MARIGOLD™

API Technical Guide: Advanced Event Trigger

Cheetah Messaging

Table of Contents

1 Introduction

Purpose

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

Overview

Advanced Event Trigger (AET) allows clients to "trigger" the deployment of an existing Event-based Campaign in Messaging through the use of an API request message. With AET, you can trigger Campaigns in the Email, Push Notification, and SMS Text channels.

As a common example, the Advanced Event Trigger could be used to deploy an order confirmation email. In this scenario, the consumer makes a purchase from your website. Your system collects all of the necessary purchase information (items, cost, consumer name, email address, etc.) and sends it to Messaging in an API request message. This data is used to render, personalize, and send a message synchronously with your system's request. In parallel, the data you sent will be loaded into one or more tables in your Messaging database.

The following table describes the key advantages of the Advanced Event Trigger.

Key Features	Description
Relational Data Inserts	You can use the AET API to: • Specify which Event-based Campaign to trigger. • Within the payload of the API call, provide all the necessary information to populate the Campaign message. • Update multiple data tables, including relational data.

Key Features	Description
Sophisticated Messaging Features	The AET API supports the following Campaign-related features: • Campaigns in the Email, Push Notification, and SMS Text channels • "From address" mapping • "From name" personalization • "To name" personalization • Subject line personalization • Field value personalization in the mailing content • Default personalization values • Personalization formatting • Content Blocks • Dynamic Blocks (with Filters based on field values) • Looping Blocks (that don't use Filters) • "Record" metadata values • Link tracking • Checking Global and Custom Email Ban Lists to validate a recipient's eligibility to be contacted • Checking the Sender Profile Status ID field to validate a recipient's eligibility to be contacted • Attachments (Email channel only) • Blind Carbon Copy (BCC) email addresses
Rapid Response Messaging	By separating the response process from the database updates, AET is able to perform these functions independently of each other, providing faster API response messaging.

The ADVANCED EVENT TRIGGER endpoint doesn't support the following functionality:

- Dynamic Blocks with Filters based on Activity data
- Looping Blocks with any type of Filter
- Channels other than Email, Push Notification, or SMS Text
- Campaigns with Split Cells
- Some social media features, including Share-to-Social Blocks and Facebook Like buttons
- Checking Preference flags to validate a recipient's eligibility to be contacted

- "View as Web Page" links in the email message content

Use Cases

The ADVANCED EVENT TRIGGER API supports the following use cases:

- Order confirmation and order status (shipping) notifications
- Website registration welcome messages
- Mailing list opt-ins
- Point-of-sale e-receipts
- Cart abandonment
- Event status updates, such as flight changes, or gate changes

Endpoints

The ADVANCED EVENT TRIGGER API utilizes the following endpoints. The platform supports two variants of the AET endpoint -- one that contains the Campaign identifier within the URL, and one that contains the Campaign identifier within the message payload. The payload variant is intended for use by clients who need the URL to remain stable, and not change for each AET Campaign.

- **North America**:

- https://aet.eccmp.com/services2/api/Campaign/{ID}/Trigger
- https://aet.eccmp.com/services2/api/Campaign/Trigger

- **Europe**:

- https://api.ccmp.eu/services2/api/Campaign/{ID}/Trigger
- https://api.ccmp.eu/services2/api/Campaign/Trigger

- **Japan**:

- https://aet.marketingsuite.jp/services2/api/Campaign/{ID}/Trigger
- https://aet.marketingsuite.jp/services2/api/Campaign/Trigger

Optionally, you can use the relational data insert functionality of AET without triggering the deployment of a Campaign. This "load only" endpoint is called the ADVANCED DATA LOAD endpoint. For more details, please see the *Cheetah Messaging -- Advanced Data Load Technical Guide*.

Methods

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

The ADVANCED EVENT TRIGGER endpoint supports the following HTTP method:

- **POST**: Trigger the deployment of an Event-triggered Campaign.

Authentication

Access to the ADVANCED EVENT TRIGGER 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.

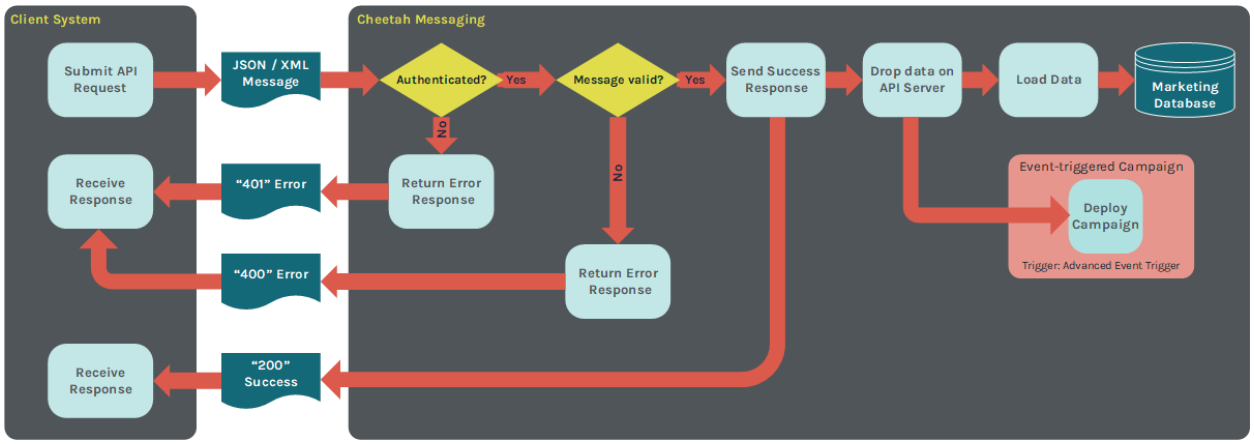
Next, you'll use your Consumer Key and Consumer Secret to request a "token." A token is a text string that, when provided in a request message, will allow the user access to the requested service. Tokens are valid only for a certain period of time.

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

Process Flow

Other campaign-trigger APIs in Messaging use a "sequential" process, meaning that the platform first updates your database, then creates and deploys the message to the intended recipient. This architecture results in slower message deployment times, because of the time required to first write to the database.

The ADVANCED EVENT TRIGGER API instead separates the database update from the message deployment, thereby allowing these processes to happen simultaneously, rather than sequentially. This architecture allows the message to be deployed more quickly.



Because the message creation is separated from the database load, the payload of the API request must include all of the necessary information needed to populate and send the message (such as Personalization values, email address, Push Registration ID, etc.). The endpoint is capable of receiving relational data, stored across multiple tables, and consisting of multiple records (such as purchased items, for example), without having to use a pre-configured Data Map, and without making multiple inserts to the parent table(s).

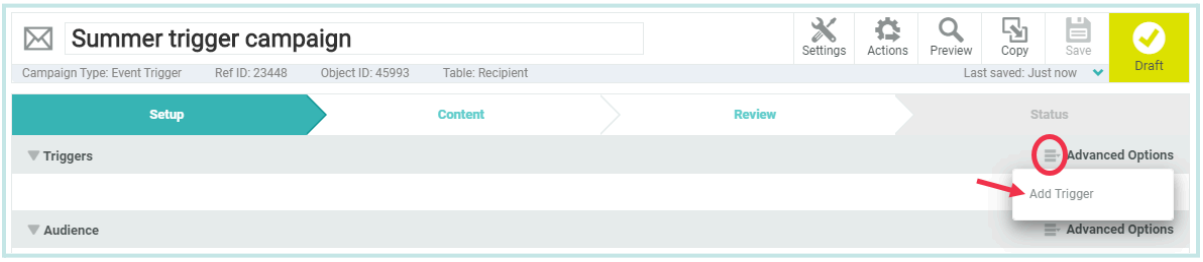
In addition, the endpoint supports sophisticated Campaign content creation options, including Looping Blocks, Dynamic Blocks, and Personalization. To be clear, any Filters included in the content definition are limited to the fields that can be included in the API payload. An example of data that can't be queried by AET content Filters would be any activity data, like clicks, opens, or bounces.

The ADVANCED EVENT TRIGGER API provides more accurate response messaging by performing the same validations that will be later be performed when the data is actually loaded to the database. In this manner, the system will return a "Success" message with a high degree of confidence that the data insert will ultimately succeed.

2 Event-Triggered Campaigns

The ADVANCED EVENT TRIGGER API is designed to be used as a Campaign deployment trigger. Messaging supports a variety of these trigger mechanisms – such as a file import, a web form submission, or an email click, for example -- which can be used to initiate the process of sending a message to a recipient.

When using AET as the Campaign trigger, you must first set up an Event-triggered Campaign. Select the **Setup** chevron, and from the Triggers section, select **Add Trigger**.



From the Add Trigger pop-up window, select **Advanced Event Trigger** as the trigger type.

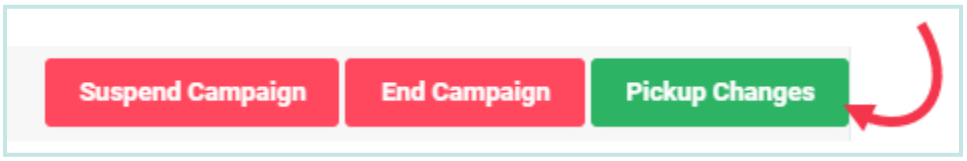
> **Note**
>
> You can also use the EMAIL CAMPAIGN or PUSH NOTIFICATION CAMPAIGN or SMS TEXT CAMPAIGN endpoints to create the AET Event-triggered Campaign.

The Event-triggered Campaign must be launched, which tells the platform to begin "listening" for the triggering event – in this case, an AET request message. Upon receipt of an AET request message that specifies this Campaign, Messaging will build the content for the message, and deploy the message to its intended recipient.

For general information on how to configure Campaigns, please see the *Cheetah Messaging Online Help* system. The following sections describe Campaign set-up features unique to an AET Event-triggered Campaign.

Pick Up Changes

If you need to make changes to the Event-triggered Campaign after you launch it, you must execute the platform's Pick Up Changes process. However, please note that when you run Pick Up Changes, the system has to refresh the Campaign definition on the AET server, which can take up to ten minutes. Any AET messages sent during that interval, prior to the refresh taking place, will receive the old version of the Campaign. Once the Campaign definition has been refreshed on the AET server, all subsequent messages will trigger the new version of the Campaign.



Stop or Suspend a Campaign

If you Suspend, Cancel, or Stop the AET Event-triggered Campaign, you may see messages continue to get deployed for a few minutes. The system has to refresh the Campaign definition on the AET server, which can take up to ten minutes. Once the Campaign definition has been refreshed on the AET server, all message deployment will be halted.

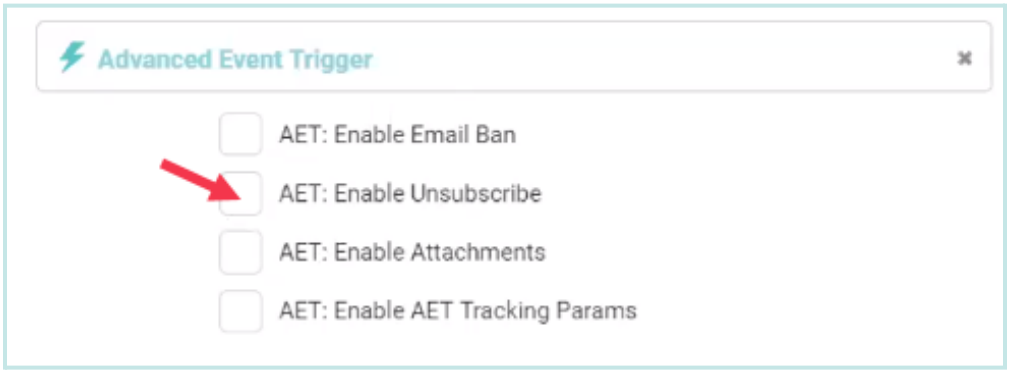
Check Unsubscribe Status

Within Messaging, the Sender Profile serves as the broadest level for maintaining the contact permission status of your recipients. Every Sender Profile has a corresponding "Status ID" field in the Campaign source table. The Status ID field uses various codes to indicate the eligibility of this recipient to be included in a Campaign. For example, a value of "1500" in the Status ID field indicates that the recipient opted-out by means of a web form.

When using Advanced Event Trigger to deploy Campaigns, you have the option of validating the addresses in the API payload against the Sender Profile Status ID field. If you enable this validation check, the system will suppress any recipients that have an

"unsubscribed" Status ID value. If you disable this validation check, the system will automatically deploy messages to every recipient included in the API payload.

The Check Unsubscribe option is displayed on the Campaign details screen for an Event-triggered Campaign that has "Advanced Event Trigger" selected as the trigger mechanism. The option is displayed within the "Triggers" section in the Setup chevron. By default, this validation check is disabled.



Please note that if you check the "Enable AET Tracking Params" option, you can't check "Enable Unsubscribe." See below for more details on the "Enable AET Tracking Parms" option.

Email Channel

This section describes the Advanced Event Trigger features and configurations that are unique to the Email channel. Please see Section 4 for sample payloads used to trigger an Email Campaign:

- JSON Message -- Single data table
- JSON Message -- Joins to multiple tables
- XML Message -- Joins to multiple tables

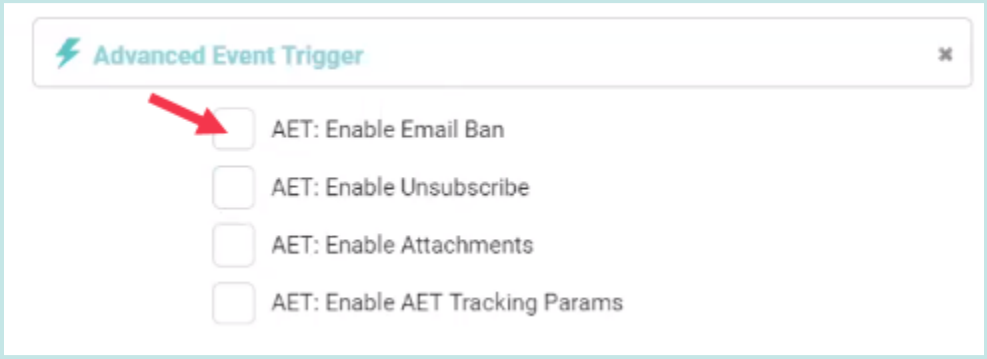
Check Email Ban List

Messaging maintains a global, integrated Banned Email list that's utilized by every client in the platform. This global Banned Email list contains, for example: domains listed by the Federal Communications Commission that can't be mailed to; inactive or retired

domains; and email addresses with prefixes commonly used by spammers. In addition to this global Banned Email list, the platform allows you to define your own custom Banned Email list that contains specific email addresses, user names, or domains that you want to suppress from your email Campaigns.

When using Advanced Event Trigger to deploy email Campaigns, you have the option of validating the email addresses in the API payload against the global and custom Banned Email lists. If you enable this validation check, the system will suppress any email addresses that are found on either the global or custom lists. If you disable this validation check, the system will automatically deploy email messages to every email address included in the API payload.

The Email Ban List option is displayed on the Campaign details screen for an Event-triggered email Campaign that has "Advanced Event Trigger" selected as the trigger mechanism. The option is displayed within the "Triggers" section in the Setup chevron. By default, this validation check is disabled.

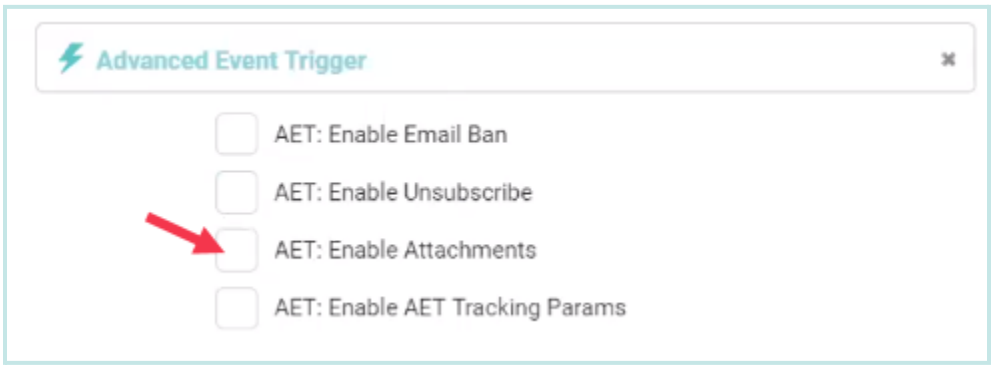


Attachments

The ADVANCED EVENT TRIGGER endpoint allows you to optionally send attachment information along with the API request message. The attachments will then be included within the email message that gets sent to the Campaign's recipients. The use of AET attachments must be enabled within your client account, and also within the Campaign itself. Please speak with your Client Services Representative to enable AET attachments at the client account level.

At a Campaign level, the AET Attachments option is displayed on the Campaign details screen for an Event-triggered email Campaign that has "Advanced Event Trigger" selected

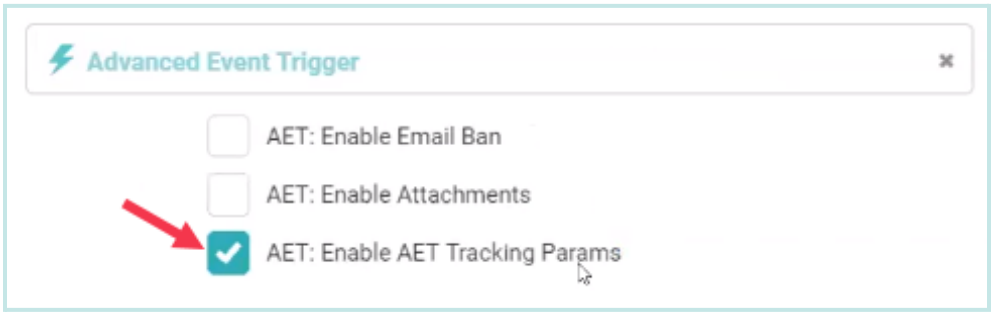
as the trigger mechanism. The option is displayed within the "Triggers" section in the Setup chevron. By default, this flag is disabled.



For details on how to configure Attachments within the AET payload, please see __Attachments.

AET Tracking Parameters

The AET Tracking Parameters feature is designed to provide higher throughput and more consistent response times for AET messages. The AET Tracking Parameters option is displayed on the Campaign details screen for an Event-triggered email Campaign that has "Advanced Event Trigger" selected as the trigger mechanism. The option is displayed within the "Triggers" section in the Setup chevron. By default, this flag is enabled.



Please note that if you check the "Enable AET Tracking Params" option, the "Enable Unsubscribe" flag is removed from the screen. The platform doesn't support the use of the "Enable Unsubscribe" feature if you're using AET Tracking Parameters.

If you need to suppress unsubscribed recipients from the Campaign, then you must first uncheck "Enable AET Tracking Params," then check "Enable Unsubscribe."

Push Notification Channel

This section describes the Advanced Event Trigger features and configurations that are unique to the Push Notification channel. Please see **Section 4** for sample payloads used to trigger a Push Notification Campaign:

- **JSON Message** -- Single data table

Push Registration ID

To send Campaigns in the Push Notification channel, you must have the recipient's Push Registration ID (PRID). The PRID allows Messaging to target a device for Push Notifications on its respective Push Notification Service (PNS).

The PRID is typically the Unique Identifier (or "Alternate Key") in the custom table used when importing devices. Using the PRID, Messaging can identify the device token and the device type and send Push Notifications to an app on that device. The system uses a one-to-one relationship between PRID and device token.

Messaging provides several different methods of looking up the PRID for a recipient. These methods are described below in more detail.

Search Records API

Using the SEARCH RECORDS API, you can submit a query and return all of the records in a specified table that match the query logic.

For example, let's say you want to retrieve records from your Push table where the value in the "push_detail" field is "iOS." You could send a request message like this:

```
https://api.eccmp.com/services2/api/SearchRecords?viewName=push_table&
prop=push_registration_id,push_detail,platform_name&columnName=push_de
tail&operation==&param=ios
```

Sample response:

```
{
  [
        {
            "Id": 1,
            "Properties": [
                {
                    "PropName": "push_registration_id",
                    "Value": "E429575C-E943-49B3-B8CA-32F69AF1CAA4"
```

```
                    },
                    {
                        "PropName": "push_detail",
                        "Value": "iOS"
                    },
                    {
                        "PropName": "platform_name",
                        "Value": "My Iphone "
                    }
                ]
            }
        ]
    }
```

For more details, please see the *Search Records API Technical Guide* or the *Cheetah Messaging Online Help*.

Record Lookup

You can look up the PRID for a recipient using the Record Lookup screen within the application:

1. From the System Tray, select *Data Management* > *Management* > *Record Lookup*.
2. From the "Table" drop-down menu, select the database table that you want to search.
3. From the "Find" menu, select the field that you want to search.
4. Select a mathematical operator.
5. Enter the text string for which you're searching.
6. Click **Search**. The system displays a list of all the records that meet your search criteria. Within the search results, click "Edit" next to the desired record to see the full record.

For more details on how to use the Record Lookup screen, please see the *Cheetah Messaging Online Help*.

Export

Messaging allows you to extract data from your database in order to use that data outside of the platform for further analysis, or as input into some other process or system. These extracts from the database are called "Exports," and the platform provides a wide range of options for how to define these Exports.

For example, you could define an Export that runs every night, and contains the Push Registration ID for all records on your Push table.

An Export is defined through the use of a repeatable Template. An Export Template controls the content and layout of the extract, the schedule of when and how often it runs, and the destination of the Export file (such as an FTP server, for example).

For more details on how to define an Export, please see the *Cheetah Messaging Online Help*.

Data View API

The DATA VIEW endpoint is used to query a table by providing a single value in a single field. The response message will return up to fifty records that match the query conditions. The response will include every active field in the database for each matching record.

For example, let's say you want to retrieve records from your Push table where the value in the "push_detail" field is "iOS." You could send a request message like this:

```
{
  "objId": 19019,
  "keyProp": "push_detail",
  "keyValue": "ios"
}
```

Sample response:

```
{
 "push_registration_id": "E429575C-E943-49B3-B8CA-32F69AF1CAA4",
 "push_detail": "iOS",
 "platform_name": "My Iphone"
}
```

For more details, please see the *Data View API Technical Guide* or the *Cheetah Messaging Online Help*.

Note

The DATA VIEW endpoint must be enabled within your client account. Please speak with your Client Services Representative to enable this feature.

Mobile Application

Your mobile application should know the PRID, so the application can send the PRID to your Customer Relationship Management system. If you integrate the EMSMobileSDK into

the app, when the recipient opens the device and enables push notifications, the SDK internally calls the XTS endpoint and registers the device_token. When the device_token is registered, the response contains the Push Registration ID.

Sample Request:

```
{
"DeviceToken" : "91af15a0eb4393699b923541fed4f6fc5ee4d363090ef10001d1"
}
```

Sample Response:

```
{
"Push_Registration_Id": "2c89aebb-2462-4b14-9876-ee00ac50cbec",
"Cust_Id": 100,
"Application_Id": "b8ff768e-0f78-4213-9139-bcec7a87d840",
"Device_Token":
"91af15a0eb4393699b923541fed4f6fc5ee4d363090ef10001d1",
"Time_Stamp": "2018-09-28T23:48:35.5797817Z"
}
```

SMS Text Channel

This section describes the Advanced Event Trigger features and configurations that are unique to the SMS Text channel. Please see **Section 4** for sample payloads used to trigger an SMS Text Campaign:

- **JSON Message** -- Single data table

Mobile Phone Numbers

To send Campaigns in the SMS Text channel, you must have the recipient's mobile phone number. The basic rules for the phone number value are as follows:

- The field must not be empty.
- The value should not be less than 5 characters.
- The value should not be greater than 20 characters in length.
- The value should not contain any non-numeric characters, like a plus sign. For example, the phone number "+447700900765" should be sent as "447700900765"

- The recipient's phone number must have a country code, and must not include any additional symbols or letters. For example, "447700900765" (a UK number) or "12515550145" (a US number).

3 Trigger a Campaign

Overview

This section describes how to trigger the deployment of an Event-triggered Campaign via a POST request to the ADVANCED EVENT TRIGGER endpoint.

This section assume that you've already created the Event-triggered Campaign with all the necessary assets and message content, configured the Campaign with a trigger type of "Advanced Event Triggered," and launched the Campaign.

> **Note**
>
> For general information on how to configure Campaigns, please see the *Cheetah Messaging Online Help* system.

Payload

The Advanced Event Trigger request payload is composed of the following main objects:

- **_data**: This required object defines the table relationships for the payload, and contains the data for the relational insert and the Campaign message.
- **_campaignMetadata**: If used, this optional object contains Campaign data that is never inserted into a table, but is available in the Campaign for Personalization within the message content and within Content Blocks (the data isn't available for use in Looping Blocks).
- **_importOptions**: If used, this optional object defines the import rules for tables (and fields) present in the _data section. If this object is not defined for a given table, the data will be imported according to the default behavior.

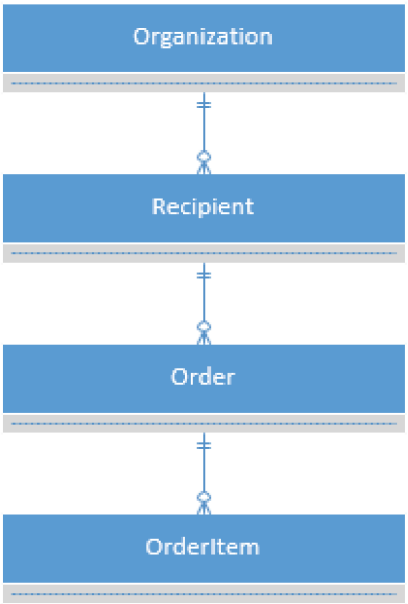
- **_attachments**: If used, this optional object defines the file, or files, that are to be included as attachments with the email message sent to the recipient.

These objects are described below in more detail.

_data

The **_data** object contains the data that you're using to build and populate the email message or Push Notificiation, and to load to your database. To illustrate the proper construction of this object, we will use the schema defined in the diagram below.

This account has the following tables: Organization, Recipient, Order, and OrderItem. Recipient has a join up to Organization (multiple Recipient records can belong to a single Organization), Order has a join up to Recipient (each recipient can be related to multiple orders), and OrderItem has a join up to Order (each Order can have multiple OrderItem records).



The **_data** object is composed of a nested series of objects, each one representing a table.

When sending data to the ADVANCED EVENT TRIGGER endpoint, the root node within **_data** must reference the Table Name of the target table (i.e., the source table) for the Campaign you wish to trigger. For instance, if the Campaign you wish to trigger was created targeting the table named "recipient," then the root node of **_data** must be "recipient."

Within each table object in **_data**, you can use three types of fields:

- Table fields (like first name, last name, email address, mobile phone number, or Push Registration ID on a Recipient table)
- "Upward" joins (for a join from the current table up to a parent, like joining Recipient up to Organization); please note that the term "upward" is relative to whichever table is your reference point.

- Child joins (joins from "child" tables up to the current table, like a join from Order up to Recipient).

> **Note**
>
> If your Event-triggered Campaign uses a Dynamic Block with rules based on Filters, the values that you include in the AET message payload are case-sensitive. For example, if the Filter is looking for a value of "TRUE" in a particular field, and you submit a value of "true," the Filter will not select that record. The best practice is to ensure that the data being passed is case-matched. Or, optionally, you can adjust your Filter logic to include "in" logic, such as "Flag in 'TRUE,' 'true, 'True'" in order to accommodate variations in case.

In regard to joins in Messaging -- joins represent a relationship between two tables, and they tell us how those two tables are related. In Messaging, all joins are "many-to-one," meaning that one record in the first table can be linked to many records in the second table. However, no record in the second table can be related to more than one record in the first table via that same join. Further, it should be noted that joins in Messaging are created against, and owned by, the "many" table.

Using the table diagram above, we can see a join between Recipient and Organization. This join might be named "recipient_to_organization," and it was created on the Recipient table (the "many" table), and it points to the Organization table. This join tells us that many recipients can be members of the same organization, but no recipient can be a member of two organizations (at least, not using this same join).

In the figure below, you can see the structure for the **_data** section. First, notice that the root node within **_data** is named "recipient," and that it's a collection of objects. At the current time, AET doesn't support multiple records in this root node collection (i.e., sending bulk or multiple messages at once, or inserting multiple records at once, for the target table).

Example:

```
"_data": {
                                                    "recipient": [
                                                                  {
                                                "name_first": "John",
```

```
                                                    "name_last": "Smith",
                                                "email": "john.smith@gmail.com",
                                                    "recipient_to_organization":
                                                                                    {
//An upward join
                                            "organization_name": "Cheetah Digital",
                                                  "organization_city": "Costa Mesa"
                                                                                    },
                                                   "order.order_to_recipient":
                                                                                    [
// A child join
                                                                                   {

"order_no": "2834737",

"total": "129.99",
                                                                                  },
                                                                                   {

"order_no": "2834738",

"total": "59.99",
                                                                                   }
                                                                                    ]
                                                                                   }
                                                                                    ]
                                                                                   }
```

```
}
```

The above example illustrates all three types of fields:

- **Table Fields** -- 'name_first,' 'name_last,' and 'email' are all fields on the Recipient table.
- **Upward Joins** -- "recipient_to_organization" is an upward join from the Recipient table to the Organization table. The join was created on the Recipient table, and it points at the Organization table. These types of joins will always go from "many" on the current table to "one" on the referenced table.
- **Child Joins** -- Because an table can't have a join that points "down" to a child table, AET uses dot notation to first reference the table that owns the join, and then specify which join on that table defines the specific relationship between these two tables. The dot notation convention looks like <table_name>.<join_name>, such as "order.order_to_recipient" in this example. This join tells us that each recipient can have many orders, but no order can belong to more than one recipient.

You can also see in this example that the recipient being targeted here has multiple orders being submitted. Using the join from Order to Recipient, we can pass through all of the related data needed for the message for that given Recipient.

Below is another example that shows a recipient record linked to the Order table two different times using two different joins.

First, through the same join we see above ("order_to_recipient") and second, through another join named "order_to_recipient_return_order." The second join still uses the Order table, but it will associate it to the Recipient through a different join to be called out in different areas of the campaign.

Example

```
"_data": {
                                                    "recipient": [
                                                                  {
                                                "name_first": "John",
                                                "name_last": "Smith",
```

```
                                    "email": "john.smith@gmail.com",
                                         "order.order_to_recipient":
                                                                        [
// a child join
                                                                         {

"order_no": "2834737",

"total": "129.99",
                                                                       },
                                                                         {

"order_no": "2834738",

"total": "59.99",
                                                                         }
                                                                          ],

"order.order_to_recipient_return_order":
                                       [       // a second child join
                                                                         {

"order_no": "2834737",

"total": "129.99",
```

```
                                                                    }
                                                                     ]
                                                                    }
                                                                     ]
                                                                    }
}
```

The next example shows the **_data** section referencing multiple Orders that each have Items associated with them. This example shows how the relational structure can be used to reference multiple nested child elements.

Example:

```
"_data": {
                                                          "recipient": [
                                                                    {
                                                  "name_first": "John",
                                                  "name_last": "Smith",
                                        "email": "john.smith@gmail.com",
                                             "order.order_to_recipient":
                                                                     [
// A child join
                                                                    {
                                                 "order_no": "2834739",
                                                      "total": "59.99",
                                       "orderitem.orderitem_to_order":
                                                                     [
//A nested child join
                                                                    {
                                               "orderitem_no": "3473657",
                                                  "itemname": "Product 1"
                                                                   },
                                                                    {
```

```
                                                    "orderitem_no": "2345788",
                                                       "itemname": "Product 2"
                                                                    }
                                                                     ]
                                                                   },
                                                                    {
"order_no": "2834740",
"total": "89.99",
"orderitem.orderitem_to_order":
                                                                     [
//A nested child join
                                                                    {
                                                    "orderitem_no": "8794355",
                                                       "itemname": "Product 3"
                                                                    }
                                                                     ]
                                                                    }
                                                                     ]
                                                                    }
                                                                     ]
```

```
                                                                    }
    }
```

Above you can see that we now have two Orders, each with one or more Items. The "orderitem_to_order" join is used within each Order to associate the correct child records.

_campaignMetadata

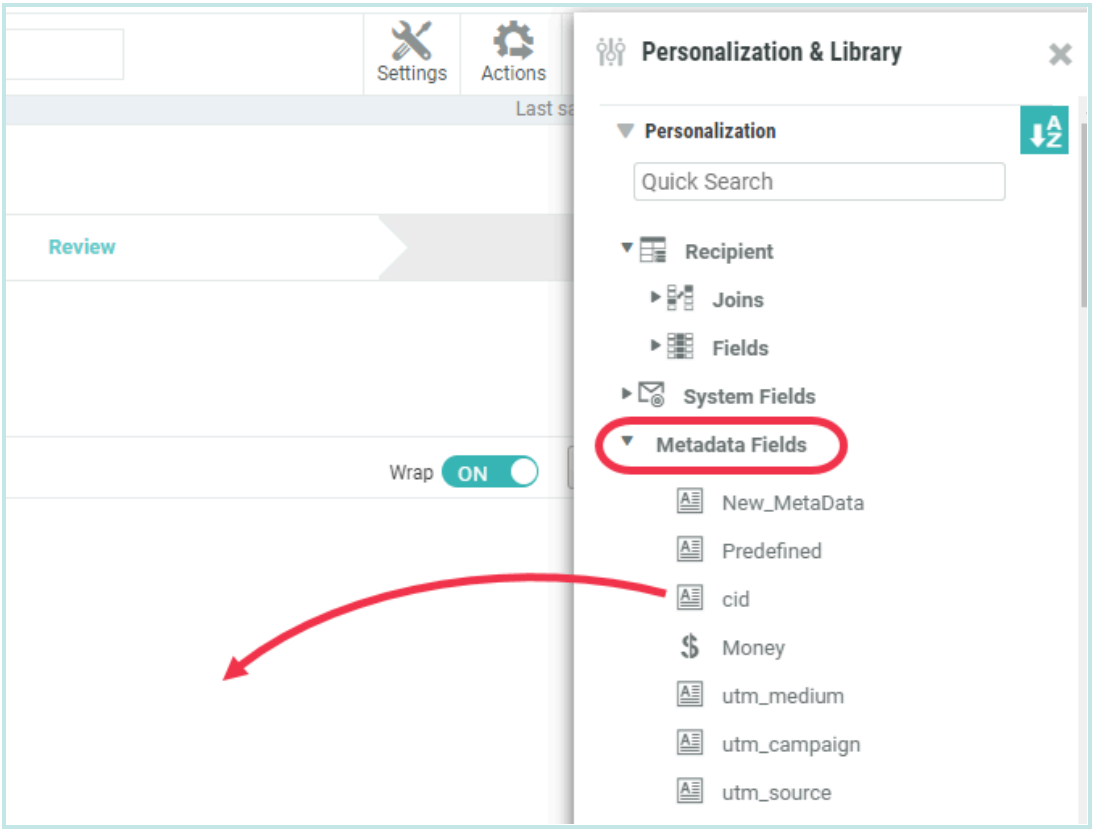
The optional **_campaignMetadata** object can be used to submit data that is never inserted into the platform's tables, but instead contains additional values available for personalization within the Campaign or in Content Blocks (this feature is not supported on the tables that require Looping Blocks to personalize). The objects in this section are explicitly never saved, as they don't relate to physical columns or the relational structure.

The **_campaignMetadata** object is optional, and it's irrelevant if using the ADVANCED DATA LOAD endpoint to only load data (and not to trigger a Campaign).

It's important to note the distinction between metadata that's provided through the AET message (referred to as "**record metadata**"), and the metadata feature that's available within the Messaging application (referred to as "**Campaign Metadata**"). The record metadata provided through AET is not defined or saved anywhere in the platform. Conversely, the Campaign Metadata feature available within the application allows you to define Metadata fields, to assign them a value at the Campaign level, and to then add those fields to Campaign content. Both options (described below) can be used in an AET request message.

Adding Campaign Metadata

In the Campaign screen in Messaging, drag the desired Campaign Metadata field from the Personalization Pane, and drop into the desired location in the message content. The Personalization Pane contains all of the Campaign Metadata fields defined in your account.



The platform inserts the Merge Symbol for this field along with the prefix "meta:" For example:

```
<table width="100%" border="0" cellspacing="0"
  <tr>
    <td align="center">

    {(meta:cid)}

<table width="550" border="0" cellspacing="0"
    <tr>
```

Next, you need to provide the Metadata field name and value within your message payload, being sure to use the correct field name.

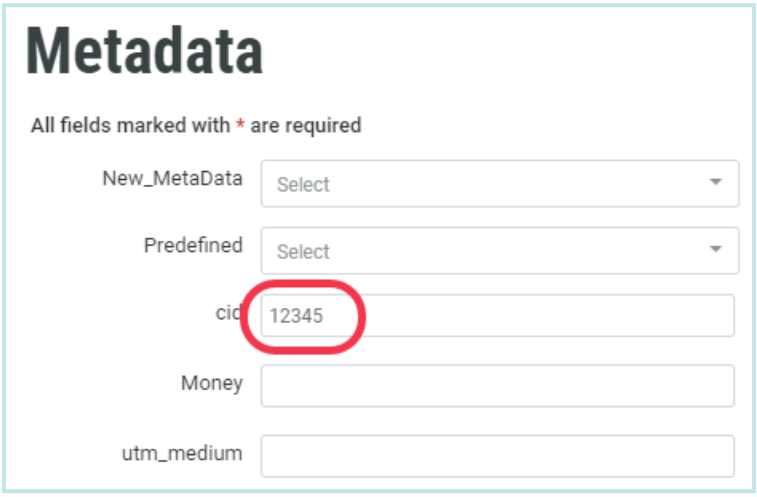
Example:

```
"_campaignMetadata":
                    {

"cid": "67890"
                    }
```

The value you provide in the message payload will get inserted into the message content, at the location of the Merge Symbol.

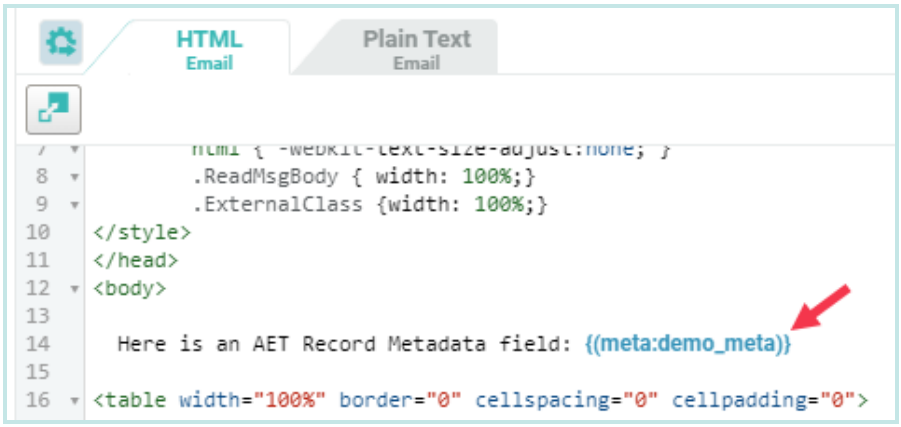
Please note that a value you provide in the API call will always take precedence over any value defined for this Metadata field within the Campaign. For example, let's say you defined a value of "12345" for the Metadata field "cid" within this Campaign.



But in the AET request message, you passed a value of "67890" for this field. The platform would personalize the Campaign content using the value passed in the AET call, overriding the value defined at the Campaign level.

Adding Record Metadata

You can personalize the message content using metadata fields that aren't defined in your account. Simply enter a custom Merge Symbol for the field in the desired location within the Campaign content. This Merge Symbol must be surrounded by double-brackets, and use the prefix "meta" followed by a colon, then the field name. For example: {(meta:demo_meta)}.



Please note that the Merge Symbol you use for this purpose should be different than any of the existing Campaign Metadata fields.

Next, you need to provide the Merge Symbol name and value within your message payload, being careful to use the same field name.

Example:

```
"_campaignMetadata":
                    {
"demo_meta": "SAVE15"
                    }
```

The value you provide in the message payload will get inserted into the message content, at the location of the Merge Symbol. This value will not be loaded into the Messaging database.

_importOptions

The optional **_importOptions** object can be used to specify import behavior on the ADVANCED EVENT TRIGGER endpoint. This object is necessary only when you want to apply special rules when importing data.

When an inbound record matches on a primary (or unique) key to an existing record in the database, the inbound record becomes an "update" instead of an "insert." Within the platform, you can use a Data Map to specify how each field should react in that case. The ADVANCED EVENT TRIGGER endpoint supports the same behavior, using the **_importOptions** object instead of a Data Map. In normal operation, all such options are disabled, or "false,"

by default. This object, and each individual option within it, only needs to be included and specified if you need some behavior other than the default behavior.

The **_importOptions** object should be flat. You don't need to nest the related tables as they were nested in the **_data** object. Each table that needs special import options can appear only once per join used to reference it. For example, if you want to define import behaviors for each of the tables in the **_data** example used above, the **_importOptions** object would have sub-sections for each for the tables used.

In addition, if you define **_importOptions** for the schema described above in the **_data** section, you can declare separate import behaviors for each reference to the Order table. The tables in this case must be differentiated with a declaration of the join in the table reference. Ultimately, for the sample schema above, you could have up to three objects in the **_importOptions** object.

Example:

```
_"importOptions":
                                                                [
                                                                 {
                                                "_table": "recipient",
                                             "_doNotUpdateExisting": false,
                                                "_minimalImport": true
                                                                },
                                                                 {
                                                    "_table": "order",
                                             "_join": "order_to_recipient",
                                             "_doNotUpdateExisting": false,
                                                "_minimalImport": true
                                                                },
                                                                 {
                                                    "_table": "order",
                                  "_join": "order_to_recipient_return_order",
                                             "_doNotUpdateExisting": false,
                                                "_minimalImport": true
                                                                }
                                                               ]
```

The parameters in the **_importOptions** object are described below in more detail.

_table

This string parameter is optional.

The **_table** parameter represents the name of the table for which you're defining import behaviors.

_join

This string parameter is optional.

If you're defining import behaviors for a table that's joined to the Campaign source table, you must use the **_join** parameter to indicate the name of the table join.

_doNotUpdateExisting

This Boolean parameter is optional.

Applied at the table level, the **_doNotUpdateExisting** flag dictates whether the submitted information should be imported if a record with a matching unique key has been found in the database. Setting this to "true" means that only new records will be created, and existing records will not be updated. If you don't provide this parameter, the system defaults the value to "false."

- "True" = Create new records only
- "False" = Update/Create records (default)

_minimalImport

This Boolean parameter is optional.

Applied at the primary sending table level, the **_minimalImport** flag indicates whether to load ALL of the data provided in the API payload, or whether to perform a "minimal load." A minimal load means the only data that will be imported and/or updated into the database are the fields that compose the Unique Identifier (also known as the Alternate Key ID , or AK_ID), and the to_address field. If you don't provide this parameter, the system defaults the value to "false."

- "True" = Perform a "minimal load" into the sending table.
- "False" = Load all data provided in the API payload.

_fieldOptions

The **_fieldOptions** object is used to define field-level import behaviors. This object should be nested beneath the **_importOptions**, for the desired table. For example:

```
"_importOptions":
                                                                    [
                                                                     {
                                                    "_table": "recipient",
                                                "_doNotUpdateExisting": false,
                                                      "_minimalImport": false,
                                                             "_fieldOptions":
                                                                      [

                                                                    {


"_applyToFields":


                                                                    [


"name_first",


"name_last"


                                                                  ],


"_insertNull": true,


"_ignoreCase": true


                                                                  },
```

```
                                                                    {

    "_applyToFields":

                                                                    [


    "create_date"

                                                                  ],

    "_preserveData": true

                                                                    }
                                                                     ]
                                                                    }
                                                                     ]
```

_applyToFields

This array is optional.

The **_applyToFields** array contains the **Column Name** for the field (or fields) for which you are defining import behavior. You must use the field's internal system name in this array, and not the field's user-friendly display name..

_preserveData

This Boolean parameter is optional.

Setting the **_preserveData** parameter to "true" will preserve existing data if this is an update to an existing record. With this option, you could personalize a Campaign by passing data that's different than the data in the database for a given recipient, without having to overwrite the existing data. If you don't provide this parameter, the system defaults the value to "false."

- "True" -- Do not overwrite existing data
- "False" -- Overwrite existing data

_insertNull

This Boolean parameter is optional.

Setting the **_insertNull** parameter to "true" will overwrite data in the database as NULL if the supplied value is empty. Using this option, you can "blank out," or delete, existing data for a record that may no longer be valid. If you don't provide this parameter, the system defaults the value to "false."

- "True" -- Overwrite existing data as NULL if parameter value is empty.
- "False" -- Don't overwrite existing data if parameter value is empty.

_caseSensitive

This Boolean parameter is optional.

The **_caseSensitive** parameter is valid only for Unique Identifier and Primary Key fields. Setting this parameter to "true" will instruct the system to perform case-sensitive comparisons to the value in this field. If you don't provide this parameter, the system defaults the value to "false."

- "True" -- Perform case-sensitive comparisons to this field.
- "False" -- Perform case-insensitive comparisons to this field.

_ignoreBanList

This Boolean parameter is optional.

The **_ignoreBanList** parameter is valid only for fields with a Data Type of "Email." Setting this value to "true" will instruct the system not to check this email field against the platform's Global and Custom Email Ban lists. If you don't provide this parameter, the system defaults the value to "false."

- "True" -- Skip the Ban List validation for this email field.
- "False" -- Run the Ban List validation for this email field.

_preserveOptOut

This Boolean parameter is optional.

The **_preseveOptOut** parameter is valid only for fields with a Data Type of "Preference." When set to "true," this parameter forbids the new data in the API message from overwriting a current opt-out preference status. That is, once a recipient has opted-out, he or she can't re-opt-in again if this flag is set to "true." If you don't provide this parameter, the system defaults the value to "false."

- "True" -- Don't use the value in this field to re-opt-in a recipient who previously opted-out.
- "False" -- Allow the system re-opt-in a recipient who previously opted-out.

_appendMultiValue

This Boolean parameter is optional.

The **_appendMultiValue** parameter is valid only for multi-value fields. When set to "true," this parameter instructs the system to append new data to this field, rather than overwriting it.

- "True" -- Append values to this field.
- "False" -- Overwrite value in this field.

_attachments

The **_attachments** object is optional, and can be used to define files that are to be included as attachments in the email message to the recipient (this feature is supported only in the Email channel). Attachments sent via the ADVANCED EVENT TRIGGER endpoint are not actually stored within the Messaging platform, but are instead passed from the API request to the email message.

> **Note**
>
> The AET attachments feature must be enabled within your client account, and also within the Event-triggered Campaign itself. Please speak with your Client Services Representative to enable the feature at the account level. For more information on enabling AET attachments at the Campaign level, please see Attachments.

The request message may contain up to five attachments, with a total combined file size of two megabytes. The platform supports the following attachment file types: .pdf, .xls, .png, .jpg, .gif,.ico, .txt, and .zip. The contents of the attachments must be base64-encoded.

For example:

```
"_attachments":
                                                                    [
                                                                     {
                                                "_fileName": "File1.pdf",
                                             "_mimeType": "application/pdf",
                                "_fileContents": "VGhlIHZlcnkgfdsjfadsipfZSdfdsdSg=="
                                                                    },
                                                                     {
                                              "_fileName": "LogoImage.png",
                                                  "_mimeType": "image/png",
                                 "_fileContents": "VGhlIHZlcnkgZmlycdfdfdfbGluZSENCg"
                                                                     }
                                                                    ]
```

The parameters in this object are described below in more detail.

_fileName

This string parameter is required if using attachments.

The **_fileName** parameter represents the name of the attachment.

_mimeType

This string parameter is required if using attachments.

The **_mineType** parameter represents the MIME type of the attachment. A MIME type is a way of identifying files on the Internet according to their nature and format.

_fileContents

This string parameter is required if using attachments.

The **_fileContents** parameter contains the base64-encoded contents of the attachment.

_responsePayload

This Boolean parameter is optional.

By default, Messaging will return the entire request payload in the success response message. Optionally, you can use the **_responsePayload** object to instruct the system to return a limited response message, that doesn't include the entire request payload.

If the **_responsePayload** object isn't included in the request message, the platform defaults it to "true," meaning that the response message will contain the entire request payload.

Example:

```
"_responsePayload": false
```

_campaignId

This integer parameter is required only if the Campaign ID isn't provided in the URL.

The platform supports two variants of the endpoint -- one that contains the Campaign identifier within the URL itself, and another one that contains the Campaign identifier within the **_campaignID** parameter in the message payload. The payload variant is intended for use by clients who need a stable URL that doesn't change with each AET Campaign.

If the Campaign ID is provided in both the URL and the **_campaignId** parameter, the two values must match, or else the system will return an error message.

The **_campaignId** parameter represents the **Running Campaign ID** of the Event-triggered Campaign that you want to trigger with this API message.

Example:

```
"_campaignId": "28595"
```

_requestTimeout

This integer parameter is optional.

The **_requestTimeout** is used to define a timeout parameter (in seconds). If the indicated duration elapses (using the API request timestamp as the starting point), and the platform still hasn't sent back a response message, then the platform drops the request, and does NOT trigger the Campaign deployment. The client can then safely retry the request, without the possibility of sending duplicate Campaign messages.

The platform allows you to define a client-level timeout parameter (this value must be configured by your Client Services Representative), or you can use **_requestTimeout** to provide this timeout parameter at a Campaign level.

A value provided in the **_requestTimeout** parameter will override the client-level setting. If you don't provide the **_requestTimeout** parameter in the request payload, the system will use the client-level setting instead (if defined). If you don't have a client-level setting defined, the platform defaults to 300 seconds.

The valid values for the **_requestTimeout** parameter are integers between 1 and 300.

Example:

```
"_requestTimeout": "60"
```

Payload Validation

Because the insert to the database happens independently of the message sending process, Messaging needs to be sure that the import will succeed before returning a "success" response message. To this end, the platform will validate the schema and data types of the submission. Any problems with the payload will result in an appropriate error response message; see the **Payload Errors** section for more details on these messages.

The message is checked to make sure all tables exist, are appropriately nested, and are joined correctly. In addition, the platform will check that nested child records that require primary keys have those records in their parent chain.

Further, all submitted data will be checked to confirm that it matches the data type specified in the platform for the given field.

However, the ADVANCED EVENT TRIGGER endpoint does NOT validate that all necessary data is present for the Campaign being triggered. The onus for sending the correct data required for a specific Campaign is on the caller. In the event that some field necessary for content personalization is missing in the request payload, that value will simply be blank in the resulting message.

4 Response

This section describes all of the possible response messages sent back from the ADVANCED EVENT TRIGGER endpoint .

Success

A successful response to a POST message will generate a response code of "200," followed by the entire request message payload.

Optionally, you can use the **_responsePayload** parameter to receive only a "limited" response message, which will not return the full request message payload (see **responsePayload** for details on this feature).

The response message will also include a few additional parameters, as described below:

- **messageId**: This parameter contains the system-generated Message ID for the triggered message. This parameter is used only if you have disabled **AET Tracking Parameters** for the Campaign.
- **messageIdString**: This parameters uniquely identifies the delivered AET message. This parameter is used only if you have enabled AET Tracking Parameters for the Campaign.
- **sendTime**: This parameter contains the date / time when the triggered message was deployed.

In addition, you may see the following parameter if the system encountered any issues with the provided **_attachment** object:

- **message**: This parameter contains error message details regarding the attachment.

Authorization Errors

The following table lists the possible error messages that might be returned if the authorization step fails.

Error Code	Error Message	Possible Solution
401	Authorization has been denied for this request.	Get a token from the Token endpoint and pass it to the Trigger endpoint in an Authorization header, with the value "Bearer: {token}".
403	Forbidden to access campaign/trigger on this server. Please use correct URL.	Find out the correct URL for AET and use that in the request.

Payload Errors

The following table lists the possible error messages that might be returned if the payload validation step fails.

Error Code	Message Code	Error Message	Possible Solution
400		Sender Profile is not set for the Campaign.	Select valid Sender Profile for the Campaign.
400		"To" address is not set for the Campaign.	Select valid "To" address in the Campaign, or make sure passed "To" address is valid.
400		"From" address is not set for the Campaign.	Select valid "From" address in the Campaign.
400	107	Campaign does not exist.	Selected Campaign is not started, or is not an Event-triggered Campaign; be sure to use the Running Campaign ID, and not the Campaign's Object Reference ID.
400	108	Campaign should be Advanced Event Trigger type.	Select "Advanced Event Trigger" as the trigger type for the Campaign.
400	109	Campaign is in status {0}. Campaign should be in launched status.	Select Campaign is not running. Start Campaign or wait until status has changed to RUNNING.

Error Code	Message Code	Error Message	Possible Solution
400	105	There is no matching column in data for the Campaign's sending value. Please include field '{0}' on the table.	"To" field is not part of payload or not part of selected table. Add "To" field to payload or add needed field to table.
400	104	You can only include one joined record for table '{0}' since the Campaign is sending from this table.	The payload contains more than one record for the sending table. AET can send to only one recipient at a time. Reduce joins in payload to send to only one recipient.
400	103	There is no matching table - expecting {0} as root level table.	Change the table name in the root of the payload to match the name of the table that is associated with the specified campaign ID.
400	101	Field {0} does not exist for table {1}.	Field passed in payload doesn't exist in the table. Change field in payload, or add field to table.
400	101	Table {0} does not exist.	Choose correct table, or create table.
400	101	You must not send a blank submission.	Add data to submission (might be caused by incorrect JSON).
400	101	Join {0} is not a valid join. It does not Join table {1} to table {2}.	Join exists but doesn't join tables used in payload; Use correct join, or correct tables.
400	101	Join {0} is not a valid join. It does not exist.	Use correct join, or correct tables.
400	101	Join {0} is invalid. Table {1} does not exist.	Invalid join used. Create valid join, or add missing table.
400	101	You must include the 'Join' field with the value of the join name.	Join added incorrectly to payload (refer to the AET Technical Guide for more details on how to define the joins in the message payload.
400	101	All fields that make up the Alternate Key sequence key of table '{0}' are not present. You must include all Alternate Key sequence fields: '{1}'	Not all fields that are needed to create the Alternate Key are passed in the message. Add missing Alternate Key fields to the payload.
400	101	All fields that make up the Alternate Key sequence key of table '{0}' do not have values. You must	Some (or all) of the fields for the Alternate Key have a null or empty value. Add a valid value for all Alternate Key fields in the payload.

Error Code	Messag e Code	Error Message	Possible Solution
		provide values for all Alternate Key sequence fields: '{1}'	
400	101	You cannot set the Null Flag Update Option on a Primary Key or Alternate Key Field.	Primary Key and Alternate Key fields need to be part of the update option.
400	118	Advanced Event Trigger not enabled. Please contact administrator.	Contact an administrator to confirm that you have AET access; ensure that the correct username is being used when getting an API token.
400		The "apiData" parameter is null. This can be caused by an improperly-formatted payload.	Confirm that the payload is not blank. Confirm that your payload has properly-formatted and complete JSON or XML, including delimiters. Also confirm that the Content-Type header you are passing matches the payload type; for JSON, it should be "application/json"; for XML, it should be "application/xml".
400		Your content is not valid. Please check {key}	Confirm that the payload does not contain multiple entries for the specified key within one JSON object. This message might also appear if the payload is missing a required element such as "_data".
400		The field _campaignId is missing in the payload.	The payload needs a **_campaignId** key and value in the payload to specify what campaign to trigger.
400		The field _campaignId: {value} in the payload is invalid.	The value for **_campaignId** must be greater than or equal to 1.
400		The campaign Id provided: {value} in the URL is invalid.	The value for **campaignId** in the URL must be greater than or equal to 1.
400		The field _campaignId in the payload is not matching with the campaignId in the URL.	The values for the Campaign ID within the URL and **_campaignId** in the payload are not equal. You could make them equal or remove one.
400		Cannot have an array at parent level join : {0}	This error message seems to indicate the wrong problem. What is actually expected is an array, but an object was found.

Error Code	Messag e Code	Error Message	Possible Solution
400	106	Message cannot be sent. The given email address was rejected by Email Ban Mask.	It's possible that the ban masks in the account are wrong -- that can be checked in the UI -- but it's more likely that the ban masks are correct and that you are trying to send to an email address that is on a banned list. There is nothing wrong with the payload, but the request can't succeed with the specified email address.
400		Attachments are not enabled for this customer on Advanced Event Triggers. Please contact support with further questions.	The use of attachments in an AET request must be enabled at a client level, and also at a Campaign level.
400		The \"_requestTimeout\" parameter value must be between 1 and 300 seconds."	The valid values for **_requestTimeout** are integers between 1 and 300.
400	120	Attachments are not supported by AET push campaign.	The Attachments feature is supported only by the email channel.
400	124	Attachments are not supported by AET SMS campaign.	The Attachments feature is supported only by the email channel.
400	101	{0} is not a valid Guid.	Invalid or unknown GUID.
400	117	Invalid data: Push Device not registered in the system. Please check Push Registration ID	Unknown Push Registration ID.
400	101	{0} is not a valid Phone Number.	Invalid phone number.
400	122	To phone number field is missing	Unknown or missing mobile phone number field.
400	125	SMS message content is empty	SMS Text message content can't be blank.
400		SMS message content is too long. Content should not exceed 2680 bytes	SMS Text message content is too long.
400	126	AET is not enabled for Push Channel for this customer. Please contact administrator.	The use of the Push Notification channel in an AET request must be enabled at a client level. Contact your Client Service Representative.

Error Code	Messag e Code	Error Message	Possible Solution
400	127	Push Registration Id provided is not registered with a valid Push Application.	Make sure PRID is associated to a valid Application ID.
400	128	AET is not supported for {channelType}. Please enable it in config.	The use of AET Push/SMS must be enabled at a client level. Contact your Client Service Representative.
400	110	Attachments have not been enabled for this campaign. {0}.	The use of attachment in an AET request must be enabled at a campaign level.
400	111	Total attachment size exceeds the maximum allowable {0} MB	Max attachment size must not exceed client level settings.
400	112	Max allowable attachments for an AET request is {0}. Your request exceeded that maximum with {0} attachments.	Max attachment count must not exceed client level settings.
400	113	Message not sent and data not inserted. Please use the data insert endpoint to update data.	When using opt-out status for the recipient in the AET payload, user should use the ADVANCED DATA LOAD endpoint to update the sender profile status to update the data.
400	115	Channel Type {channelType} not supported by Advanced Event Trigger	AET support only Email, Push, and SMS (coming soon) channels.
400	121	AET is not enabled for SMS Channel for this customer. Please contact administrator.	The use of SMS in an AET request must be enabled at a client level. Contact your Client Service Representative.
400	101	Invalid data: Apple authentication certificate has already expired for this applicationId={0} , authenticationName={1}	Make sure the iOS certificate used to send push notification is not expired.
400	101	Invalid data: Notification size exceeded limit. Maximum allowable by APNS, {} bytes	Max notification size must not exceed client level settings.
400	110	_fileContents is missing.	Invalid space found within the **_fileContents** parameter name
400	110	Invalid File content - expects base64 string.	Invalid **_fileContents**. Parameter must be base64-encoded string.

Error Code	Messag e Code	Error Message	Possible Solution
400	110	_fileName is missing.	Invalid space found within the **_fileName** parameter name
400	110	File will not be imported because it is of a type that has not been whitelisted. Please contact a system administrator if you believe this type of file should be allowed.	File is not whitelisted as per whitelisted rules.
400	110	_mimeType is missing.	Invalid space found within the **_mimeType** parameter name
400	101	Field \'{0}\' must not be greater than {1} characters.	Max length for data type "string" is 255 bytes; max length for data type "long string" is 8000 bytes.
400	101	{0} must use 8 bit code page converatable characters	String field value should be compatible to convert UTF-16.
400	101	{0} is not a valid Email Address.	Invalid Email Address.
400	101	{0} must be between {1} and {2} and cannot be {3}. Use Update Options to set the field to null.	Make sure the field value is within the data type limit.
400		The top level data object must have one property the name of the root table (e.g. \"recipient\") and value - an array with one element representing a single row from that table.	Make sure there is an array with one element under **_data** section. More than one array is not supported.
400	129	There is no valid value in data for the campaign's sending value. Please include valid value for field '{0}'.	The "to prop" field value is empty or null. For email channel, its Email field; for Push channel, its PRID field.

Other Errors

The following table lists the other error messages that might be returned in response to your request message.

Error Code	Error Message	Possible Solution
500	The 'ObjectContent`1' type failed to serialize the response body for content type 'application/xml; charset=utf-8'.	The Accept header is set to return XML, but the response data could not be serialized to XML. Setting the Accept header to "application/json" could be the solution.
500	The top level data object must have one property the name of the root table (e.g. \"recipient\") and value - an array with one element representing a single row from that table.	Confirm that the "_data" object in the payload contains an array property for the table, and an element for the row to insert.
500	PkId is null for targeting entity-{0}	It could be that fixing an error in the format of the value of the unique identifier property would prevent this, but it's more likely that you will just need to contact an administrator.
500	Content does not exist.	You might need to wait up to 10 minutes for a cache item to expire. If it does work after 10 minutes, contact an administrator.
500	Property not found for case dynamic block.	The property name could have changed after the campaign was launched. A possible solution is to run Pick Up Changes on the campaign.
500	Pairs {0} and {1} do not match.	Try re-saving the campaign and running Pick Up Changes to fix this campaign content error.
500	Empty block {0}{1} found.	Try re-saving the campaign and running Pick Up Changes to fix this campaign content error.
500	Block symbol found within content (cont_id={0}) without parts	Try re-saving the campaign and running Pick Up Changes to fix this campaign content error.
500	Child content (cont_id={0}) not found in content parts	Try re-saving the campaign and running Pick Up Changes to fix this campaign content error.
500	Error in GetParsedContent	Try re-saving the campaign and running Pick Up Changes to fix this campaign content error.

Error Code	Error Message	Possible Solution
500	Invalid MailAddress parts: name={0}, address={1}	The most likely cause of this error is that the specified email address for the message has invalid characters -- particularly double-byte characters.
500	Not Win_1252 character at position {0}	Remove non-Windows-1252-encoding characters from the email subject line -- you might need to run Pick Up Changes for this if it is done in the UI.
500	Request timed out, try again	The platform did not send the API response message within the timeout period specified in **_requestTimeout**
500	Property not found for case dynamic block	Verify that the Dynamic Block referenced in the payload is built off the same table as the Campaign.

Best Practices for Performance

This section contains our best practices and suggestions for getting the best performance from the ADVANCED EVENT TRIGGER API.

- Re-use the same authentication token for all AET requests until that token expires. Requesting a new token adds time to each AET request, and increases load on the API servers.
- Send your AET requests in parallel, using multi-threading or multiple servers. While making requests serially can work fine, once you reach certain request volumes, you'll see diminishing returns with the serial approach.

5 Sample Message

Overview

This section provides an example of an Advanced Event Trigger request message.

Sample JSON Request (Email)

Below is a sample of an Advanced Event Trigger request message in JSON format. This message uses a simple **_data** structure that references only one table.

```
{
"_data":{
"recipient": [
                                                                    {
                                                "name_first": "John",
                                                  "name_last": "Doe",
                                        "email": "john.doe@cheetahdigital.com"
                                                                    }
                                                                  ]
                                                                 },
"_campaignMetadata":
                                                                    {
"meta_coupon_code": "COUPON123"
                                                                    },
"_importOptions":
                                                                   [
                                                                    {
                                                "_table": "recipient",
                                                      "_fieldOptions":
                                                                    [
```

```
                                                                    {

"_applyToFields":

                                                                    [

"name_last"

                                                                   ],

"_insertNull": true

                                                                    }
                                                                     ]
                                                                    }
                                                                     ],
"_campaignId": "28595",
"_responsePayload": true
                                                                    }
```

Sample JSON Request (Email)

Below is a sample of an Advanced Event Trigger request message in JSON format. This message uses a more complex **_data** structure with joins to several additional tables.

```
{
   "_data":{
      "recipient":[
         {
            "name_first": "John",
            "name_last": "Smith",
            "email": "john.smith@example.com",

                                                "recipient_to_organization": {
```

```
                "oraganization_name": "Cheetah Digital",
                "organization_city": "Costa Mesa"

            },

         "order.order_to_recipient":[
            {
               "order_no": "2834737",
               "total": "129.99",
               "email": "john.smith@example.com",
               "order_items.order_items_to_order":[
                  {
                     "item_name": "Misc Clothing",
                     "price": "29.99",
                     "size": "Medium"
                  },
                  {
                     "item_name": "Nice Shirt",
                     "price": "79.98",
                     "size": "Small"
                  }
               ]
            }
         ],
         "order.replacement_order_to_recipient":[
            {
               "order_no":"2837473",
               "total":"0.00",
               "trackingNumbers":"78542",
               "order_items.order_items_to_order":[
                  {
                     "item_name": "Nice Shirt",
                     "price": "79.98",
                     "size": "Medium"
                  }
               ]
            }
         ]
      }
   ]
},
"_campaignMetadata":{
   "promotion_title": "Labor Day Sale Deals",
   "promotion_code": "LD2015"
},

"_attachments":[
   {
     "_fileName": "File1.pdf",
     "_mimeType": "application/pdf",
     "_fileContents": "VGhlIHZlcnkgZmlyc3QgbGluZSENCg=="
   },
   {
     "_fileName": "File2.pdf",
     "_mimeType": "application/pdf",
```

```
          "_fileContents": "VGhlIHZlcnkgZmlyc3QgbGluZSENCg=="
       }
       ],
    "_importOptions":[
       {
          "_table": "recipient",
          "_doNotUpdateExisting": false,
          "_fieldOptions":[
             {
                "_applyToFields":[
                   "name_first",
                   "name_last"
                ],
                "_insertNull": true
             },
             {
                "_applyToFields":[
                   "email"
                ],
                "_caseSensitive": true
             }
          ]
       },
       {
          "_table":"order",
          "_joinName":"order_to_recipient",
          "_doNotUpdateExisting": true,
          "_fieldOptions":[
             {
                "_applyToFields":[
                   "order_no",
                   "total"
                ],
                "_insertNull": true
             }
          ]
       },
       {
          "_table":"order",
          "_joinName":"replacement_order_to_recipient",
          "_doNotUpdateExisting": true,
          "_fieldOptions":[

          ]
       }
    ]
"_campaignId": "28595",
"_responsePayload": true,
"_requestTimeout": "60"
}
```

Sample JSON Request (Push Notification)

Below is a sample of an Advanced Event Trigger request message in JSON format. This message is being used to trigger a Push Notification Campaign.

```
{
   "_campaignId":53508,
   "_data":{
      "recipient":[
         {

"push_registration_id":"E9E047A4-A25B-431F-8D9E-478EB4BB4F05",
            "order.order_to_recipient":[
               {
                  "order_no":1,
                  "order_name":"First Order"
               }
            ]
         }
      ]
   },
   "_importOptions":[
      {
         "_table":"recipient",
         "_minimalImport":"false",
         "_doNotUpdateExisting":false
      },
      {
         "_table":"order",
         "join":"order_to_recipient",
         "_doNotUpdateExisting":false
      }
   ]
}
```

Sample JSON Request (SMS Text)

Below is a sample of an Advanced Event Trigger request message in JSON format. This message is being used to trigger an SMS Text Campaign.

```
{
  "_campaignId": 6480,
  "_data": {
    "recipient": [
      {
        "Recipient_id": "2",
        "first_name": "John",
```

```
        "last_name": "Smith",
        "phone": "12235552233",
        "order.order_to_recipient": [
          {
            "order_no":1,
            "order_name": "First Order"
          }
        ]
      }
    ]
  },
  "_importOptions":[
      {
         "_table":"recipient",
         "_minimalImport":"false",
         "_doNotUpdateExisting":false
      },
      {
         "_table":"order",
         "join":"order_to_recipient",
         "_doNotUpdateExisting":false
      }
   ]
}
```

Sample XML Request (Email)

Below is a sample of an Advanced Event Trigger request message in XML format.

```
<?xml version="1.0"?>
<_apiData>
                                                                    <_data>

<recipient xmlns:json="http://james.newtonking.com/projects/json"
json:Array="true">

<name_first>John

</name_first>

<name_last>Smith</name_last>

<email>john.smith@example.com</email>

                                              <recipient_to_organization
xmlns:json="http://james.newtonking.com/projects/json"
json:Array="true">


<oraganization_name>Cheetah Digital</oraganization_name>


<organization_city>Costa Mesa</organization_city>
```

```
                                        </recipient_to_organization>

                                          <order.order_to_recipient
xmlns:json="http://james.newtonking.com/projects/json"json:Array="true
">

<order_no>2834737</order_no>

<total>129.99</total>

<email>john.smith@example.com</email>

<order_items.order_items_to_order json:Array="true">

                                  <item_name>Misc Clothing</item_name>

                                              <price>29.99</price>

                                               <size>Medium</size>

</order_items.order_items_to_order>

<order_items.order_items_to_order json:Array="true">

                                <item_name>Nice Shirt</item_name>

                                              <price>79.98</price>

                                                <size>Small</size>

</order_items.order_items_to_order>

                                        </order.order_to_recipient>

                                <order.replacement_order_to_recipient
xmlns:json="http://james.newtonking.com/projects/json"json:Array="true
">

<order_no>2837473</order_no>
```

```
<total>0.00</total>

<trackingNumbers>78542</trackingNumbers>

<order_items.order_items_to_order
xmlns:json="http://james.newtonking.com/projects/json"
json:Array="true">

                                    <item_name>Nice Shirt</item_name>

                                                <price>79.98</price>

                                                 <size>Medium</size>

</order_items.order_items_to_order>

                              </order.replacement_order_to_recipient>

</recipient>
                                                                    </_data>
                                                                <_attachments>

<_fileName>File1.pdf</_fileName>

<_mimeType>application/pdf</_mimeType>

<_fileContents>VGhlIHZlcnkgZmlyc3QgbGluZSENCg==</_fileContents>
                                                            </_attachments>
                                                             <_attachments>

<_fileName>File2.pdf</_fileName>

<_mimeType>application/pdf</_mimeType>

<_fileContents>VGhlIHZlcnkgZmlyc3QgbGluZSENCg==</_fileContents>
                                                          </_attachments>
                                       <_campaignId>12345</_campaignId>
                            <_responsePayload>false</_responsePayload>
                                                 <_campaignMetadata>

<promotion_title>Labor Day Sale Deals</promotion_title>

<promotion_code>LD2015</promotion_code>
                                                        </_campaignMetadata>
                                                            <_importOptions>

<_table>recipient</_table>
```

```
<_doNotUpdateExistingfalse</_doNotUpdateExisting>

<_fieldOptions>

                          <_applyToFields>name_first</_applyToFields>

                           <_applyToFields>name_last</_applyToFields>

                                      <_insertNull>true</_insertNull>

</_fieldOptions>

<_fieldOptions>

<_applyToFields xmlns:json="http://james.newtonking.com/projects/json"
json:Array="true">email</_applyToFields>

                                <_caseSensitive>false</_caseSensitive>

</_fieldOptions>
                                                          </_importOptions>
                                                           <_importOptions>

<_table>order</_table>

<_joinName>order_to_recipient</_joinName>

<_doNotUpdateExisting>true</_doNotUpdateExisting>

<_fieldOptions xmlns:json="http://james.newtonking.com/projects/json"
json:Array="true">

                             <_applyToFields>order_no</_applyToFields>

                                <_applyToFields>total</_applyToFields>

                                       <_insertNull>true</_insertNull>

</_fieldOptions>
                                                          </_importOptions>
                                                           <_importOptions>

<_table>order</_table>

<_joinName>replacement_order_to_recipient</_joinName>

<_doNotUpdateExisting>true</_doNotUpdateExisting>
                                                          </_importOptions>
</_apiData>
```

6 Appendix A: Identifiers

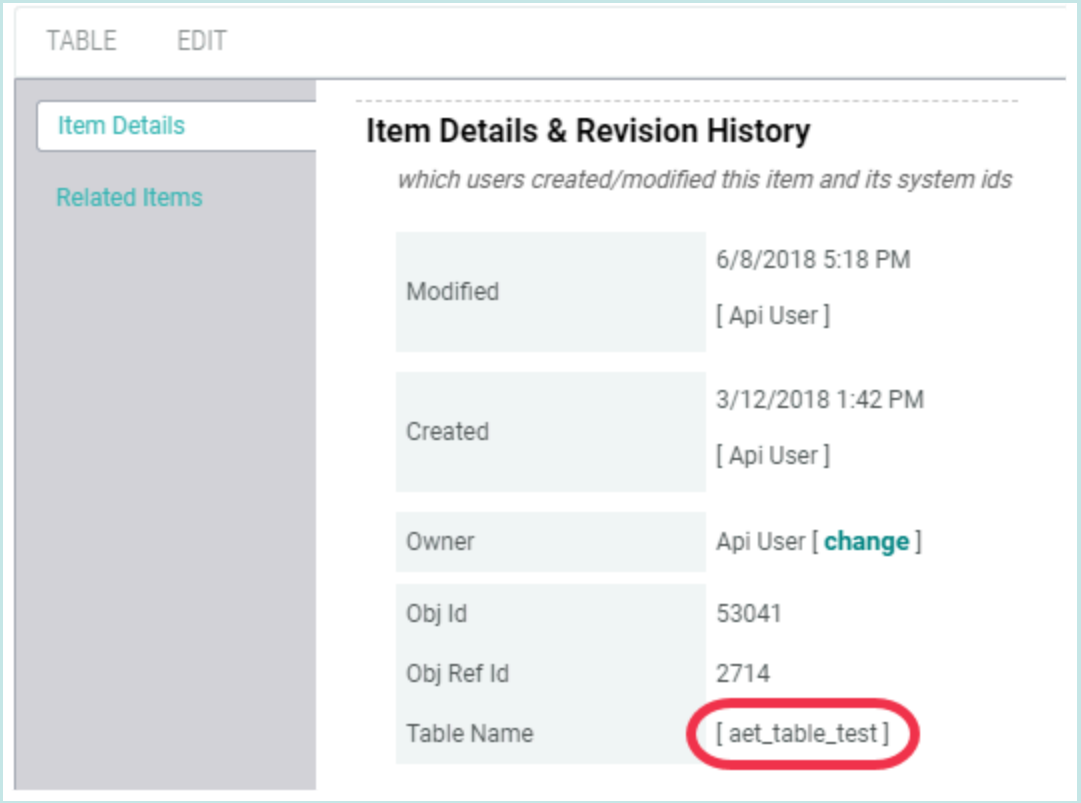
Messaging uses several different types of IDs when referencing assets, such as tables, fields, folders, Filters, and so forth. This appendix describes these different types of IDs, and provides steps on how to look up the value of an ID.

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 ADVANCED EVENT TRIGGER message, you must use the system name in the **_data** object.

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.

Sample Response:

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

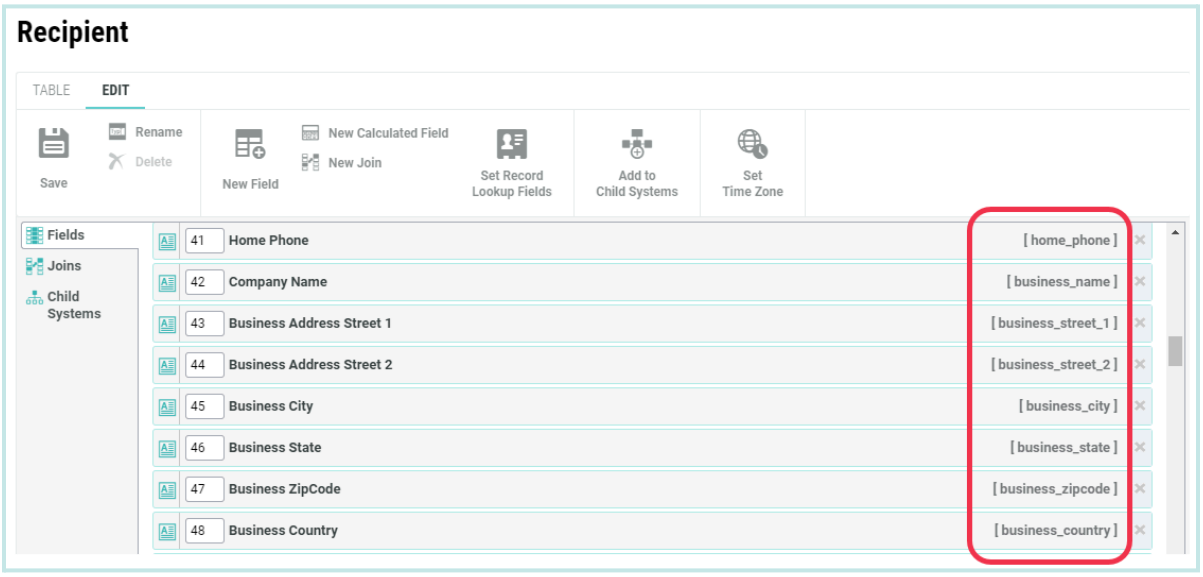
```
    "tableName": "aet_table_test"
}
```

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 Column 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": 2196,
  "entityId": 305,
  "displayName": "Ship Date",
  "propId": 17442,
  "columnName": "ship_date"
}
```

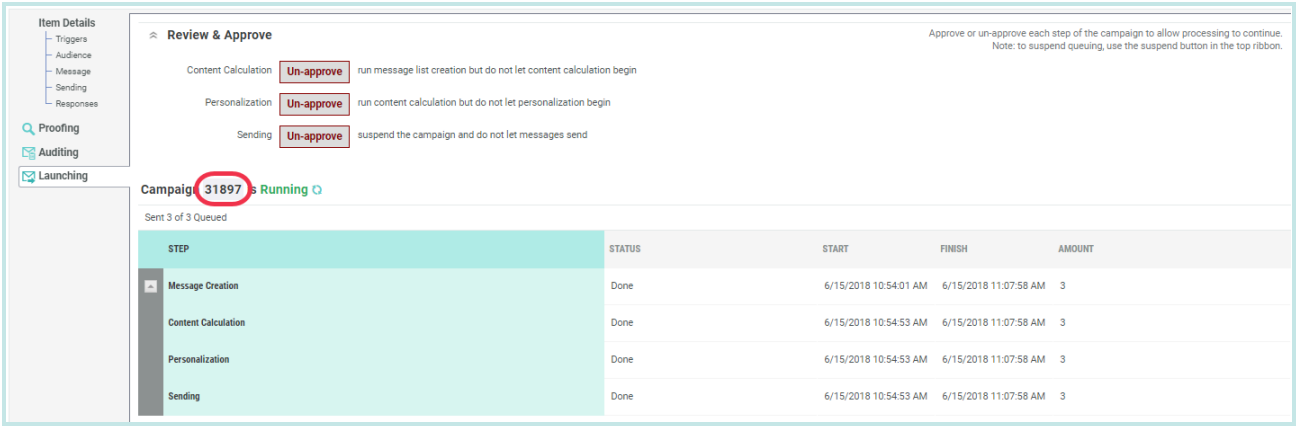
Running Campaign ID

The "Running Campaign ID" is a system-generated identifier for a Campaign, that gets created only when the Campaign is launched. Please note that this ID is different from the Campaign's "Object Reference ID," which is typically used in other Messaging endpoints.

The Running Campaign ID can be found on the Campaign details screen within the Messaging application, or by using the CAMPAIGN STAT endpoint.

To look up the Running Campaign ID within the Messaging application:

1. From the System Tray, select *Campaigns* > *Management* > *Campaigns*.
2. Browse to and select the desired launched Campaign. The Campaign details screen is displayed.
3. In the Function Menu on the left-hand side of the Campaign details screen, click "Launching."
4. The Running Campaign ID is displayed beneath the approval options.



To retrieve the Running Campaign ID using the CAMPAIGN STAT endpoint:

1. Submit a GET request to the CAMPAIGN STAT API endpoint. This request must contain the Campaign's Object Reference ID. For example:

```
https://api.eccmp.com/services2/api/CampaignStat?campId=31896
```

2. Within the API response message, the Running Campaign ID value is provided in the **campId** parameter.

Sample Response:


```
{
"campId": 31897,
"mergeSetupTime": "2018-06-15T15:53:59.13",
"dmsSetupTime": "2018-06-15T15:53:59.137",
"rtsSetupTime": "2018-06-15T15:53:59.143",
"inbSetupTime": "2018-06-15T15:53:59.243",
"msgListCreationStatusId": 700,
"msgListCreationStartTime": "2018-06-15T15:54:01.237",
"msgListCreationFinishTime": "2018-06-15T16:07:58.323",
"msgCreatedAmount": 3,
"contCalculationStatusId": 700,
"contCalculationStartTime": "2018-06-15T15:54:53.58",
"contCalculationFinishTime": "2018-06-15T16:07:58.323",
"contCalculatedAmount": 3,
"personalizationStatusId": 700,
"personalizationStartTime": "2018-06-15T15:54:53.58",
"personalizationFinishTime": "2018-06-15T16:07:58.323",
"personalizedAmount": 3,
"sendingStatusId": 700,
"sendingStartTime": "2018-06-15T15:54:53.58",
"sendingFinishTime": "2018-06-15T16:07:58.323",
"sentAmount": 3
}
```