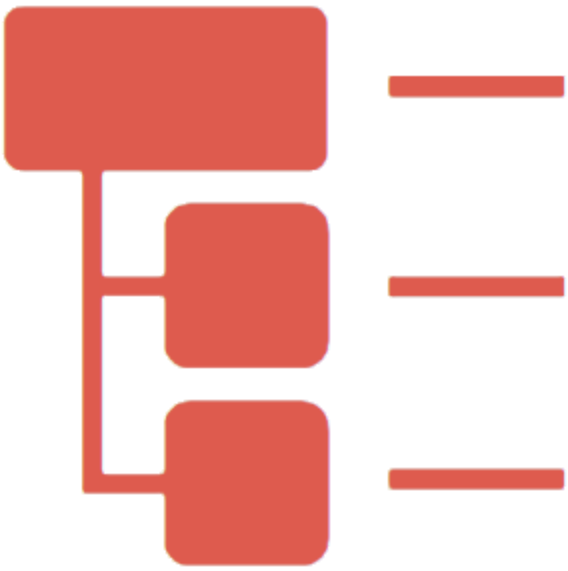
API Technical Guide: Dynamic Block

Cheetah Messaging

Table of Contents

1 Introduction



Purpose

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

Overview

The DYNAMIC BLOCK endpoint is used to create, view, edit, and delete Dynamic Blocks. A Dynamic Block is a special type of asset that allows you to customize the content of a Campaign based on some logical rule, or condition. Instead of having to create separate Campaigns for each content variation, you can create a single Campaign, and use a Dynamic Block to determine the appropriate Content Block (or External Content Block) to send to each recipient. Dynamic Blocks can be used to control many different components of a Campaign, such as text, images, HTML code, an email subject line, or an email friendly-from name.

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

Methods

The DYNAMIC BLOCK endpoint supports the following HTTP methods:

- **POST**: Create a new Dynamic Block.
- **GET**: Retrieve information about a specified Dynamic Block.
- **PUT**: Submit modifications to an existing Dynamic Block.
- **DELETE**: Delete a specified Dynamic Block.

Authentication

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

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

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

2 Create a Dynamic Block

Overview

This section describes how to create a new Dynamic Block via a POST request to the DYNAMIC BLOCK endpoint.

A Dynamic Block consists of one or more logical criteria. These criteria are referred to as "Rule / Block Pairs" because they consist of both a condition to be met (the "Rule") and a resulting action (the Content Block to be used when the condition is met). A Dynamic Block can consist of one or more Rule / Block Pairs to drive specific, personalized content to the correct audience. Dynamic Blocks also support the use of "default" content, which will be sent to Campaign recipients who don't meet any of the defined rules.

Messaging provides two different methods (called "query types") for establishing how you want to identify the "Rule" -- either by inclusion in a Filter, or by a value in a specific field. You must choose one of these two query types for your Dynamic Block; you can't use both types within the same Dynamic Block.

If your Dynamic Block has multiple Rule / Block Pairs, you must identify the logical structure for how the Rule / Block Pairs work together. Messaging supports two logical structures: "IF > ELSE IF" and "IF > IF. "

The IF > ELSE IF structure is useful if you know you want a recipient to be selected by only one rule. The system works from the top down, evaluating candidates against the first rule. Only the non-matching recipients drop down to the second rule.

For example, let's say you have a rewards program with three levels (Gold, Silver, and Bronze), and you want to send a different offer to members in each of these three levels. You could set up your logical rules as follows:

- IF Recipient is a Gold Member, then send Gold offer,

- ELSE IF Recipient is a Silver Member, then send Silver offer,
- ELSE IF Recipient is a Bronze Member, then send Bronze offer,
- ELSE send Default offer.

Each recipient would be evaluated against the first rule (Gold Members). Only recipients who don't match this rule would drop down to the next step, and be evaluated against the second rule (Silver Members). Likewise, only recipients not selected by the second rule would get evaluated by the third rule (Bronze Members). And finally, any recipients not selected by any of the rules would receive the Default offer content. In this manner, each recipient can be selected by only one rule, and receive only one content variation.

With the IF > IF structure, a recipient can potentially match to multiple rules, and therefore could receive multiple content variations. For example, let's say you want to select customers based on their product purchase history, in order to send them one or more coupons. You could set up your logical rules as follows:

- If Recipient purchased shoes, then send shoes coupon,
- IF Recipient purchased a shirt, then send shirt coupon,
- IF Recipient purchased socks, then send socks coupon,
- ELSE send Default coupon.

Every recipient is evaluated against all three rules. If, for example, a recipient purchased shoes and a shirt, then she would match to two of the above rules, and receive both coupons.

Pre-requisites

Before you can create a new Dynamic Block, you typically need to create some supporting assets first, then reference those assets within the DYNAMIC BLOCK POST request. These assets may include the following:

- **Filter** -- You need a Filter if you're creating a "query by Filter" type of Dynamic Block that assigns recipients to a Rule / Block Pair based on inclusion in a Filter. Each

Rule / Block Pair will need its own Filter. You'll need to reference the Filter's **Object Reference ID** within the POST request (see **filter_id** below for details).

- **Field** -- You need a Field on your source table if you're using a "query by value in a field" type of Dynamic Block that assign recipients to a Rule / Block Pair based on a specified value in this Field. You'll need to reference the Field's **Field ID** within the POST request (see **prop_id** below for details).
- **Content Block / External Content Block** -- For each Rule / Block Pair in your Dynamic Block, you need to indicate the content that the recipients selected by this Rule / Block Pair will receive. You'll need to reference the Content Block's (or External Content Block's) **Object Reference ID** within the POST request, along with additional parameters about the asset (see **childContent** below for details).

Parameters

The options and parameters described in this section explain how to create a new Dynamic Block.

cust_id

This integer parameter is required.

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

Example:

```
"cust_id": 394
```

type_id

This string parameter is required.

For a Dynamic Block, the value of this parameter must be "LOGIC_BLOCK."

Example:

```
"type_id": "LOGIC_BLOCK"
```

entity_id

This integer parameter is required.

The **entity_id** parameter represents the **Entity ID** of the Dynamic Block's source table.

Example:

```
"entity_id": 100
```

view_id

This integer parameter is required.

The **view_id** parameter represents the **Object Reference ID** of the Dynamic Block's source table. The **view_id** must refer to the same table as the **entity_id** parameter.

Example:

```
"view_id": 1002
```

contBodies

This object is required in the message, but it's not currently used for anything. Simply include the empty object in the message.

Example:

```
"contBodies": []
```

contModelProps

As described above, Messaging provides two different methods (called "query types") for establishing how you want to identify the "Rule" in a Dynamic Block -- either by inclusion in a Filter, or by values in a specific field. The **contModelProps** object is used if you're querying by values in a field.

> Note
> ımic Block is limited to using only one field, and that field must be in the Dynamic Block's source table (you can't join to another table).

For example:

```
"contModelProps": [
    {
    "prop_id": 1210
    }
]
```

Please note that the **contModelProps** object is required in the message, even if you're using the "by Filter" query type. In this case, simply include the empty object in the message.

For example:

```
"contModelProps": []
```

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

prop_id

This integer parameter is optional.

The **prop_id** represents the **Field ID** of the field you want to use to define the Rule / Block Pairs in your Dynamic Block. Once you identify this field, you can set up Rule / Block Pairs that will select recipients based on specific values in this field (see **case_val** for details on how to define the value).

contParts

The **contParts** object is used to define all of the Rule / Block Pairs. Your Dynamic Block can have one or more Rule / Block Pairs, each of which is contained within the **contParts** object. You can also define the "default" content within the **contParts** object; default content is sent to recipients who aren't selected by any of the Rule / Block Pairs.

Below is an example of a Dynamic Block that uses the query type "by a value in a field." This Dynamic Block as two Rule / Block Pairs, and default content.

```
"contParts": [
        {
            "seq": 1,
            "child_cont_id": 50380,
            "default_flag": 0,
            "else_flag": 1,
            "case_val": "IL",
            "childContent": { ... }  //Content Block details
        },
```

```
            {
                "seq": 2,
                "child_cont_id": 50645,
                "default_flag": 0,
                "else_flag": 1,
                "case_val": "IA",
                "childContent": { ... }  //Content Block details
            },
            {
                "seq": 3,
                "child_cont_id": 50269,
                "default_flag": 1,
                "else_flag": 1,
                "childContent": { ... }  //Content Block details
            }
        ]
```

Below is an example for a Dynamic Block that uses the query type "by a Filter." This Dynamic Block has two Rule / Block Pairs, and default content.

```
    "contParts": [
            {
                "seq": 1,
                "child_cont_id": 52686,
                "filter_id": 30400,
                "default_flag": 0,
                "else_flag": 1,
                "childContent": { ... }   //Content Block details
            },
            {
                "seq": 2,
                "child_cont_id": 50380,
                "filter_id": 30654,
                "default_flag": 0,
                "else_flag": 1,
                "childContent": { ... }   //Content Block details
            },
            {
                "seq": 3,
                "child_cont_id": 59887,
                "default_flag": 1,
                "else_flag": 1,
                "childContent": { ... }   //Content Block details
            }
        ]
```

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

seq

This integer parameter is required.

The **seq** parameter is used to define the sequence for the Rule / Block Pairs in your Dynamic Block. Provide a value of "1" for the first Rule / Block Pair, a value of "2" for the second, and so on.

Even though this parameter is always required, it's really only used for Dynamic Blocks with an IF > ELSE IF logical structure. Once a recipient matches a rule, he or she is removed from any further evaluations. Therefore, the sequence of the rules is important, because even if a recipient qualifies for more than one rule, he or she would receive only the Content Block from the FIRST rule to which he or she matched.

With an IF > IF structure, the sequence of the rules doesn't matter, as the system evaluates every customer against every rule. So, even if a customer met the first rule, he or she would still get evaluated against all subsequent rules as well.

child_cont_id

This integer parameter is optional.

The **child_cont_id** parameter represents the **Object Reference ID** of the Content Block (or External Content Block) for this Rule / Block Pair. Please note that you'll need to provide additional details on this Content Block within the **childContent** object, described below.

filter_id

This integer parameter is optional.

The **filter_id** parameter is used if the query type in your Dynamic Block is "by Filter." This parameter represents the **Object Reference ID** of the desired Filter that you want to use to select the records for this Rule / Block Pair.

default_flag

This integer parameter is optional.

The **default_flag** parameter is used to indicate the "default" content. The default content is sent to a recipient who doesn't meet the criteria for any of the Rule / Block Pairs. The use of default content isn't required, but is usually recommended.

The possible values for this parameter are:

- "1" -- Indicates this is the default content

- "0" -- Indicates this is not the default content

If you don't provide this parameter, the system defaults to a value of "0."

else_flag

This integer parameter is optional.

The **else_flag** parameter is used to indicate which logical structure to use for this Dynamic Block -- "IF > ELSE IF" or "IF > IF."

The possible values for this parameter are:

- "1" -- Use an IF > ELSE IF logical structure
- "0" -- Use an IF > IF logical structure

If you don't provide this parameter, the system defaults to a value of "0."

case_val

This string parameter is optional.

The **case_val** parameter is used if the query type in your Dynamic Block is "by a value in a field." This parameter represents the desired value for the field that you specified (see **contModelProps** for details on how to select the desired field).

childContent

The **childContent** object is used to specify the content that gets sent to a recipient selected by this Rule / Block Pair. This content can be either a Content Block or an External Content Block.

This object must contain all of the information about the referenced Content Block, including the name, source table, and content format versions (HTML, Plain Text, etc.). The simplest way to populate this object is to use the CONTENT BLOCK API endpoint to retrieve all of these necesssary parameters.

For more details on the CONTENT BLOCK endpoint, please see the *Content Block API Technical Guide*, or the Messaging Online Help system.

The recommended method is as follows:

1. Create the Content Block, either through the application's user interface, or through the CONTENT BLOCK API endpoint. Make a note of the Content Block's Object Reference ID.
2. Submit a GET request to the CONTENT BLOCK endpoint, using the above Object Reference ID. The GET response will contain all of the necessary parameters and information about the Content Block.
3. Copy the GET response payload to your clipboard.
4. In the DYNAMIC BLOCK POST payload, paste the contents of your clipboard into the **childContent** object.
5. Within the **childContent** object, make any necessary changes to the Content Block, such as modifying the HTML or Plain Text content, for example. Please note that if you make any changes to the Content Block, then submit the POST request, these changes will be applied to the existing Content Block.

As an example, let's say you're creating a Dynamic Block that shows different content based on the recipient's membership in your rewards program -- Gold, Silver, and Bronze. You also have default content to display to all other recipients who aren't in one of the rewards program levels.

For your Gold-level members, you've created the desired content in a Content Block named "Gold." You need to submit a GET request to the CONTENT BLOCK endpoint, using the Object Reference ID for the "Gold" Content Block. The response message is as follows:


```
{
    "cont_id": 52686,
    "cont_name": "Gold",
    "cust_id": 394,
    "entity_id": 100,
    "type_id": "PARAGRAPH",
    "view_id": 1002,
    "contBodies": [
        {
            "cont_id": 52686,
            "type_id": "TEXT",
            "usage_mask": "EMAIL",
            "body": "Gold level plain text content goes here."
        },
        {
            "cont_id": 52686,
            "type_id": "HTML",
            "usage_mask": "ALL_EMAIL_STYLE_USAGE_MASK",
            "body": "Gold level HTML content goes here."
```

```
            }
        ],
        "contModelProps": [],
        "contParts": [],
        "obj": {
            "obj_id": 45935,
            "display_name": "Gold",
            "type_id": "ContentBlock",
            "ref_id": 52686,
            "parent_obj_id": 37249,
            "eligibility_status_id": "READY"
        }
    }
```

Copy the entire response message to your clipboard.

Next, begin creating your DYNAMIC BLOCK POST request. Within the message payload, you define your first Rule / Block Pair based on a Filter that selects your Gold-level members. Within the **childContent** object, paste the contents of your clipboard (highlighted below in red).

For example:

```
{
    "cust_id": 394,
    "entity_id": 100,
    "type_id": "LOGIC_BLOCK",
    "view_id": 1002,
    "contBodies": [],
    "contModelProps": [],
    "contParts": [
        {
            "seq": 1,
            "child_cont_id": 50380,
            "default_flag": 0,
            "else_flag": 1,
            "filter_id": 30400,
            "childContent": {


"cont_id": 52686,


"cont_name": "Gold",


"cust_id": 394,


"entity_id": 100,


"type_id": "PARAGRAPH",
```

```
"view_id": 1002,

"contBodies": [

                                                                    {

"cont_id": 52686,

"type_id": "TEXT",

"usage_mask": "EMAIL",

                                                                     "body": "Gold
level plain text content goes here."

                                                                   },

                                                                    {

"cont_id": 52686,

"type_id": "HTML",

"usage_mask": "ALL_EMAIL_STYLE_USAGE_MASK",

                                                                     "body":
"Gold level HTML content goes here."

                                                                    }

                                                                  ],

"contModelProps": [],

"contParts": [],

"obj": {

"obj_id": 45935,

"display_name": "Gold",

"type_id": "ContentBlock",

"ref_id": 52686,

"parent_obj_id": 37249,

"eligibility_status_id": "READY"

                                                                    }

          }
        }
     ],
```

```
        "obj": {
            "display_name": "Reward Level Dynamic Block",
            "parent_obj_id": 37249
        }
    }
```

Then, repeat the above steps for the Silver and Bronze levels. For both of these levels, you'll need to create the Filter that selects the desired records, then create a Content Block, and copy / paste the Content Block details into the POST payload (incrementing the **seq** counter for each one).

Lastly, you'll need to create a "default" Content Block and copy / paste it into the POST payload as well. The default content doesn't use a Filter, but you do need to set the **default_flag** parameter to "1."

obj

This object contains the name and location of the new Dynamic Block.

Example:

```
"obj":
  {
  "display_name": "Sample Dynamic Block",
  "parent_obj_id": 37249
  }
```

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

display_name

This string parameter is required.

The **display_name** parameter contains the name of the Dynamic Block. This name must be unique within the selected folder location.

parent_obj_id

This integer parameter is required.

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

3 Edit a Dynamic Block

Overview

This section describes the options for viewing, modifying, and deleting Dynamic Blocks via the DYNAMIC BLOCK endpoint.

Retrieve a Dynamic Block

The GET method is used to retrieve all of the information about a specified Dynamic Block.

When submitting a GET request to the DYNAMIC BLOCK endpoint, the request message must include the Dynamic Block's **Object Reference ID** as a query type parameter within the URL.

For example:

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

Delete a Dynamic Block

The DELETE method is used to delete a specified Dynamic Block.

When submitting a DELETE request to the DYNAMIC BLOCK endpoint, the request message must include the Dynamic Block's **Object Reference ID** as a query type parameter within the URL.

For example:

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

Edit a Dynamic Block

The PUT method allows you to submit modifications to an existing Dynamic Block. Using this method, you can change the Dynamic Block name, modify the Rule / Block Pairs, and changes the Filters, field values, or Content Blocks referenced within the Rule / Block Pairs. The parameters for the PUT method are the same as described in the **Create a Dynamic Block** section.

To remove a Rule / Block Pair from an existing Dynamic Block, simply omit it from the request message; any existing Rule / Block Pairs that aren't referenced in the PUT request message will be removed from the Dynamic Block.

When submitting a PUT request to the DYNAMIC BLOCK endpoint, the request message must include the Dynamic Block's **Object Reference ID** as a query type parameter within the URL.

For example:

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

When constructing a PUT request message, a common practice is to first perform a GET request on the asset, copy/paste the GET response payload into the PUT request, then make whatever revisions are necessary. In this manner, you can simply make modifications to the payload, rather than re-constructing the entire PUT request payload from scratch.

However, care must be taken if using this method on the Dynamic Block endpoint because of the way the platform handles the content in any Content Blocks associated to the Dynamic Block. In the GET response, the platform will not return the content in the associated Content Blocks. This means that within the **childContent** object, the **contBodies** parameter will be empty, even if a Content Block actually has content in it.

The following example shows the **childContent** object from a Dynamic Block GET response. You'll note that **contBodies** is empty, even though this Content Block actually does have content defined.

```
"childContent": {
                                        "cont_id": 50378,
                                "cont_name": "Illinois block",
                                  "cust_id": 394,
```

```
                                                    "entity_id": 100,
                                                "type_id": "PARAGRAPH",
                                                    "view_id": 1002,
                                                    "contBodies": [],
                                                "contModelProps": [],
                                                    "contParts": [],
                                                            "obj": {

    "obj_id": 43953,

    "display_name": "Illinois block",

    "type_id": "ContentBlock",

    "ref_id": 50378,

    "parent_obj_id": 37249,

    "eligibility_status_id": "READY"
                                                                    }
    }
```


If you were to simply copy this GET response into a PUT request payload, and submit it, the platform would delete all of the content from the referenced Content Block (because **contBodies** is blank), which may not be desired.

Therefore, when constructing a PUT request payload, you must use the same methodology as used when constructing a POST request (see **childContent** for details).

4 Response

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

Success

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

A successful response to a GET message to retrieve a Dynamic Block will generate a response code of "200," followed by the details of the specified Dynamic Block contained within the body of the response message.

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

> **Note** In the POST, PUT, and GET response payloads, the platform will not return the content for any of the Content Blocks used in the Dynamic Block. This means that within the **childContent** object, the **contBodies** parameter will be empty, even if a Content Block actually has content in it.

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

Errors

If Messaging encounters a problem with a DYNAMIC BLOCK 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	"An Obj with this name already exists"	A Dynamic Block with this same name already exists within the designated folder; the **display_name** value must be unique within a folder.

5 Sample Messages

This section contains sample request messages for the DYNAMIC BLOCK endpoint.

Sample Request #1

This sample POST request message creates a new Dynamic Block with one Rule / Block Pair based on inclusion in a Filter, and an IF > ELSE IF lol'vegical structure. The payload also includes default content.

JSON Payload

```
{
    "cust_id": 394,
    "entity_id": 100,
    "type_id": "LOGIC_BLOCK",
    "view_id": 1002,
    "contBodies": [],
    "contModelProps": [],
    "contParts": [
        {
            "seq": 1,
            "child_cont_id": 52686,
            "default_flag": 0,
            "else_flag": 1,
            "filter_id": 30400,
            "childContent": {
                "cont_id": 52686,
                "cont_name": "Gold",
                "cust_id": 394,
                "entity_id": 100,
                "type_id": "PARAGRAPH",
                "view_id": 1002,
                "contBodies": [
                    {
```

```
"cont_id": 52686,

"type_id": "TEXT",

"usage_mask": "EMAIL",
                                                                    "body":
"Gold plain text content goes here. "
                                                                    },
                                                                    {

"cont_id": 52686,

"type_id": "HTML",

"usage_mask": "ALL_EMAIL_STYLE_USAGE_MASK",

"body": "Gold HTML content goes here. "
                                                                    }
                                                                    ],

                                                "contModelProps": [],

                                                     "contParts": [],

                                                             "obj": {

                                                     "obj_id": 45935,

                                                "display_name": "Gold",

                                              "type_id": "ContentBlock",

                                                     "ref_id": 52686,

                                                "parent_obj_id": 37249,

                                           "eligibility_status_id": "READY"

                                                                         }
            }
        },
        {
            "seq": 2,
            "child_cont_id": 57940,
            "default_flag": 1,
            "else_flag": 1,
            "childContent": {
                                                          "cont_id": 57940,

                                  "cont_name": "Default Content Block",

                                                            "cust_id": 394,

                                                          "entity_id": 100,
```

```
                                                    "type_id": "PARAGRAPH",
                                                           "view_id": 1002,
                                                            "contBodies": [
                                                                          {
                                                          "cont_id": 57940,
"type_id": "TEXT",
"usage_mask": "EMAIL",
                     "body": "Plain Text default content goes here."
},
                                                                                {
"cont_id": 57940,
"type_id": "HTML",
"usage_mask": "EMAIL, WEB, WEB_SOCIAL",
"body": "HTML default content goes here."
                                                                             }
                                                                           ],
                                                      "contModelProps": [],
                                                           "contParts": [],
                                                                   "obj": {
                                                          "obj_id": 50595,
                                  "display_name": "Default Content Block",
"type_id": "ContentBlock",
                                                          "ref_id": 57940,
                                                    "parent_obj_id": 37249,
                                               "eligibility_status_id": "READY"
                                                                          }
```

```
            }
        }
    ],
    "obj": {
        "display_name": "Gold Level API Dynamic Block",
        "parent_obj_id": 37249
    }
}
```

Sample Request #2

This sample POST request message creates a new Dynamic Block with one Rule / Block Pair based on the "by value in a field" query type, and an IF > IF logical structure. The Rule / Block Pair is looking for a value of "IL" within a State field.

JSON Payload

```
{
    "cust_id": 394,
    "entity_id": 100,
    "type_id": "LOGIC_BLOCK",
    "view_id": 1002,
    "contBodies": [],
    "contModelProps": [
                                                                                {

"prop_id": 1210
                                                                                }
                                                                            ],
    "contParts": [
        {
            "seq": 1,
            "child_cont_id": 50378,
            "default_flag": 0,
            "else_flag": 0,
            "case_val": "IL",
            "childContent": {
                                                        "cont_id": 50378,
                                                "cont_name": "Illinois block",
                                                    "cust_id": 394,
                                                "entity_id": 100,
                                            "type_id": "PARAGRAPH",
                                                "view_id": 1002,
                                                "contBodies": [
```

```
{
"cont_id": 50378,
"type_id": "TEXT",
"usage_mask": "EMAIL",
"body": "IL content block - Plain text"
},
{
"cont_id": 50378,
"type_id": "HTML",
"usage_mask": "ALL_EMAIL_STYLE_USAGE_MASK",
"body": "IL content block - HTML"
}
],
"contModelProps": [],
"contParts": [],
"obj": {
"obj_id": 43953,
"display_name": "Illinois block",
"type_id": "ContentBlock",
"ref_id": 50378,
"parent_obj_id": 37249,
"eligibility_status_id": "READY"
}
}
}
  ],
    "obj": {
        "display_name": "API Dynamic Block",
        "parent_obj_id": 37249
    }
}
```

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.

Entity ID

The Entity ID is a unique, system-generated identifier for every table in your database. This value is not displayed within the application user interface anywhere, so to get the Entity ID for a table, you must retrieve it by means of the TABLE API endpoint.

To retrieve the Entity ID for a table:

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 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 the field details, the response message provides the Entity ID for this table.

Sample Response:

```
{
  "viewId": 1002,
  "entityId": 100,
  "displayName": "create_date",
  "propId": 1030,
  "columnName": "create_date"
}
```

For more details on the TABLE endpoint, please see the Messaging Online Help system or the *Messaging -- Table API Technical Guide*.

Folder ID

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

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

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

2. The response message provides a list of all the folders in your system. Find the desired folder in the response message.
3. As part of the API response message, the system provides the Folder ID, which is referred to as the **obj_id**.

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

Sample Response:

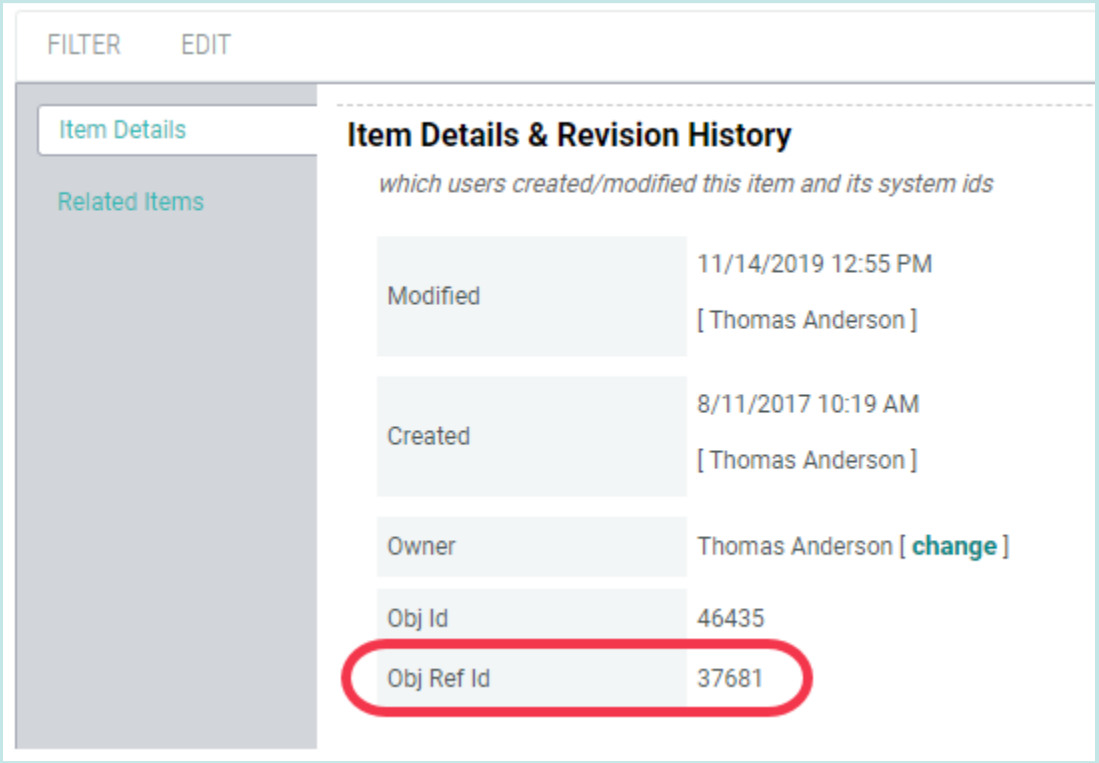
```
{
  "obj_id": 37465,
  "display_name": "Content Block Folder",
  "type_id": "Folder",
  "ref_id": 37465,
  "parent_obj_id": 22817,
  "eligibility_status_id": "READY"
}
```

Object Reference ID

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

For some asset types, the value for this identifier can be found within the Messaging application:

1. From the System Tray, navigate to desired screen for this asset type.
2. In the Tool Ribbon, click the first tab; the name of this tab corresponds to the asset type, such as "Filter" if you're on the Filter screen, for example.
3. The "Item Details" screen is displayed. The Object Reference ID is listed on this screen.



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

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

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

2. The response message provides a list of all the assets in your system that match the search criteria. Find the desired asset in the response message.
3. As part of the API response message, the system provides the Object Reference ID, which is referred to as the **ref_id**. For example:

```
{
  "obj_id": 44737,
  "display_name": "Reward Members Filter",
  "type_id": "Filter",
  "ref_id": 40329,
  "parent_obj_id": 43269,
  "eligibility_status_id": "READY"
}
```


Field ID

The Field ID (or "Property ID") is a unique, system-generated identifier for every field in a table. This value is not displayed within the application user interface anywhere, so to get the Field ID for a field, you must retrieve it by means of the TABLE API endpoint.

To retrieve the Field ID for a field:

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 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 Field ID (referred to as the **propId**).

Sample Response:


```
{
 "viewId": 1002,
 "entityId": 100,
 "displayName": "create_date",
 "propId": 1030,
 "columnName": "create_date"
}
```