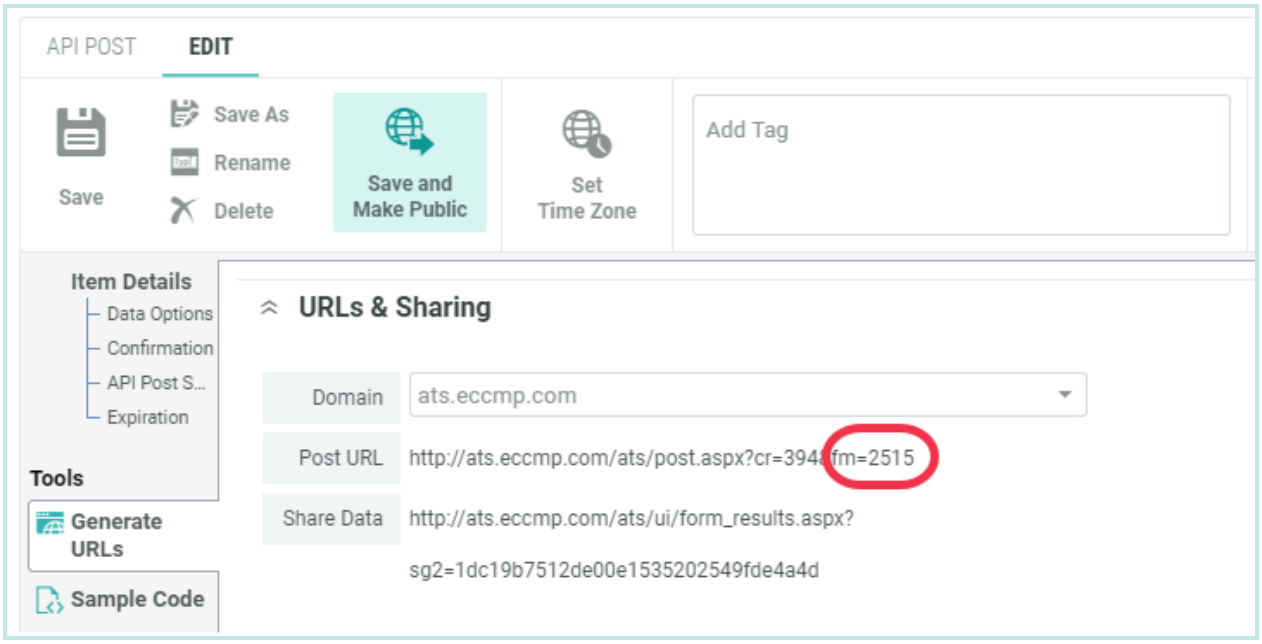
The Form ID is a system-generated identifier for every API Post in your account.

> **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:

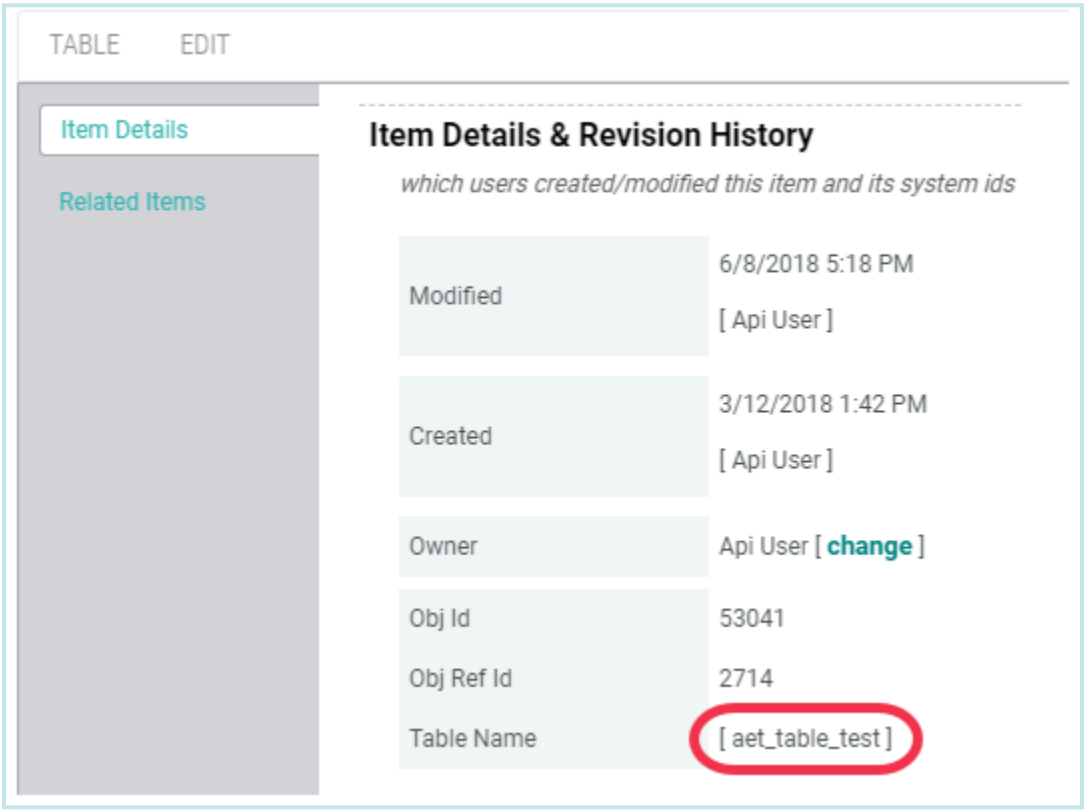
```
{
  "obj_id": 44737,
  "display_name": "Sequential Data Load API Post",
  "type_id": "ImportFormApi",
  "ref_id": 2515,
  "parent_obj_id": 37249,
  "eligibility_status_id": "READY"
}
```

Table Name

Tables in Messaging have a user-friendly display name, and a corresponding system-generated name. For example, let's say you have a table with a display name of "Order Item Table." By default, the platform will automatically generate the system name for this table as "order_item_table." When you're submitting the SEQUENTIAL DATA LOAD message, you can use either name.

You can look up table names within the application:

1. From the System Tray, select *Data Management* > *Structures* > *Tables*. The system provides a list of all the Tables in your account, using the tables' display names.
2. Select the desired Table. The Table Details screen is displayed.
3. In the Tool Ribbon, click the "Table" tab. The "Item Details" screen is displayed. The system name for this table is displayed.



You can also retrieve table names using the TABLE API endpoint.

To retrieve table names:

1. Submit a request to the TABLE API endpoint. The simplest method is to use the version of the TABLE endpoint that allows you to retrieve information for all tables. Please note that depending on the number of tables in your account, you may need to increase the response message size (by default, the system returns the first 20 tables). Within the URL, add the **count** query type parameter, and enter the number of tables you want to return. For example:

```
https://api.eccmp.com/services2/api/Table?count=50
```

2. Within the API response message, the system returns the table details. The table's system name is provided in the **tableName** parameter, and the user-friendly display name is provided in the **viewName** parameter.

Sample Response:

```
{
  "viewId": 2714,
  "viewName": "AET table test",
  "entityId": 816,
  "tableName": "aet_table_test"
}
```

Column Name

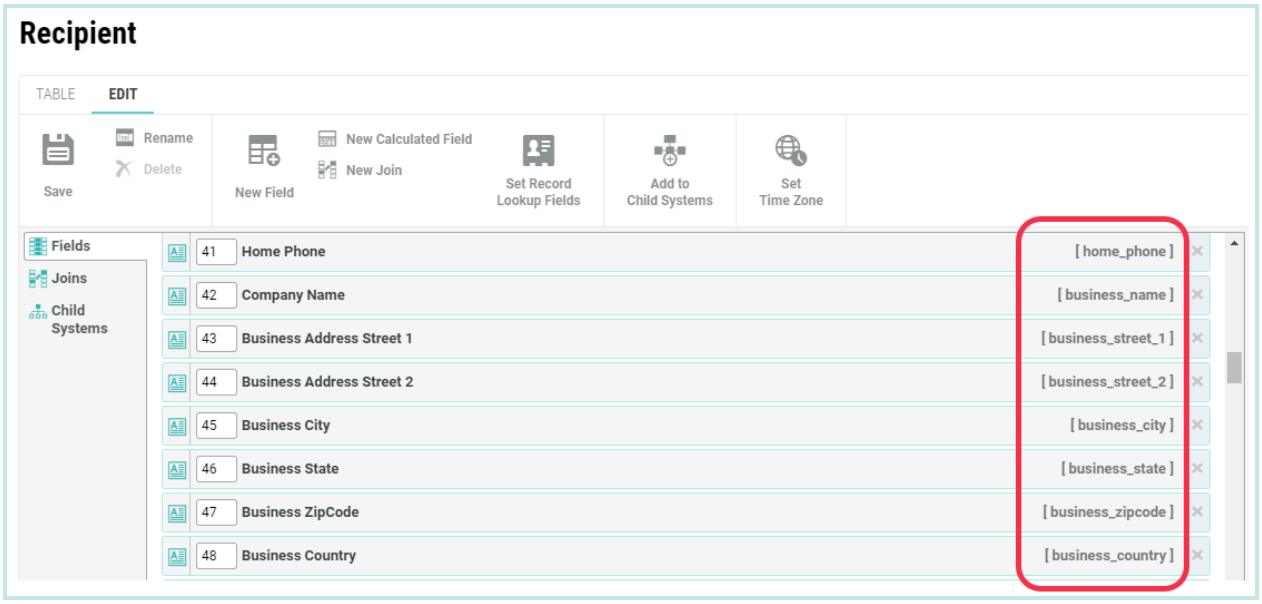
The Column Names for fields can be found on the Tables screen within the Messaging application, or by using the TABLE endpoint.

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"
}
```