API Technical Guide: Table

Cheetah Messaging

Table of Contents

1 Introduction

Purpose

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

Overview

The TABLE endpoint is used to retrieve information about the existing tables in your Messaging database. The endpoint allows you to submit a variety of different search criteria in order to identify the desired table; the response message will then contain detailed information about every field in the specified table. You can also request a list of all the tables in your database.

The TABLE endpoint is typically used to look up various system-generated identifiers. For example, you can use the TABLE endpoint to look up the Entity ID for a table, as well as the Field ID for every field within that table. These IDs are then used in other endpoints, such as the DATA MAP endpoint, for example.

Please note that you can't create a new table, or modify an existing table, through the TABLE endpoint. To create or modify a table, you'll need to use the Tables screen within the Messaging application.

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/Table

- **Europe**: https://api.ccmp.eu/services2/api/Table
- **Japan**: https://api.marketingsuite.jp/services2/api/Table

Methods

The TABLE endpoint supports the following HTTP methods:

- **GET**: Retrieve information about a specified Table, by providing either of the following:

- The table's Object Reference ID
- The table's name

- **GET**: Retrieve a list of all tables in the database.

Authentication

Access to the TABLE 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 Retrieve Table Details

Overview

This section describes how to retrieve information about the tables in your database using the TABLE endpoint.

Retrieve via Object Reference ID

Using a GET method, you can retrieve all of the information about a single table by specifying the table's **Object Reference ID**.

When submitting a GET request to the TABLE endpoint, the request message must include the table's Object Reference ID as a query type parameter within the URL.

For example:

```
https://api.eccmp.com/services2/api/Table?viewId=3456
```

count / page

These two integer parameters are optional, and should be provided as query type parameters in the URL. For example:

```
https://api.eccmp.com/services2/api/Table?viewId=3456&page=2&count=2
```

If you don't provide a value for these parameters, the system will default them to blank.

These parameters are used to control how many fields, and which fields, should be included within the response message. By default, the system will return all fields within the specified table, sorted by **displaySequence** in ascending order. This sort order mimics the way the fields are displayed on the Tables screen within the application.

However, you can use the **count** and **page** parameters to limit the response message to only a certain quantity and selection of fields.

The **count** parameter will split the response message up into "pages" of the designated size. The **page** parameter then tells the system which page you want to see in the response message.

For example, let's say your table has twenty fields in it, and you want to see only the last ten fields in the response message. You could set **count** to "10" so that the system splits the response into two pages (page 1 with fields 1 through 10, and page 2 with fields 11 through 20). You would also set **page** to "2," so that you receive only the second page of fields in the response.

Table Name

Using a GET method, you can retrieve all of the information about a single table by providing that table's full, exact name.

When submitting a GET request to the TABLE endpoint, the request message must include the table's Display Name as a query type parameter within the URL.

For example:

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

The **tableName** parameter represents the exact name of the desired table. The platform doesn't perform any fuzzy matching, or "contains this text" type of search logic. The value provided in the request message must match the full, exact name of the desired table (the endpoint is not case-sensitive).

You must use the reader-friendly name of the table, as its displayed within the user interface, and not the system name for the table. 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 a request message using this GET method, you must use the display name: "Order Item Table."

The table name can be found within the Messaging application, or by using the **All Tables** method described below.

count / page

These two integer parameters are optional, and should be provided as query type parameters within the URL. For example:

```
https://api.eccmp.com/services2/api/Table?viewId=3456&page=2&count=2
```

If you don't provide a value for these parameters, the system will default them to blank.

These parameters are used to control how many fields, and which fields, should be included within the response message. By default, the system will return all fields within the specified table, sorted by **displaySequence** in ascending order. This sort order mimics the way the fields are displayed on the Tables screen within the application.

However, you can use the **count** and **page** parameters to limit the response message to only a certain quantity and selection of fields.

The **count** parameter will split the response message up into "pages" of the designated size. The **page** parameter then tells the system which page you want to see in the response message.

For example, let's say your table has twenty fields in it, and you want to see only the last ten fields in the response message. You could set **count** to "10" so that the system splits the response into two pages (page 1 with fields 1 through 10, and page 2 with fields 11 through 20). You would also set **page** to "2," so that you receive only the second page of fields in the response.

All Tables

Instead of retrieving information about the fields in a single table, you can use a GET method to retrieve a list of all the tables in your database.

The parameters for this GET method are listed below.

count / index

These two integer parameters are optional, and should be provided as part of the URL, not in the body. For example:

```
https://api.eccmp.com/services2/api/Table?index=2&count=2
```

If you don't provide a value for these parameters, the system will default **index** to "0" and **count** to "20."

These parameters are used to control how many tables, and which tables, should be included within the response message. By default, the system will return the first twenty tables in your database, sorted alphabetically by **tableName**. This sort order mimics the way the tables are displayed on the Tables screen within the application.

However, you can use the **count** and **index** parameters to specify a certain quantity and selection of tables.

The **count** parameter will split the response message up into "pages" of the designated size. The **index** parameter then tells the system which page you want to see in the response message.

For example, let's say your database has twenty tables in it, and you want to see only the last ten tables in the response message. You could set **count** to "10" so that the system splits the response into two pages (page 1 with tables 1 through 10, and page 2 with tables 11 through 20). You would also set **index** to "2," so that you receive only the second page of tables in the response.

3 Response

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

Success

A successful response to the **Object Reference ID** or **Table Name** GET methods will generate a response code of "200," followed by information about the fields in the table, contained within the body of the response message.

The response message will contain both active and inactive fields, as well as any joins defined for the specified table.

Inactive fields are listed first in the response message, followed by active fields, and then by joins. The easiest way to differentiate between inactive and active fields in the response is that inactive fields don't include the **displaySequence** parameter.

A successful response to the **All Tables** GET method will generate a response code of "200," followed by information about your tables, contained within the body of the response message.

Errors

If Messaging encounters a problem with a TABLE 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	Table not found, tableName or viewId is wrong.	Unknown Object Reference ID or table name. Please note that if using the Table Name GET method, the tableName value must exactly match the full name of the table.

4 Sample Messages

Response Message #1

This sample message shows a response to either the **Object Reference ID** or **Table Name** options. The response lists all of the inactive fields, followed by the active fields, followed by the joins, in the specified table.

JSON Payload


```
[
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "Prefix",
    "propId": 17438,
    "columnName": "prefix"
  },
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "Ship Date",
    "propId": 17442,
    "columnName": "ship_date"
  },
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "AccNo",
    "displaySequence": 1,
    "propId": 12467,
    "columnName": "ac"
  },
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "Email",
    "displaySequence": 2,
    "propId": 14875,
    "columnName": "email"
  },
  {
    "viewId": 2196,
    "entityId": 305,
```

```
    "displayName": "ak_recipient",
    "displaySequence": 3,
    "propId": 12465,
    "columnName": "ak_recipient"
  },
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "pk_recipient_id",
    "displaySequence": 4,
    "propId": 12464,
    "columnName": "pk_recipient_id"
  },
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "Phone 64207 - CNV Shared Short Code (29) Status ID",
    "displaySequence": 5,
    "propId": 12470,
    "columnName": "phone_sp29_status_id"
  },
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "Order Date",
    "displaySequence": 6,
    "propId": 17443,
    "columnName": "order_date"
  },
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "Date of Birth",
    "displaySequence": 7,
    "propId": 17441,
    "columnName": "date_of_birth"
  },
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "Score",
    "displaySequence": 8,
    "propId": 17439,
    "columnName": "score"
  },
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "Phone",
    "displaySequence": 9,
    "propId": 17440,
    "columnName": "phone"
  },
  {
```

```
    "viewId": 2196,
    "entityId": 305,
    "displayName": "Gender",
    "displaySequence": 10,
    "propId": 17437,
    "columnName": "gender"
  },
  {
    "viewId": 2196,
    "entityId": 305,
    "displayName": "Order Join",
    "displaySequence": 11,
    "propId": 18473,
    "columnName": "order_join"
  }
]
```


Response Message #2

This sample message shows a response to the **All Tables** option. The response lists all of the tables in the database.

JSON Payload


```
[
  {
    "viewId": 2444,
    "viewName": "Order",
    "entityId": 551,
    "tableName": "order"
  },
  {
    "viewId": 2445,
    "viewName": "Order Item",
    "entityId": 552,
    "tableName": "order_item"
  },
  {
    "viewId": 2380,
    "viewName": "Recipient",
    "entityId": 489,
    "tableName": "recipient"
  }
]
```


The **viewName** and **tableName** parameters in the response message can be confusing. In most cases, the system-generated **tableName** replaces all upper-case letters with lower-case letters, and replaces spaces with underscores. It's important to note that if

you're looking up the name of the table in order to use the **Table Name** GET method, you actually want to use the value in the **viewName** parameter, and not the value in the **tableName** parameter.

5 Appendix A -- Identifiers

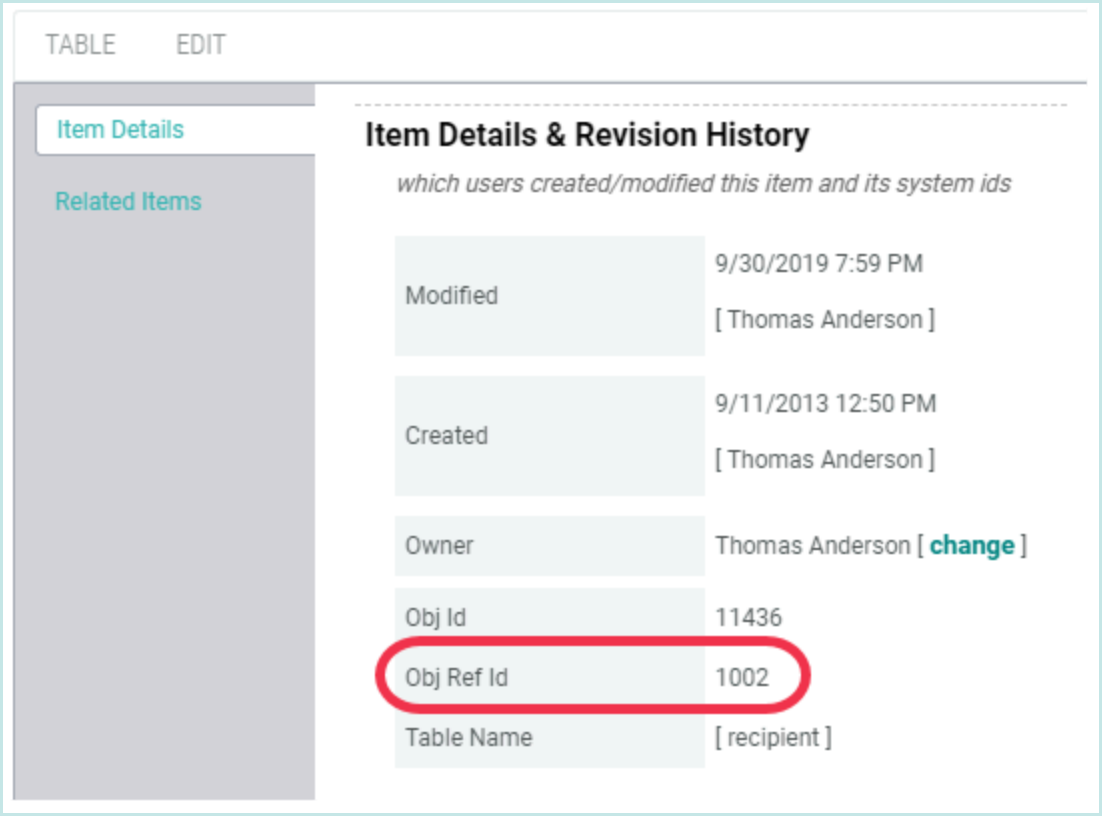
Messaging uses several different types of IDs when referencing assets, such as tables, fields, folders, Filters, and so forth. This appendix describes these different types of IDs, and provides steps on how to look up the value of an ID.

Object Reference ID

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

For Tables, the value for this identifier can be found within the Messaging application:

1. From the System Tray, navigate to *Data Management* > *Structures* > *Tables*.
2. In the Tool Ribbon, click the Table tab.
3. The "Item Details" screen is displayed. The Object Reference ID is listed on this screen.



Optionally, you can use the **Retrieve All Tables** method of the TABLE endpoint. In the response message, the Object Reference ID is contained within the **viewId** parameter. For example:

```
{
  "viewId": 1002,
  "viewName": "Recipient",
  "entityId": 100,
  "tableName": "recipient"
}
```