



# EclecticIQ Platform API how-tos

Hands-on articles on specific API use cases

Last generated: May 26, 2017



©2017 EclecticIQ

All rights reserved. No part of this document may be reproduced in any form or by any electronic or mechanical means, including information storage and retrieval systems, without written permission from the author, except in the case of a reviewer, who may quote brief passages embodied in critical articles or in a review.

Trademarked names appear throughout this book. Rather than use a trademark symbol with every occurrence of a trademarked name, names are used in an editorial fashion, with no intention of infringement of the respective owner's trademark.

The information in this book is distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author nor the publisher shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this book.

©2017 by EclecticIQ BV. All rights reserved.  
Last generated on May 26, 2017

## Table of contents

Table of contents	2
How to check system health	4
Check system health via the GUI	4
Check system health via the API	7
API endpoint	7
Status request	8
Status response	8
How to make API calls with a script	12
Dependencies	12
How the script works	12
Create the papi.sh file	12
Authentication	13
Make the script executable	14
Create an alias for the script	15
Input parameters	15
Error handling	16
Examples	17
Make HTTP calls with the script	17
Make cURL calls with the script	18
Trigger a dynamic dataset update	19
How to retrieve outgoing feeds through the API	21
Download outgoing feeds manually	21
Download outgoing feeds via the API	22
Authentication	22
Auth request	22
Auth response	23
Public outgoing feed endpoints	24
Download outgoing feeds	25
API request outgoing feeds	25
API response outgoing feeds	25
Download a specific outgoing feed	26
API request specific outgoing feed	27
API response specific outgoing feed	27
Download the latest run	28
API request latest run	29
API response latest run	29
Download the latest content	30
API request latest content	30
API response latest content	31
Download specific content	32
API request specific content	33
API response specific content	33
HTTP status codes	34
Error handling	34
How to report sightings through the API	36
Architecture overview	36
Before you start	37
Create an automation user group	37
Create an automation user role	38
About permissions	38
Create an automation user	39
Get the automation user group ID	40
Get the automation user group ID example	40
Authentication	42

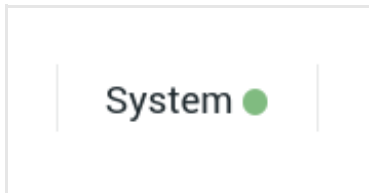
Auth request	42
Auth response	43
Create a sighting	44
Creation request	44
Creation response	44
Creation examples	45
cURL API request — creates new sighting	45
HTTP request message body	45
API response — successful sighting creation	46
Update a sighting	47
Update request	48
Update response	49
Update examples	49
cURL API request — updates existing sighting — update 1/2	49
HTTP request message body	49
API response — successful sighting creation — update 1/2	50
cURL API request — updates existing sighting — update 2/2	51
HTTP request message body	52
API response — successful sighting update — update 2/2	53
Create relationships	53
Relationship request	55
Relationship response	55
Relationship examples	55
cURL API request — creates new version of sighting — relationship 1/3	56
HTTP request message body	56
API response — successful sighting creation — relationship 1/3	56
cURL API request — creates relationship — relationship 2/3	57
HTTP request message body	58
API response — successful relationship creation — relationship 2/3	58
cURL API request — updates sighting — relationship 3/3	59
HTTP request message body	59
API response — successful sighting update — relationship 3/3	60
Error handling	61

# How to check system health

System health gives you a clear basic overview of the overall platform health, as well as the operational status of its components.

## Check system health via the GUI

The first and easiest way to check normal platform operation is by using the GUI system health tool on the status bar:



A green or red dot next to **System** on the status bar indicates the global system health status. Click **System** to drill down to a platform component-level. On the pop-up window click either **Processes** or **Services** to view the status of the normal platform processes or of the platform core services.

System Health ●



PROCESSES

SERVICES

Name	Processes	Status
newrelic_redis_agent	1	●
task	5	●
intel-ingestion	4	●
neo4j-batching	1	●
newrelic_postgresql_agent	1	●
newrelic_elasticsearch_agent	1	●
neo4j-console	1	●
opentaxii	1	●
kibana	1	●
graph-ingestion	1	●
platform-api	1	●
search-ingestion	1	●

## System Health ●



## PROCESSES

## SERVICES

Name	Status
postgresql-9.4	● active
logstash	● active
elasticsearch	● active
redis	● active

Process	Description
graph-ingestion	Data funnel to the Neo4j graph database. Handles data updates for Neo4j
intel-ingestion	Intel ingestion through feeds and enrichers. Consumes incoming data and saves it PostgreSQL, Neo4j, and Elasticsearch. The platform executes one <code>intel-ingestion</code> per processor core. Running tasks are sequentially numbered starting from 0. For example, a platform instance running on a quad core machine normally executes 4 such processes, progressively numbered from <code>intel-ingestion:0</code> to <code>intel-ingestion:3</code>
kibana	Generates dashboard graphs
neo4j-batching	Neo4j graph database batch processing application. It lives on the same server hosting the Neo4j database. It prepares data for ingestion into the Neo4j database
neo4j-console	The platform uses Neo4j via this console. Neo4j is available through the platform as a Linux service
newrelic_elasticsearch_agent	Starts/Restarts Elasticsearch services, including data logging to monitor component health
newrelic_postgresql_agent	Starts/Restarts PostgreSQL services, including data logging
newrelic_redis_agent	Starts/Restarts Redis (message broker) services, including data logging
opentaxii	TAXII server responsible for STIX data transport

Process	Description
platform-api	The web application implementing the platform API and the API endpoints. The endpoints expose services that can be consumed by making API calls and by passing arguments
search-ingestion	Search indexer. Handles Elasticsearch data updates
task	Celery-managed tasks like enrichers, feed integrations, incoming feed data providers, and utilities

Service	Description
elasticsearch	Elasticsearch search and indexing database
postgresql-9.5	PostgreSQL (intel database)
redis	Redis (message broker)
logstash	Log and data aggregation, data pipeline and funneling
nginx	Web server

## Check system health via the API

You can retrieve system health information also by making an API call to the `/status` API endpoint.



First, make an authentication call to receive the bearer token you need to pass with all subsequent API calls.

## API endpoint

API endpoint	<code>/status</code>
Create method	GET
HTTP headers	"Content-Type: application/json", "Accept: application/json", "Authorization: Bearer <token>"



<b>API request</b>	GET + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer <token>" + <platform_host>/status
<b>API response</b>	{ "data" : { <status_reponse> } }

When you make a GET request and obtain a JSON object with entity data in the response, the entity object is wrapped in a data wrapper: { "data" : { ... } }.

## Status request

```
$ curl -X GET
-v
--insecure
-i
-H "Content-Type: application/json"
-H "Accept: application/json"
-H "Authorization: Bearer <token>"
https://platform.host/api/status
```



### Warning:

#### About cURL calls

- If you make HTTPs cURL calls to the API *and* you have a self-signed or an invalid certificate, include the `-k` or the `--insecure` parameter in the cURL call to skip the SSL connection CA certificate check.
- Always append a / trailing slash at the end of an API URL endpoint. The only exception is `/auth`, which does not take a trailing slash.
- In the cURL call, the `-d` data payload with the entity information always needs to be flat JSON, not hierarchical JSON.  
If you want to pass a hierarchical JSON object, include the `--data-binary` parameter, followed by the path to the JSON file, for example `@/path/to/entity_file.json`.

## Status response

```
{
  "data": {
    "health": "GREEN",
```

```
"process_states": [  
  
  {  
    "first_down_at": null,  
    "first_up_at": "2016-06-30T11:28:30+00:00",  
    "health": "GREEN",  
    "n_processes": 1,  
    "n_processes_down": 0,  
    "name": "newrelic_redis_agent"  
  },  
  
  {  
    "first_down_at": null,  
    "first_up_at": "2016-07-06T20:03:27+00:00",  
    "health": "GREEN",  
    "n_processes": 5,  
    "n_processes_down": 0,  
    "name": "task"  
  },  
  
  {  
    "first_down_at": null,  
    "first_up_at": "2016-07-06T20:04:36+00:00",  
    "health": "GREEN",  
    "n_processes": 4,  
    "n_processes_down": 0,  
    "name": "intel-ingestion"  
  },  
  
  {  
    "first_down_at": null,  
    "first_up_at": "2016-07-06T20:05:07+00:00",  
    "health": "GREEN",  
    "n_processes": 1,  
    "n_processes_down": 0,  
    "name": "neo4j-batching"  
  },  
  
  {  
    "first_down_at": null,  
    "first_up_at": "2016-06-30T11:28:35+00:00",  
    "health": "GREEN",  
    "n_processes": 1,  
    "n_processes_down": 0,  
    "name": "newrelic_postgresql_agent"  
  },  
  
  {  
    "first_down_at": null,  
    "first_up_at": "2016-06-30T11:28:35+00:00",  
    "health": "GREEN",  
    "n_processes": 1,  
    "n_processes_down": 0,  
    "name": "newrelic_elasticsearch_agent"  
  },  
]
```

```
{
  "first_down_at": null,
  "first_up_at": "2016-07-06T20:00:14+00:00",
  "health": "GREEN",
  "n_processes": 1,
  "n_processes_down": 0,
  "name": "neo4j-console"
},

{
  "first_down_at": null,
  "first_up_at": "2016-06-30T11:28:35+00:00",
  "health": "GREEN",
  "n_processes": 1,
  "n_processes_down": 0,
  "name": "opentaxii"
},

{
  "first_down_at": null,
  "first_up_at": "2016-07-01T06:54:15+00:00",
  "health": "GREEN",
  "n_processes": 1,
  "n_processes_down": 0,
  "name": "kibana"
},

{
  "first_down_at": null,
  "first_up_at": "2016-07-06T20:04:58+00:00",
  "health": "GREEN",
  "n_processes": 1,
  "n_processes_down": 0,
  "name": "graph-ingestion"
},

{
  "first_down_at": null,
  "first_up_at": "2016-07-06T20:03:21+00:00",
  "health": "GREEN",
  "n_processes": 1,
  "n_processes_down": 0,
  "name": "platform-api"
},

{
  "first_down_at": null,
  "first_up_at": "2016-07-06T20:05:04+00:00",
  "health": "GREEN",
  "n_processes": 1,
  "n_processes_down": 0,
  "name": "search-ingestion"
}
],
```

```
"service_states": [  
  
  {  
    "health": "GREEN",  
    "name": "postgresql-9.5",  
    "state": "active"  
  },  
  
  {  
    "health": "GREEN",  
    "name": "logstash",  
    "state": "active"  
  },  
  
  {  
    "health": "GREEN",  
    "name": "elasticsearch",  
    "state": "active"  
  },  
  
  {  
    "health": "GREEN",  
    "name": "redis",  
    "state": "active"  
  }  
  
]  
}
```

# How to make API calls with a script

Make calls to the EclecticIQ API using our simple 'papi' script.

## Dependencies

The script we are going to use in this example relies on the following dependencies:

- **HTTPie** (<http://httpie.org/>)
- **jq** (<https://stedolan.github.io/jq/>)

The default values for user name and password in the script are:

- `USERNAME=test`
- `PASSWORD=test`

## How the script works

You can use the `papi.sh` script to make calls to the EclecticIQ Platform API from the command line. The script uses *HTTPie* and *jq* to perform the following actions:

- It communicates with the `auth` API endpoint to authenticate and obtain a bearer token.
- It makes calls to the platform API and it includes the token along with the other call parameters.

## Create the papi.sh file

Create or modify the `papi.sh` script so that it looks like the following example:

```
#!/bin/bash

#
# Helper for interacting with the platform API from the command line. This tool
# is a wrapper around httpie. It will take care of authentication and some
# other things. It takes at least three arguments:
#
# - host name
# - http method
# - relative url (without the /api/ part)
#
# Any additional arguments are propagated to httpie.
#
# Examples:
#
#   platform-api-http https://some.host/ GET /users/
#
#   platform-api-http localhost:8000 POST /some/endpoint/ @request-stored-in-a-
#   file.json

set -e

readonly HTTPIE=http
readonly HTTPIE_ARGS="--check-status --verify=no"
readonly USERNAME=test
readonly PASSWORD=test

usage() {
    echo "Usage: ${basename $0} host method path [http-args]" > /dev/stderr
    exit 1
}

main () {
    local HOST="$1"
    local METHOD="$2"
    local API_PATH="$3"
    shift 3 || usage
    local TOKEN=$( ${HTTPIE} ${HTTPIE_ARGS} POST "${HOST}/api/auth"
username=${USERNAME} password=${PASSWORD} | jq --raw-output '.token')
    local URL="${HOST}/api${API_PATH}"
    ${HTTPIE} ${HTTPIE_ARGS} ${METHOD} ${URL} Authorization: '"Bearer ${TOKEN}"' "$@"
}

main "$@"
```

## Authentication

When you make a call to the platform API with `papi.sh`, the script takes care of the following tasks:

- It carries out the authentication step by tokenizing the user's credentials (user name and password).
- When you make a call using the script, it sends the bearer token in the `Authorization` HTTP header:  
`Authorization: Bearer <token>`.

The authentication mechanism is based on **JSON web tokens** (<http://jwt.io/>).

By default, the token expires 2 hours after successfully signing in. The corresponding session is terminated, and you need to sign back in to the platform.

When human interaction is detected — for example, keystrokes or mouse activity — the token is automatically refreshed every 2 minutes. This prevents the system from signing out users who may be working or saving data at that time.

Therefore, the default maximum amount of idle time without any human interaction before being automatically signed out equals to *session token validity - 2 minutes*.

To authenticate and access the platform, do the following:

- Make a `POST` call.
- In the call, pass your authentication credentials as a JSON object to the `/auth` endpoint. The credential data is used to generate a token that is returned with the response.

You need to include the generated bearer token in the `Authorization` HTTP header with each subsequent API call.

The `Authorization` HTTP header has the following format: `Authorization: Bearer <token>`

The following examples show the isolated code snippets in the script that take care of authorization credentials, and passing the token with the `Authorization` HTTP header, respectively.

```
local TOKEN=$( ${HTTPIE} ${HTTPIE_ARGS} POST "${HOST}/api/auth" username=${USERNAME}  
password=${PASSWORD} | jq --raw-output '.token' )
```

```
local URL="${HOST}/api${API_PATH}"  
${HTTPIE} ${HTTPIE_ARGS} ${METHOD} ${URL} Authorization: '"Bearer ${TOKEN}"' "$@"
```

## Make the script executable

To make the script executable, run the following command(s):

```
$ chmod +x ~/papi.sh
```

Examples:

```
$ ./papi.sh https://<platform_host>/ get /enricher-tasks/
```

```
$ ./papi.sh https://<platform_host>/ get /configurations/taxii.settings
```

```
$ ./papi.sh https://<platform_host>/ post /utility-tasks/12/run
```

## Create an alias for the script

You can create an alias in `~/.bash_profile` to make it easier to call the script; run the following command(s):

```
alias papi="~/papi.sh"
```

Examples:

```
$ papi https://<platform_host>/ get /enricher-tasks/
```

```
$ papi https://<platform_host>/ get "/entities/?limit=1&sort=-created_at"
```

```
$ papi https://<platform_host>/ get "/entities/?limit=1&sort=-created_at"
```

## Input parameters

To use the script to make calls to the platform API, do the following:

- In the terminal or in the command line, call the script by typing its name or the corresponding alias, so either `./papi.sh` or `papi`.
- Set the target host name you want to reach, for example `https://platform.host`.
- Pass a valid **HTTP method** (<http://www.restapitutorial.com/lessons/httpmethods.html>) like `get` or `post`.
- Pass the URL corresponding to the API endpoint whose service you want to consume, for example `/outgoing-feeds/`.
- *Optional* — Pass any valid URL query parameters or a `.json` file containing any additional request parameters.



Parameter	Description
<code>https://&lt;platform_host&gt;/</code>	<i>Required</i> — The name of the host used to reach the API endpoint and to communicate with the API service.
<code>POST, GET, PUT, DELETE</code>	<i>Required</i> — A valid <b>HTTP method</b> ( <a href="http://www.restapitutorial.com/lessons/httpmethods.html">http://www.restapitutorial.com/lessons/httpmethods.html</a> ) to create, read, update, or delete a resource.
<code>/&lt;API_endpoint&gt;/</code>	<i>Required</i> — A relative URL pointing to the API endpoint that exposes the service you want to consume.
<code>?url=true&amp;query=search-or-filter&amp;params=4</code>	<i>Optional</i> — URL query parameters to send any additional search parameters and/or to filter the results returned in the response.



Besides appending URL query parameters, you can also send your request parameters as a JSON file.

Example:

```
$ papi https://platform.host/ get /entities/ @request-parameters.json
```

## Error handling

When an error occurs, the API handles it by returning a **4xx (input/client error) or 5xx (server issue) HTTP response code** (<https://httpstatuses.com/>), and by including a short explanatory message in the JSON response body.

The key/value pairs in the `errors` JSON object provide more details about the possible cause of the error, which should help you solve it.

- `status`: the HTTP response code.
- `title`: the name of the error.
- `detail`: a short explanation of the issue with more context, when available.

```
{
  "errors" : [
    {
      "detail" : "The requested URL was not found on the server. If you entered the
URL manually please check your spelling and try again.",
      "status" : 404,
      "title"  : "Not Found"
    }
  ]
}
```

## Examples



All parameter values in the example(s) are dummy values.  
Replace these sample values with actual, valid ones before testing them in your environment.

## Make HTTP calls with the script

Create or modify the *papi.sh* script so that it looks like the following example:

```
#!/bin/bash
set -e

readonly HTTPIE=http
readonly HTTPIE_ARGS="--check-status --verify=no"
readonly USERNAME=test
readonly PASSWORD=test

usage() {
    echo "Usage: $(basename $0) host method path [http-args]" > /dev/stderr
    exit 1
}

main () {
    local HOST="$1"
    local METHOD="$2"
    local API_PATH="$3"
    shift 3 || usage
    local TOKEN=$( ${HTTPIE} ${HTTPIE_ARGS} POST "${HOST}/api/auth"
username=${USERNAME} password=${PASSWORD} | jq --raw-output '.token')
    local URL="${HOST}/api${API_PATH}"
    ${HTTPIE} ${HTTPIE_ARGS} ${METHOD} ${URL} Authorization: '"Bearer ${TOKEN}"' "$@"
}

main "$@"
```

## Make cURL calls with the script

Create or modify the *papi.sh* script so that it looks like the following example:

```
#!/bin/bash
set -e

readonly HTTPIE=curl
readonly HTTPIE_ARGS="-X"
readonly USERNAME=test
readonly PASSWORD=test

usage() {
    echo "Usage: $(basename $0) host method path [http-args]" > /dev/stderr
    exit 1
}

main () {
    local HOST="$1"
    local METHOD="$2"
    local API_PATH="$3"
    shift 3 || usage
    local TOKEN=$( ${HTTPIE} ${HTTPIE_ARGS} POST "${HOST}/api/auth" -d "{\"username\": \"${USERNAME}\", \"password\": \"${PASSWORD}\"}" -ks | jq '.token' --raw-output)
    local URL="${HOST}/api${API_PATH}"
    ${HTTPIE} ${HTTPIE_ARGS} ${METHOD} ${URL} Authorization: "Bearer ${TOKEN}" "$@"
}

main "$@"
```

## Trigger a dynamic dataset update

This example shows how to trigger a dynamic dataset update task.

- Edit the `papi.sh` script to specify the correct IP address/host you want to reach.
- To retrieve the dataset ID, sign in to the platform and do the following:
  - On the left-hand navigation sidebar, click **Datasets**.
  - In the web browser address bar, inspect the URL: the numeric value in the URL immediately after `/intel-sets/` is the dataset ID.  
For example if the URL reads `https://platform.host/api/intel-sets/29`, the dataset ID is 29.
- Make a `GET` API call to retrieve the dataset corresponding to the ID you retrieved in the previous step:
 

```
$ papi https://platform.host get /intel-sets/<set_id>
```
- Make a `GET` API call to retrieve a list of all utility tasks.  
You need this information to identify the task handling dynamic sets:
 

```
$ papi https://platform.host/ get /utility-tasks/
```
- Use the appropriate task ID — the ID you retrieved in the previous step — to make a `POST` API call that triggers a dynamic set update task:
 

```
$ papi https://platform.host/ post /utility-tasks/<task_id>/run
```

- Make a GET API call to validate the task run you triggered in the previous step:

```
$ papi https://platform.host/ get /task-runs/<task_id>
```

You can find the <task\_id> value you need in the response to the previous <task-id> request.

- Verify that the dataset was correctly updated by making a new GET API call and by passing the dataset ID with it:

```
$ papi https://platform.host/ get /intel-sets/<set_id>
```

```
# Get a specific dataset ID
```

```
$ papi https://platform.host/ get /intel-sets/<set_id>
```

```
# Get the utility task list to look for the ID of the specific task you want to run
```

```
$ papi https://platform.host/ get /utility-tasks/
```

```
# In the response, find the ID of the task that updates dynamic sets
```

```
# Ex.: find the ID of the 'utilities.intel_set_update' tasks
```

```
# Use this ID to trigger and run a specific task
```

```
$ papi https://platform.host/ post /utility-tasks/<task_id>/run
```

```
# The response returns the task run ID
```

```
# Get a specific task run to check its status
```

```
$ papi https://platform.host/ get /task-runs/<task_id>
```

```
# Get the output feed content blocks
```

```
$ papi https://platform.host/ get /outgoing-feeds/<outgoing_feed_id>/content-blocks/
```

```
# Get the output feed content block ID
```

```
$ papi https://platform.host/ get /content-blocks/<content_block_id>
```

# How to retrieve outgoing feeds through the API

Fetch outgoing feeds either manually through the platform GUI or programmatically via the API.

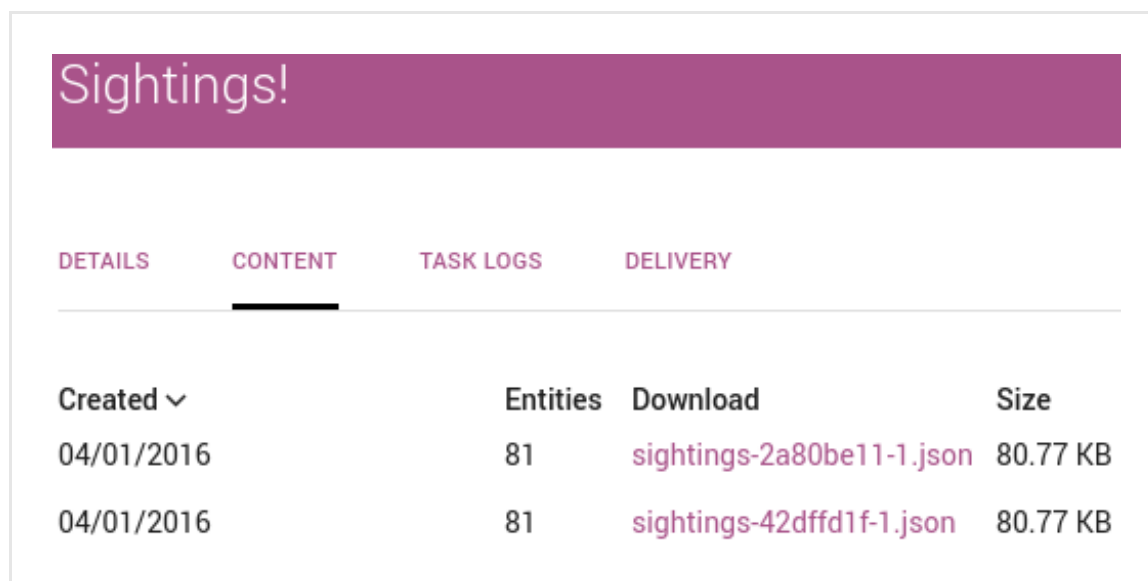
You can retrieve outgoing feed data in two ways:

- Manually, via the platform GUI
- Programmatically, via API calls to the outgoing feed endpoint URLs.

## Download outgoing feeds manually

To manually download outgoing feed data through the platform web browser-based GUI, do the following:

- On the left-hand navigation sidebar, click **Outgoing feeds**.
- The **Outgoing feeds** page displays an overview of the configured output data points, i.e. the publishing channels the platform uses to distribute intel data.  
You can sort the table view by column. To do so, click the column header you want to base the data sorting on. An upward-pointing ▲ or a downward-pointing ▼ arrow in the header indicates ascending and descending sort order, respectively.
- Click the row corresponding to the outgoing feed whose data you want to manually download.
- On the outgoing feed detail pane, select the **Content** tab.



Sightings!			
DETAILS	CONTENT	TASK LOGS	DELIVERY
Created ▼	Entities	Download	Size
04/01/2016	81	<a href="#">sightings-2a80be11-1.json</a>	80.77 KB
04/01/2016	81	<a href="#">sightings-42dffd1f-1.json</a>	80.77 KB

The **Content** tab displays an overview of all the outgoing feed execution runs since the creation of the feed. Successful executions generate and distribute content.

- To download the data for a specific run, click the file link under **Download**, and then save it to a target location.

- The file format depends on the content type for the feed data that was defined during the outgoing feed creation and configuration.

## Download outgoing feeds via the API

You can download outgoing feed data programmatically by making calls to the platform API using the HTTP protocol.

To implement this option:

- The outgoing feed configured **Transport type** needs to be set to **HTTP download**.
- Before making the first API call to fetch the outgoing feed data, you need to authenticate to receive a bearer token.
- You need to pass a valid bearer token with each API call.

## Authentication

The authentication mechanism is based on **JSON web tokens** (<http://jwt.io/>).

By default, the token expires 2 hours after successfully signing in. The corresponding session is terminated, and you need to sign back in to the platform.

When human interaction is detected — for example, keystrokes or mouse activity — the token is automatically refreshed every 2 minutes. This prevents the system from signing out users who may be working or saving data at that time.

Therefore, the default maximum amount of idle time without any human interaction before being automatically signed out equals to *session token validity - 2 minutes*.

To authenticate and access the platform, do the following:

- Make a `POST` call.
- In the call, pass your authentication credentials as a JSON object to the `/auth` endpoint. The credential data is used to generate a token that is returned with the response.

You need to include the generated bearer token in the `Authorization` HTTP header with each subsequent API call.

The `Authorization` HTTP header has the following format: `Authorization: Bearer <token>`

## Auth request

API endpoint	/auth
--------------	-------

<b>Auth method</b>	POST
<b>HTTP headers</b>	"Content-Type: application/json", "Accept: application/json"
<b>API request</b>	POST + "Content-Type: application/json" + "Accept: application/json" + { "username": "<valid_user_name>", "password": "<valid_password>" } + <platform_host>/auth
<b>API response</b>	{ "expires_at": "<expiry timestamp>", "token": "<token>" }

The following example uses cURL to authenticate:

```
$ curl -X POST
-H "Content-Type: application/json"
-d '{ "username" : "<valid_user_name>", "password" : "<valid_password>" }'
https://platform.host/api/auth
```

## Auth response

When the user name and password credential are valid, the `POST` call returns a JSON web token:

```
{
  "expires_at": "2016-03-30T12:11:40.078219+00:00",
  "token"      :
  "abHpYXQiOjE0NTkzMzI3MDAsIm4TcCI6MTQ1OTMzMzOTkwMCwiYWxnIjoisSFMyNTYifQ.oyY1c2VyX2lkIjo1fQ
.LQQ3NdUHp4s-QCXsxd3feI0Dy6tf5XQX9DOML1RNIzQ"
}
```

You need to include the bearer token value in each subsequent API call. You pass the token by including an `Authorization` HTTP header in the API request.

The `Authorization` HTTP header has the following format: `Authorization: Bearer <token>`

In the following example, you make a `GET` request to the `/api/` endpoint to retrieve a list of the available API endpoints and the corresponding methods:

```
$ curl -X GET
-v
--insecure
-i
-H "Content-Type: application/json"
-H "Accept: application/json"
-H "Authorization: Bearer <token>"
https://platform.host/api/
```



**Warning:****About cURL calls**

- If you make HTTPs cURL calls to the API *and* you have a self-signed or an invalid certificate, include the `-k` or the `--insecure` parameter in the cURL call to skip the SSL connection CA certificate check.
- Always append a `/` trailing slash at the end of an API URL endpoint. The only exception is `/auth`, which does not take a trailing slash.
- In the cURL call, the `-d` data payload with the entity information always needs to be flat JSON, not hierarchical JSON.  
If you want to pass a hierarchical JSON object, include the `--data-binary` parameter, followed by the path to the JSON file, for example `@/path/to/entity_file.json`.

## Public outgoing feed endpoints

The endpoints in this section expose only open/public HTTP download outgoing feeds, i.e. the outgoing feeds that are flagged as public upon creation.

To obtain a list of the private HTTP download outgoing feeds, use the `/api/outgoing-feed-download/` endpoint. This endpoint returns only private outgoing feeds, i.e. the outgoing feeds that were not flagged as public when they were created. The outgoing feeds returned with the response require authorization to access their content.

- `/api/open-outgoing-feed-download/`  
Returns all public outgoing feeds with HTTP transport type.
- `/api/open-outgoing-feed-download/<feed-id>/`  
Returns a specific outgoing feed, including all successful feed executions.
- `/api/open-outgoing-feed-download/<feed-id>/runs/latest`  
Returns the latest/most recent successful feed execution with the corresponding published content blocks for a specific outgoing feed.
- `/api/open-outgoing-feed-download/<feed-id>/runs/<run-id>/content-blocks/latest`  
Returns the latest content blocks of a specific successful feed execution for a specific outgoing feed.
- `/api/open-outgoing-feed-download/<feed-id>/runs/<run-id>/content-blocks/<content-block-id>`  
Returns a specific content block of a specific successful feed execution for a specific outgoing feed.



The API request examples in the following sections use cURL and the *papi.sh* script available with the platform.

## Download outgoing feeds

*Download a list of all available public outgoing feeds*

This call returns a JSON object with an array listing all available public outgoing feeds with HTTP transport type.

<b>API endpoint</b>	/open-outgoing-feed-download/
<b>API method</b>	GET
<b>HTTP headers</b>	"Content-Type: application/json", "Accept: application/json", "Authorization: Bearer <token>"
<b>API request</b>	GET + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer <token>" + <platform_host>/open-outgoing-feed-download/
<b>API response</b>	{ "data" : [ <open_outgoing_feed_array> ] }

## API request outgoing feeds

### cURL call

```
$ curl -X GET
      -v
      --insecure
      -i
      -H "Content-Type: application/json"
      -H "Accept: application/json"
      -H "Authorization: Bearer <token>"
      https://platform.host/api/open-outgoing-feed-download/
```

### papi script call

```
$ papi https://<platform_host>/ get /open-outgoing-feed-download/
```

## API response outgoing feeds

```
{
  "data": [
    {
      "id": 1,
      "link": "/api/open-outgoing-feed-download/1",
      "name": "Default outgoing feed"
    },
    {
      "id": 16,
      "link": "/api/open-outgoing-feed-download/18",
      "name": "Public feed with electrolytes"
    },
    {
      "id": 25,
      "link": "/api/open-outgoing-feed-download/25",
      "name": "XYZ"
    }
  ]
}
```

## Download a specific outgoing feed

### *Download the details of a specific outgoing feed*

This call returns a JSON object containing the details of a specific public outgoing feed with HTTP transport type.

To select the public outgoing feed whose details you want to retrieve, include the feed ID in the API request endpoint.

<b>API endpoint</b>	/open-outgoing-feed-download/<feed-id>/
<b>API method</b>	GET
<b>HTTP headers</b>	"Content-Type: application/json", "Accept: application/json", "Authorization: Bearer <token>"
<b>API request</b>	GET + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer <token>" + <platform_host>/open-outgoing- feed-download/<feed-id>/
<b>API response</b>	{ "data" : { <specific_feed_details> } }

## API request specific outgoing feed

### cURL call

```
$ curl -X GET
      -v
      --insecure
      -i
      -H "Content-Type: application/json"
      -H "Accept: application/json"
      -H "Authorization: Bearer <token>"
      https://platform.host/api/open-outgoing-feed-download/18
```

### papi script call

```
$ papi https://<platform_host>/ get /open-outgoing-feed-download/18
```

## API response specific outgoing feed

The response details include an array listing the successful feed executions.

The paths in the `content_blocks` array have the following format:

```
/api/open-outgoing-feed-download/<feed-id>/runs/<run-id>/content-blocks/<content-block-id>
```

- A *run* is a feed execution to publish the feed content.
- A *content block* is a data blob whose format depends on the content type defined for the feed, for example JSON, CSV or STIX.

```
{
  "data": {
    "content_blocks": [
      "/api/open-outgoing-feed-download/18/runs/0ad2edd4-8a7b-4894-b8b3-ae90a22ebaa/content-blocks/32",
      "/api/open-outgoing-feed-download/18/runs/5fdeff71-93af-43a5-b94e-c4ab857a749c/content-blocks/33",
      "/api/open-outgoing-feed-download/18/runs/40e31ada-06e6-4647-a287-4c9b54841619/content-blocks/34",
      "/api/open-outgoing-feed-download/18/runs/0f56ec9c-cc1e-4aae-afd0-f693f412ad55/content-blocks/35",
      "/api/open-outgoing-feed-download/18/runs/d842dd68-8ecf-4ecf-b073-a591d361cf26/content-blocks/36",
      "/api/open-outgoing-feed-download/18/runs/eed28e1e-4352-42a5-8b1f-cfc918b0e0ab/content-blocks/37",
      "/api/open-outgoing-feed-download/18/runs/f830aa7b-4ddc-4725-b13c-7cbe445f306d/content-blocks/40",
      "/api/open-outgoing-feed-download/18/runs/a11bb585-720a-4c56-b650-90cb9d6a69e5/content-blocks/41",
      "/api/open-outgoing-feed-download/18/runs/6e677f4b-c91d-49dd-9c39-70266987b863/content-blocks/42"
    ],
    "id": 18,
    "name": "Public feed with electrolytes"
  }
}
```

## Download the latest run

*Download the latest successful run of a specific outgoing feed*

This call returns a JSON object containing the details of the latest execution (run) of a specific public outgoing feed with HTTP transport type.

To select the public outgoing feed whose details you want to retrieve, include the feed ID in the API request endpoint.

<b>API endpoint</b>	/open-outgoing-feed-download/<feed-id>/runs/latest
<b>API method</b>	GET
<b>HTTP headers</b>	"Content-Type: application/json", "Accept: application/json", "Authorization: Bearer <token>"
<b>API request</b>	GET + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer <token>" + <platform_host>/open-outgoing-feed-download/<feed-id>/runs/latest

**API response**

```
{ "data" : { <specific_feed_latest_run_details> } }
```

## API request latest run

### cURL call

```
$ curl -X GET
      -v
      --insecure
      -i
      -H "Content-Type: application/json"
      -H "Accept: application/json"
      -H "Authorization: Bearer <token>"
      https://platform.host/api/open-outgoing-feed-download/18/runs/latest
```

### papi script call

```
$ papi https://<platform_host>/ get /open-outgoing-feed-download/18/runs/latest
```

## API response latest run

The response details include an array with the content blocks belonging to the most recent successful execution of the specified feed.

The paths in the `content_blocks` array have the following format:

`/api/open-outgoing-feed-download/<feed-id>/runs/<run-id>/content-blocks/<content-block-id>`

- A *run* is a feed execution to publish the feed content.
- A *content block* is a data blob whose format depends on the content type defined for the feed, for example JSON, CSV or STIX.

```
{
  "data": {
    "content_blocks": [
      "/api/open-outgoing-feed-download/18/runs/6e677f4b-c91d-49dd-9c39-70266987b863/content-blocks/42"
    ],
    "id": 18,
    "name": "Public feed with electrolytes"
  }
}
```

## Download the latest content

*Download the latest content from a specific run of a specific outgoing feed*

This call returns the details of the latest content block belonging to a specific execution (run) of a specific public outgoing feed with HTTP transport type. To select the public outgoing feed and the execution whose details you want to retrieve, include the feed ID and the execution (run) ID in the API request endpoint.

<b>API endpoint</b>	/open-outgoing-feed-download/<feed-id>/runs/<run-id>/content-blocks/latest
<b>Create method</b>	GET
<b>HTTP headers</b>	"Content-Type: application/json", "Accept: application/json", "Authorization: Bearer <token>"
<b>API request</b>	GET + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer <token>" + <platform_host>/open-outgoing-feed-download/<feed-id>/runs/<run-id>/content-blocks/latest
<b>API response</b>	<content_block_blob>

## API request latest content

cURL call

```
$ curl -X GET
      -v
      --insecure
      -i
      -H "Content-Type: application/json"
      -H "Authorization: Bearer <token>"
      https://platform.host/api/open-outgoing-feed-download/18/runs/6e677f4b-c91d-
      49dd-9c39-70266987b863/content-blocks/latest
```

### papi script call

```
$ papi https://<platform_host>/ get /open-outgoing-feed-download/18/runs/6e677f4b-
c91d-49dd-9c39-70266987b863/content-blocks/latest
```

## API response latest content

The response returns the latest content block that was generated and distributed with the specific execution of a specific public outgoing feed, as specified in the API request endpoint.

The paths in the `content_blocks` array have the following format:

`/api/open-outgoing-feed-download/<feed-id>/runs/<run-id>/content-blocks/<content-block-id>`

- A *run* is a feed execution to publish the feed content.
- A *content block* is a data blob whose format depends on the content type defined for the feed, for example JSON, CSV or STIX.



```

<stix:STIX_Package xmlns:cybox="http://cybox.mitre.org/cybox-2"
xmlns:indicator="http://stix.mitre.org/Indicator-2"
xmlns:stix="http://stix.mitre.org/stix-1" xmlns:id-1="http://hailataxii.com"
xmlns:DomainNameObj="http://cybox.mitre.org/objects#DomainNameObject-1"
xmlns:stixCommon="http://stix.mitre.org/common-1" xmlns:not-yet-
configured="http://not-yet-configured.example.org/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="not-yet-configured:package-
39de53f4-cbe4-4755-8274-1f6f825639cb" timestamp="2016-01-29T10:42:19+00:00"
version="1.2">
  <stix:STIX_Header>
    <stix:Title>Entities package</stix:Title>
  </stix:STIX_Header>
  <stix:Indicators>
    <stix:Indicator id="not-yet-configured:indicator-153e9285-c99f-412e-9f88-
1f883765d897" xsi:type="indicator:IndicatorType">
      <indicator:Observable id="id-1:Observable-9f042056-7ba1-433a-b67f-
467982412fe0" sighting_count="1">
        <cybox:Title>Domain:Edited</cybox:Title>
        <cybox:Description>Domain: xn----8sbafbb5baamcbp7aadbilbi6e2bygua.xn--plai |
isFQDN: True | </cybox:Description>
        <cybox:Object id="id-1:DomainName-3cfb1c45-debd-421c-92de-5aa5b7a447bc">
          <cybox:Properties xsi:type="DomainNameObj:DomainNameObjectType">
            <DomainNameObj:Value condition="Equals">xn----
8sbafbb5baamcbp7aadbilbi6e2bygua.xn--plai</DomainNameObj:Value>
          </cybox:Properties>
        </cybox:Object>
      </indicator:Observable>
      <indicator:Kill_Chain_Phases>
        <stixComsuccemon:Kill_Chain_Phase phase_id="stix:TTP-786ca8f9-2d9a-4213-
b38e-399af4a2e5d6"/>
      </indicator:Kill_Chain_Phases>
    </stix:Indicator>
  </stix:Indicators>
</stix:STIX_Package>

```

## Download specific content

*Download specific content from a specific run of a specific outgoing feed*

This call returns the details of the specific content belonging to the specified run of a specific public outgoing feed with HTTP transport type. You select the feed, the run, and the content block you want the call to return by adding the feed ID, the run ID, and the content block ID to the API endpoint of the request.

<b>API endpoint</b>	/open-outgoing-feed-download/<feed-id>/runs/<run-id>/content-blocks/<content-block-id>
<b>Create method</b>	GET

HTTP headers	"Content-Type: application/json", "Accept: application/json", "Authorization: Bearer <token>"
API request	GET + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer <token>" + <platform_host>/open-outgoing- feed-download/<feed-id>/runs/<run-id>/content-blocks/<content- block-id>
API response	<content_block_blob>

## API request specific content

### cURL call

```
$ curl -X GET
      -v
      --insecure
      -i
      -H "Content-Type: application/json"
      -H "Authorization: Bearer <token>"
      https://platform.host/api/open-outgoing-feed-download/33/runs/2fbcc01f-6553-
      4a92-ac3d-456049a198f7/content-blocks/84
```

### papi script call

```
$ papi https://<platform_host>/ get /open-outgoing-feed-download/33/runs/2fbcc01f-
6553-4a92-ac3d-456049a198f7/content-blocks/84
```

## API response specific content

The response returns the specific content block that was generated and distributed with the specific execution of a specific public outgoing feed, as per corresponding feed, run, and content block IDs defined in the API request endpoint.

The paths in the `content_blocks` array have the following format:

`/api/open-outgoing-feed-download/<feed-id>/runs/<run-id>/content-blocks/<content-block-id>`

- A *run* is a feed execution to publish the feed content.
- A *content block* is a data blob whose format depends on the content type defined for the feed, for example JSON, CSV or STIX.

```

CEF:0|EclecticIQ|EclecticIQ Platform|0.15.0|ttp|EclecticIQ Platform \
|ttp|verylow|ad.incomingFeedSourceId=e8815882-d610-47d4-991c-e0cde68445e8
deviceReceiptTime=1453662154657 cs1Label=title cs1=Targeting: TD Ameritrade cat=ttp
cs3=name cn2=0 start=1441379114127 cs4=td ameritrade cs3Label=extractType cs2=WHITE
modelConfidence=0 cn1Label=halfLifeDays priority=2 severity=verylow
cs4Label=extractValue externalId={http://www.hailataxii.com}ttp-6730e2ef-9e9a-4eeb-
a6f4-d741f1a6cb4a relevancy=4 ad.incomingFeedName=guest.phishtank_com cn1=182
cs2Label=tlpColor cn2Label=sightingsCount attackerUserName=td ameritrade
CEF:0|EclecticIQ|EclecticIQ Platform|0.15.0|ttp|EclecticIQ Platform \
|ttp|veryhigh|ad.incomingFeedSourceId=bd74b8c9-b0cc-4fa4-8de5-6bba538fad63
deviceReceiptTime=1455630435950 cs1Label=title cs1=testt2 cat=ttp cn2=1
start=1455630435950 cs4=cirque du soleil cs3Label=extractType cs3=name
modelConfidence=9 cn1Label=halfLifeDays priority=10 severity=veryhigh
cs4Label=extractValue relevancy=8 msg=
<p>asdf</p> ad.incomingFeedName=Testing Group cn1=182 cs2Label=tlpColor
cn2Label=sightingsCount attackerUserName=cirque du soleil

```

## HTTP status codes

This is how you can read HTTP status codes for HTTP download outgoing feeds retrieved through the API:

HTTP status code	Description
<b>404</b>	<i>Not found.</i> Error. The request did not execute. Check the error message or the console/terminal for more details.
<b>200</b>	<i>OK.</i> This status code is returned when a request is executed correctly, and an empty list is returned. It can mean that no HTTP download outgoing feeds are configured, or they may not be working correctly, or they have no content blocks to return yet.
<b>200</b>	<i>OK.</i> This status code is returned when a request is executed correctly, and a populated list with content blocks is returned. Make a happy dance and have a beer.

## Error handling

When an error occurs, the API handles it by returning a **4xx (input/client error)** or **5xx (server issue)** HTTP **response code** (<https://httpstatuses.com/>), and by including a short explanatory message in the JSON response body.

The key/value pairs in the `errors` JSON object provide more details about the possible cause of the error, which should help you solve it.

- **status**: the HTTP response code.
- **title**: the name of the error.
- **detail**: a short explanation of the issue with more context, when available.

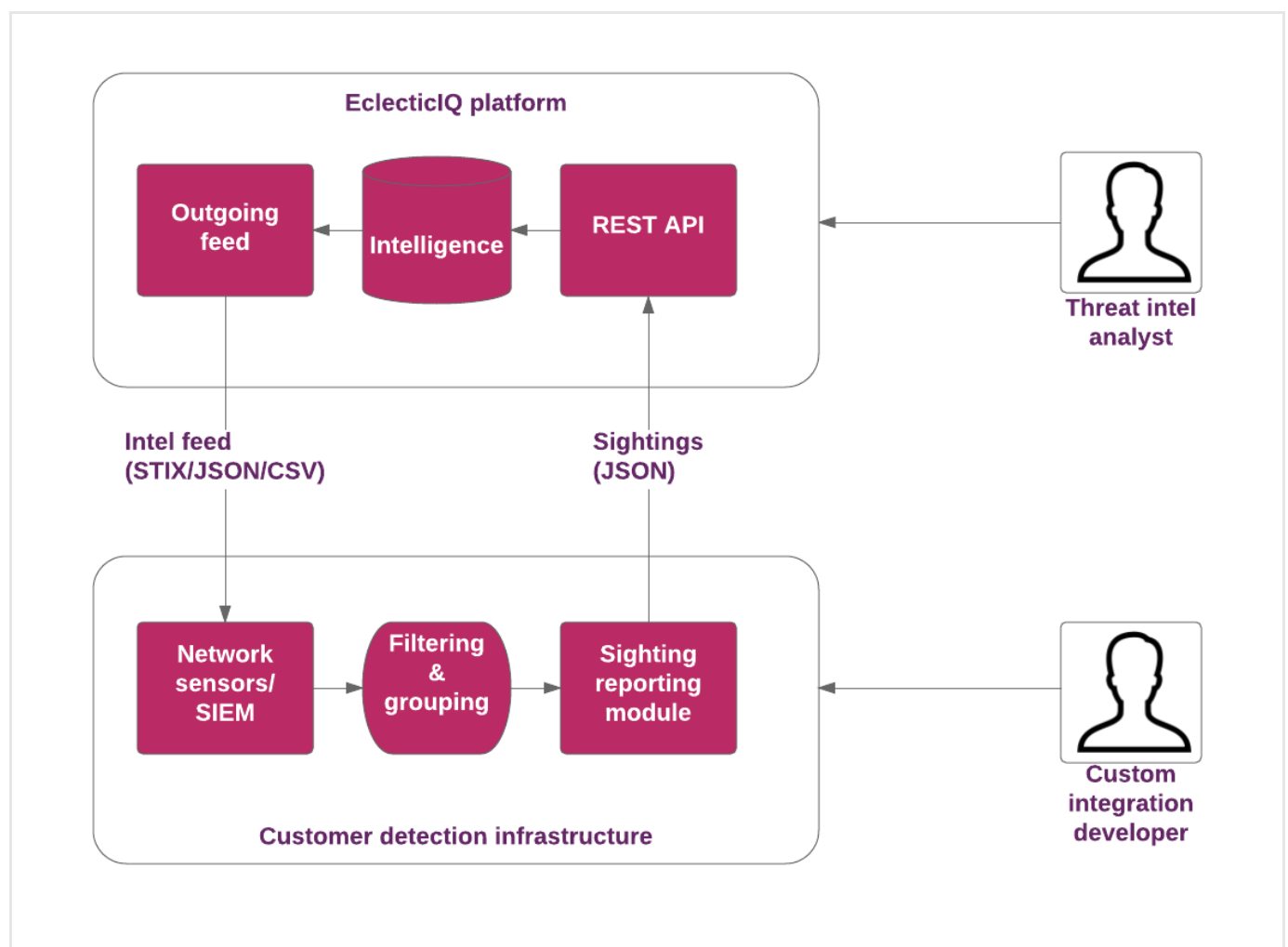
```
{
  "errors" : [
    {
      "detail" : "The requested URL was not found on the server. If you entered the
URL manually please check your spelling and try again.",
      "status" : 404,
      "title" : "Not Found"
    }
  ]
}
```

# How to report sightings through the API

Create and update sighting entities programmatically by making calls to the EclecticIQ API.

## Architecture overview

- A matching engine identifies hits in the outgoing feed data stream.
- The matching engine pushes the hits to a message broker.
- A grouping engine listens to events triggered by the message broker. The engine uses entity IDs to group hits into sightings.
- Sightings are then passed on to the EclecticIQ Platform.



## Before you start

It is a good idea to have a dedicated user and user group to handle automation tasks that interact with external products or components of your system. Therefore, before you start generating sightings and/or other entities programmatically, you may want to create a user and a user group to handle automation and integration tasks.

## Create an automation user group



**Warning:** The automation user group has to include all the data sources the group members, that is, the automation user profiles, need to access.

To add a new automation user group, do the following:

- On the left-hand navigation sidebar, click **System**.
- Under **User management**, click **Groups**.
- Under **Groups**, click the **+ Group** button.



On the forms, input fields marked with an asterisk are required.

- Under **Add new group**, define the following configuration settings:
  - **Name:** a descriptive name for the automation user group.
  - **Description:** a short description of the automation user group and its purpose.
  - **Allowed sources:** defines the cyber threat intelligence sources the group is allowed to access. Select here the sources the automation user group and its members need to access to fetch data from.
  - Click the **+ add** link.
  - From the **Source** drop-down menu, choose a source you want to make available to the automation user group.
  - From the **TLP** drop-down menu, choose a **Traffic Light Protocol** (<https://www.us-cert.gov/tlp>) color to filter the source data accordingly.
  - Click the **+ More** link to specify additional sources.
  - Click **Save** to store your changes, or **Cancel** to discard them.

## Create an automation user role

To add a new automation user role, do the following:

- On the left-hand navigation sidebar, click **System**.
- Under **User management**, click **Roles**.
- Under **Roles**, click the **+ Role** button.



On the forms, input fields marked with an asterisk are required.

- Under **Add new role**, define the following configuration settings:
  - **Name**: a descriptive name for the automation user role.
  - **Description**: a short description of the automation user role and its purpose.
  - **Available permissions**: this pane lists all available permissions the new role can be granted.
    - Select one or more permissions from the list.
    - Click **Add** to grant the role the permission(s) listed in the **Selected Permissions** pane.
    - Alternatively, start typing a permission name in the autocomplete text input field above the pane.
    - Select one or more filtered permissions from the list.
    - Click **Add** to grant the role the permission(s) listed in the **Selected Permissions** pane.
    - To revoke one or more permissions for the role, select the relevant entries under **Selected permissions**, and then click **Remove**.
  - Click **Save** to store your changes, or **Cancel** to discard them.

## About permissions

User permissions are predefined in the platform, and they are not editable or configurable. You can either assign them to user roles, or revoke them.

Permission names try to be as self-explanatory as possible:

- **Format**: `<type of action> <object of the action>`
- **Example**: *modify entities*

There are two permission actions:

- **modify**: a modification permission allows write operations.
- **read**: a read permission grants access to the data without allowing any modifications.

To get an overview of the available permissions on the platform, do the following:

- On the left-hand navigation sidebar, click **System**.

- Under **User management > Permissions**, the permission overview is displayed as a table, where each permission is assigned a row.
- To view permission details, click an area on a row.
- An overlay slides in from the side of the screen. It displays permission information in a flash-card format.

## Create an automation user

To add an automation user, do the following:

- On the left-hand navigation sidebar, click **System**.
- Under **User management**, click the **+ User** button.



On the forms, input fields marked with an asterisk are required.

- Under **System > User management > Edit user**, define the following configuration settings:
  - **First name**: n/a
  - **Last name**: n/a
  - **Username**: the designated user name to identify the user, when signed in to the platform. Choose a name that helps understand what the automation user does, for example “*Matching engine aggregator*”.
  - **Email**: an email address associated to the automation user. You can use this address to send and receive automated notifications.
  - **Enabled**: select this checkbox to enable the user.
  - **Administrator**: select this checkbox to elevate the user’s role to administrator. When the checkbox is selected, the user has administrator rights and permissions.
  - **Contact info**: n/a
  - **PGP public key**: the user’s **PGP public key** (<http://www.pgpi.org/doc/pgpintro/#p9>).
  - **Groups**: this pane lists all available groups the new user can be assigned to.
    - From the drop-down menu select one or more groups to assign the user to.
    - Alternatively, start typing a group name in the autocomplete text input field.
    - To remove the user from one or more groups, remove the relevant entries by clicking the **✕** corresponding to the group you want to remove the user from.
  - **Roles**: it works like **Groups**, the only difference being that instead of adding the user to one or more groups, this option assigns one or more roles to the user.
  - Click **Save** to store your changes, or **Cancel** to discard them.



## Get the automation user group ID

When you create a new entity by making a call to the platform API, you need to pass a group `source` ID value inside the `meta: {}` nested object.

Its value is the group ID the data source(s) *and* the user making the call belong to.



**Warning:** The automation user group has to include all the data sources the group members, that is, the automation user profiles, need to access.

### Step 1 of 2: get the group ID

To retrieve the group ID value you need to pass in your API calls, do the following:

- On the left-hand navigation sidebar click **System**.
- Under **User management**, click **Groups**.
- On the group table overview, click the row corresponding to the automation user group containing the data source(s) you want to use as input *and* the automation user making the API calls.
- The action returns a URL with the following format:  
`https://<platform_host>/#/configuration/system/user-management/groups?detail=<integer>`  
Example:  
`https://platform.host/#/configuration/system/user-management/groups?detail=30`
- The `detail` URL parameter value allows you to retrieve the `source` ID value you need to pass with the `meta: {}` nested object in your API calls.  
In the example, the `detail` value is 30; this is the group ID we need to retrieve the group `source` ID for our calls.

### Step 2 of 2: get the group source ID

To retrieve the group `source` ID value you need to pass in your API calls, do the following:

- Request all available platform user groups (requires authentication and bearer token)
- In the response, look for the group object with the `"id" : "<integer>"` key/value pair you previously retrieved.
- In the same group object, look for the `"source" : "<UUID_string>"` key/value pair.  
This is the group source ID you need to pass in your API calls to create sightings programmatically.

## Get the automation user group ID example

cURL API request — fetches all user groups

```
$ curl -X GET
-v
--insecure
-i
-H "Content-Type: application/json"
-H "Accept: application/json"
-H "Authorization: Bearer <token>"
https://platform.host/api/groups/
```

### API response — returns user group list

```
{
  "count": 18, // number of returned user groups

  "data": [

    ...

    {
      "allowed_sources": [

        {
          "allowed_tlp": "RED",
          "source": "8d49188e-2a2c-4192-92e3-c76b894c344b"
        },

        {
          "allowed_tlp": "RED",
          "source": "42c051f8-9f5b-4696-a629-b86c2ead955f"
        }

      ],

      "description": null,

      // group id, same value as the 'detail=' URL param for the group
      "id": 30,
      "name": "ddss",

      // the group source ID you need to pass in your API calls
      "source": "42c051f8-9f5b-4696-a629-b86c2ead955f",
      "type": "groups",
      "users": []
    },

    ...

  ]
}
```

## Authentication

The authentication mechanism is based on **JSON web tokens** (<http://jwt.io/>).

By default, the token expires 2 hours after successfully signing in. The corresponding session is terminated, and you need to sign back in to the platform.

When human interaction is detected — for example, keystrokes or mouse activity — the token is automatically refreshed every 2 minutes. This prevents the system from signing out users who may be working or saving data at that time.

Therefore, the default maximum amount of idle time without any human interaction before being automatically signed out equals to *session token validity - 2 minutes*.

To authenticate and access the platform, do the following:

- Make a `POST` call.
- In the call, pass your authentication credentials as a JSON object to the `/auth` endpoint. The credential data is used to generate a token that is returned with the response.

You need to include the generated bearer token in the `Authorization` HTTP header with each subsequent API call.

The `Authorization` HTTP header has the following format: `Authorization: Bearer <token>`

## Auth request

<b>API endpoint</b>	<code>/auth</code>
<b>Auth method</b>	<code>POST</code>
<b>HTTP headers</b>	<code>"Content-Type: application/json", "Accept: application/json"</code>
<b>API request</b>	<code>POST + "Content-Type: application/json" + "Accept: application/json"</code> <code>+ { "username": "&lt;valid_user_name&gt;", "password": "&lt;valid_password&gt;"</code> <code>} + &lt;platform_host&gt;/auth</code>
<b>API response</b>	<code>{ "expires_at": "&lt;expiry timestamp&gt;", "token": "&lt;token&gt;" }</code>

The following example uses cURL to authenticate:

```
$ curl -X POST
  -H "Content-Type: application/json"
  -d '{ "username" : "<valid_user_name>", "password" : "<valid_password>" }'
https://platform.host/api/auth
```

## Auth response

When the user name and password credential are valid, the `POST` call returns a JSON web token:

```
{
  "expires_at": "2016-03-30T12:11:40.078219+00:00",
  "token":
  "abHpYXQiOjE0NTkzMzI3MDAsIm4TcCI6MTQ1OTMzOTkwMCwiYWxnIjoisSFMyNTYifQ.oyY1c2VyX2lkIjo1fQ.LQQ3NdUHp4s-QCXsxq3feI0Dy6tf5XQX9DOML1RNIzQ"
}
```

You need to include the bearer token value in each subsequent API call. You pass the token by including an `Authorization` HTTP header in the API request.

The `Authorization` HTTP header has the following format: `Authorization: Bearer <token>`

In the following example, you make a `GET` request to the `/api/` endpoint to retrieve a list of the available API endpoints and the corresponding methods:

```
$ curl -X GET
-v
--insecure
-i
-H "Content-Type: application/json"
-H "Accept: application/json"
-H "Authorization: Bearer <token>"
https://platform.host/api/
```



### Warning:

#### About cURL calls

- If you make HTTPs cURL calls to the API *and* you have a self-signed or an invalid certificate, include the `-k` or the `--insecure` parameter in the cURL call to skip the SSL connection CA certificate check.
- Always append a `/` trailing slash at the end of an API URL endpoint. The only exception is `/auth`, which does not take a trailing slash.
- In the cURL call, the `-d` data payload with the entity information always needs to be flat JSON, not hierarchical JSON.  
If you want to pass a hierarchical JSON object, include the `--data-binary` parameter, followed by the path to the JSON file, for example `@/path/to/entity_file.json`.

## Create a sighting

API endpoint	/entities/
Create method	POST
HTTP headers	"Content-Type: application/json", "Accept: application/json", "Authorization: Bearer <token>"
API request	POST + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer <token>" + { "data" : { <entity> } } + <platform_host>/entities/
API response	{ "data" : { <entity> } }

When you make a `GET` request and obtain a JSON object with entity data in the response, the entity object is wrapped in a data wrapper: { "data" : { ... } }.

When you pass a JSON object with entity data in the body of your API request, you always need to wrap it in a data wrapper: { "data" : { ... } }.

## Creation request

To create a new sighting with an API call, do the following:

- Make a `POST` call to the `/entities/` API endpoint. The request should include the following HTTP headers:
  - "Authorization: Bearer <token>" — *Required*
  - "Content-Type: application/json" — *Required*
  - "Accept: application/json" — *Optional*
- In the request message body, pass the JSON object containing the data defining the new sighting you want to create. Make sure the sighting data is wrapped inside the { "data" : } wrapper.

## Creation response

When the API request payload successfully passes validation:

- A new sighting is created with the specified data.

- The API returns a JSON object containing the newly created sighting data, including a new unique `id` for the sighting.
- The sighting data is wrapped inside the `{ "data" : }` wrapper. The wrapper needs to include at least the following required information:
  - `data`: a nested JSON object holding the information that describes the sighting, like name, short description, any extra details like kill chain phase or TLP color, and so on.
    - `type`: it is inside `data`. It identifies the entity type being created. For sightings, it is always `eclecticiq-sighting`.
    - `title`: it is inside `data`. The name you assign to the new sighting.
  - `meta`: a nested JSON object holding information about the data being posted through the API call, like source and tags, if any.
    - `source`: it is inside `meta`. Its value corresponds to the group ID referring to the source of the information used to create the sighting, for example an incoming feed or a user group who in turn can have a dataset or an incoming feed as its source. This value is automatically generated when a new source is created in the platform.

## Creation examples

cURL API request — creates new sighting

```
$ curl -X POST
      -v
      --insecure
      -i
      -H "Content-Type: application/json"
      -H "Accept: application/json"
      -H "Authorization: Bearer <token>"
      -d '{ "data": { <entity> } }'
      https://platform.host/api/entities/
```

HTTP request message body

It contains the data for the new sighting entity you want to create.

```
{
  "data": {
    "data": {
      "type": "eclecticiq-sighting",
      "title": "test-eclecticiq-sighting"
    },
    "meta": {
      "source": "091bde3a-4a07-4e0f-b834-4038d2041932"
    }
  }
}
```

#### API response — successful sighting creation

```
HTTP 201 OK CREATED
```

The response body contains the data for the newly created sighting. Among the fields included in the sighting, you can notice a group ID and an entity ID:

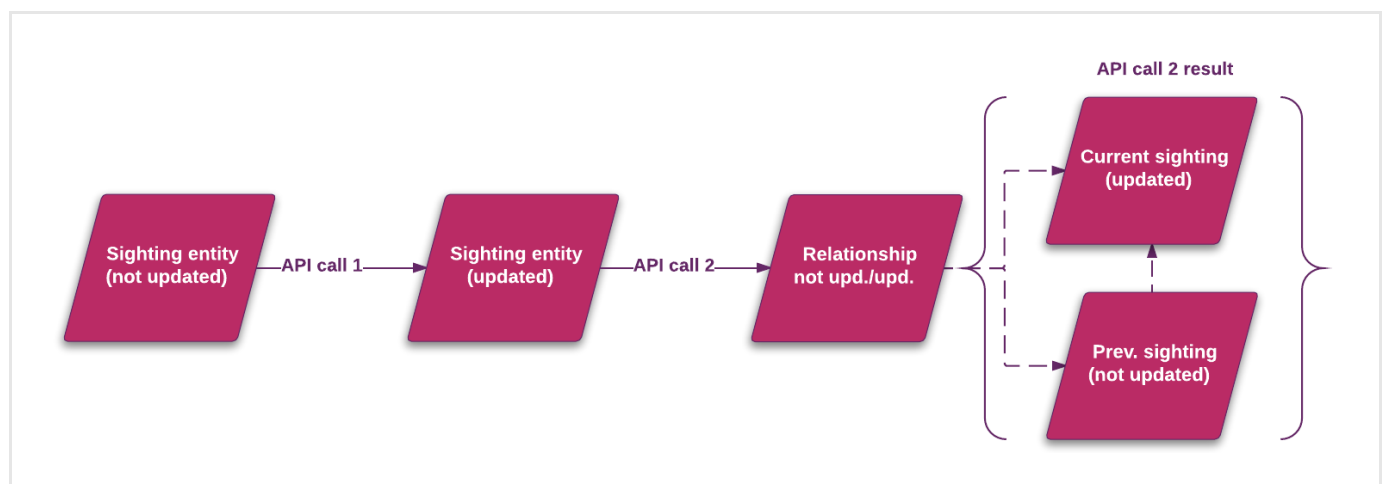
- `group_id`: the entity group the new sighting is assigned to.
- `id`: a unique identifier for the sighting entity.

```

{
  "data": {
    "data": {
      "title": "test-eclecticiq-sighting",
      "type": "eclecticiq-sighting"
    },
    "group_id": "b4585e35-c8a9-478e-a31d-ef7e2ec72aaf",
    "id": "d80da575-4788-4ef5-b0a5-cc0e9b506298",
    "meta": {
      "estimated_observed_time": "2016-03-30T14:22:02.929028+00:00",
      "estimated_threat_start_time": "2016-03-30T14:22:02.929028+00:00",
      "ingest_time": "2016-03-30T14:22:02.929028+00:00",
      "is_outdated_version": false,
      "source": "091bde3a-4a07-4e0f-b834-4038d2041932",
      "source_name": "test",
      "source_type": "group",
      "title": "test-eclecticiq-sighting"
    },
    "source": "091bde3a-4a07-4e0f-b834-4038d2041932",
    "type": "entities"
  }
}

```

## Update a sighting



- *API call 1* creates a new version of the sighting containing the updated information.
- *API call 2* relates the previous, not updated version of the sighting, with the new, updated one.

To keep data in sync with the outside world and to avoid inconsistencies, updating an entity is more similar to versioning than to a simple edit-to-update process.



A JSON entity includes two main nested objects:

- `{ "data" : { } }`  
`data` contains the non-modifiable information describing the sighting entity. This data cannot be edited, because it needs to be in sync with the same entity data in the outside world.
- `{ "meta" : { } }`  
`meta` contains the editable entity metadata. You can modify this data at any time.

When you pass a JSON object with entity data in the body of your API request, you always need to wrap it in a data wrapper: `{ "data" : { ... } }`.

<b>API endpoint</b>	<code>/entities/</code>
<b>Update method</b>	POST
<b>HTTP headers</b>	<code>"Content-Type: application/json", "Accept: application/json", "Authorization: Bearer &lt;token&gt;"</code>
<b>API call 1</b>	<code>POST + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer &lt;token&gt;" + { "data" : { &lt;new_entity&gt; } } + &lt;platform_host&gt;/entities/</code>
<b>API response 1</b>	<code>{ "data" : { &lt;new_entity&gt; } }</code>
<b>API call 2</b>	<code>POST + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer &lt;token&gt;" + { "data": { "data": { "source": "&lt;new_entity_id&gt;", "source_type": "&lt;entity_type&gt;", "target": " &lt;superseded_entity_id&gt;", "target_type": " &lt;entity_type_same_as_source&gt;", "type": "relation", "subtype": "stix_update_of" } } } + &lt;platform_host&gt;/entities/</code>
<b>API response 2</b>	<code>{ "data": { "data": { "source": "&lt;new_entity_id&gt;", "source_type": " &lt;entity_type&gt;", "target": "&lt;superseded_entity_id&gt;", "target_type": "&lt;entity_type_same_as_source&gt;", "type": "relation", "subtype": "stix_update_of" } } }</code>

## Update request

To update an existing entity, you need to make two consecutive API calls to perform the following actions:

- Create a new version of the entity containing any changed information you want to store in the updated entity.
- Create a relationship between the new version of the entity you just created and the previous, superseded version of the same entity.

The keys holding entity version information for the update are inside the nested `data` JSON object:

- `source`: holds the `id` value of the new, *updated* entity.
- `target`: holds the `id` value of the *previous*, superseded entity.

- `type`: when updating entities, its value is always `relation`.
- `subtype`: when updating entities, its value is always `stix_update_of`.

At the end of the process, the updated `source` entity has a `stix_update_of` relation with the previous/superseded `target` entity.

## Update response

A successful update operation returns two responses:

- The first call returns a JSON object containing the newly created entity with the updated information.
- The second call returns a JSON object that defines a `stix_update_of` relationship between the updated entity (`source`) and the superseded one (`target`).

## Update examples

cURL API request — updates existing sighting — update 1/2

The first call you make to update an existing entity creates a new entity of the same type, which is an updated version with changed details of the existing entity.

```
$ curl -X POST
      -v
      --insecure
      -i
      -H "Content-Type: application/json"
      -H "Accept: application/json"
      -H "Authorization: Bearer <token>"
      -d '{ "data": { <entity> } }'
      https://platform.host/api/entities/
```

HTTP request message body

The message body of the first call contains the new version of the sighting entity with updated/changed data.

```
{
  "data": {
    "data": {
      "type": "eclecticiq-sighting",
      "title": "test-eclecticiq-sighting",

      "confidence": {
        "type": "confidence",
        "value": "Medium"
      },

      "description": "<p>This is a very scary sighting for test purposes, which I am
updating to add even more evil to it.</p>",
      "description_structuring_format": "html",

      "handling": [{
        "type": "marking-specification",

        "marking_structures": [{
          "type": "marking-structure",
          "marking_structure_type": "tlp",
          "color": "AMBER"
        }]
      }]
    },

    "meta": {
      "source": "091bde3a-4a07-4e0f-b834-4038d2041932"
    }
  }
}
```

#### API response — successful sighting creation — update 1/2

```
HTTP 201 OK CREATED
```

The response body contains the new, updated sighting data.

```
{
  "data": {
    "data": {
      "confidence": {
        "type": "confidence",
        "value": "Medium"
      },
      "description": "<p>this is a very scary sighting for test purposes, which I am
updating to add more evil</p>",
      "description_structuring_format": "html",
      "handling": [{
        "marking_structures": [{
          "color": "AMBER",
          "marking_structure_type": "t1p",
          "type": "marking-structure"
        }],
        "type": "marking-specification"
      }],
      "title": "test-eclecticiq-sighting",
      "type": "eclecticiq-sighting"
    },
    "group_id": "72525f33-cfab-44fb-a46a-b1bd37675b0b",
    "id": "c38b51d0-ce71-49f6-918e-13b6b299f4ee",
    "meta": {
      "estimated_observed_time": "2016-03-30T16:12:43.499083+00:00",
      "estimated_threat_start_time": "2016-03-30T16:12:43.499083+00:00",
      "ingest_time": "2016-03-30T16:12:43.499083+00:00",
      "is_outdated_version": false,
      "source": "091bde3a-4a07-4e0f-b834-4038d2041932",
      "source_name": "test",
      "source_type": "group",
      "title": "test-eclecticiq-sighting"
    },
    "source": "091bde3a-4a07-4e0f-b834-4038d2041932",
    "type": "entities"
  }
}
```

The second call creates a relationship between the updated sighting entity (**source**) and the superseded one (**target**).

```
$ curl -X POST
      -v
      --insecure
      -i
      -H "Content-Type: application/json"
      -H "Accept: application/json"
      -H "Authorization: Bearer <token>"
      -d '{ "data": { "data": { "source": "<updated_entity_id>", "source_type":
"eclecticiq-sighting", "target": "<superseded_entity_id>", "target_type": "eclecticiq-
sighting", "type": "relation", "subtype": "stix_update_of" } } }'
      https://platform.host/api/entities/
```

### HTTP request message body

The request message body of the relationship creation call to link the updated and the superseded entities is a **data** JSON object with the following format:

```
{
  "data" : {
    "source"      : "<updated_entity_id>",
    "source_type" : "eclecticiq-sighting",
    "target"      : "<superseded_entity_id>",
    "target_type" : "eclecticiq-sighting",
    "type"        : "relation",
    "subtype"     : "stix_update_of"
  }
}
```

In your API request, make sure you wrap the **data** object in the { "data" : { ... } } wrapper.

```
{
  "data": {
    "data": {
      "source": "c38b51d0-ce71-49f6-918e-13b6b299f4ee",
      "source_type": "eclecticiq-sighting",
      "target": "d80da575-4788-4ef5-b0a5-cc0e9b506298",
      "target_type": "eclecticiq-sighting",
      "type": "relation",
      "subtype": "stix_update_of"
    }
  }
}
```

## API response — successful sighting update — update 2/2

```
HTTP 201 OK CREATED
```

The response to the relationship creation call to link the superseded and the updated entities is a `data` JSON object describing the established relationship between the older (`target`) and the newer (`source`) versions of the same entity.

```
{
  "data": {
    "data": {
      "source": "c38b51d0-ce71-49f6-918e-13b6b299f4ee",
      "source_type": "eclecticiq-sighting",
      "target": "d80da575-4788-4ef5-b0a5-cc0e9b506298",
      "target_type": "eclecticiq-sighting",
      "type": "relation",
      "subtype": "stix_update_of"
    },
    "group_id": "6768f516-627d-4d9e-916c-1db63622321a",
    "id": "f7c04e61-6086-45a0-bebb-8cd31581a476",
    "meta": {},
    "source": null,
    "type": "entities"
  }
}
```

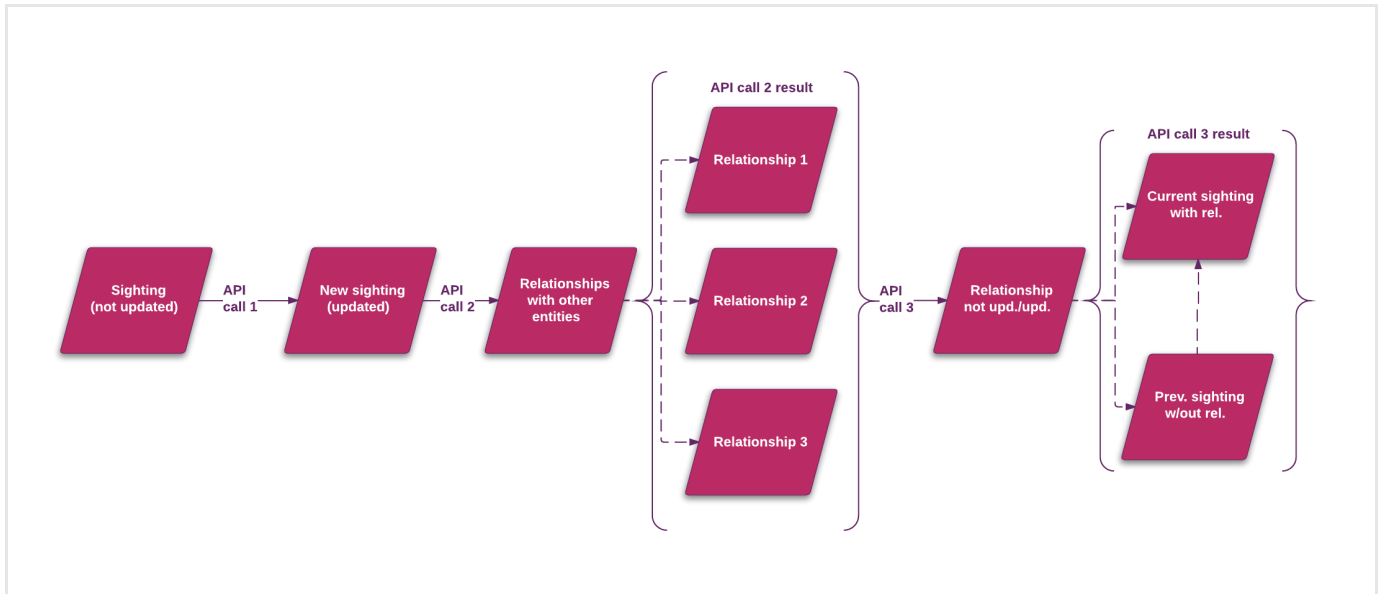
To avoid unnecessary duplication, if you pass the same update data for an entity more than once, the redundant update that would cause data duplication is ignored.

In this case, the HTTP response code notifies that no new data was created:

```
HTTP 200 OK
```

In the `data` JSON object containing the relationship details, the `subtype` JSON key holds the `stix_duplicate_of` value to notify that the update was ignored because it was a duplicate.

## Create relationships



- *API call 1* creates a new version of the sighting.
- *API call 2* adds relationships to the new version of the sighting.
- *API call 3* updates the previous version of the sighting without a relationship to the new one with a relationship.

<b>API endpoint</b>	/entities/
<b>Update method</b>	POST
<b>HTTP headers</b>	"Content-Type: application/json", "Accept: application/json", "Authorization: Bearer <token>"
<b>API call 1</b>	POST + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer <token>" + { "data" : { <new_entity> } } + <platform_host>/entities/
<b>API response 1</b>	{ "data" : { <new_entity> } }
<b>API call 2</b>	POST + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer <token>" + { "data": { "data": { "source": "<new_entity_id>", "source_type": "<entity_type>", "target": " <entity_id_to_relate>", "target_type": "<entity_type_to_relate>", "type": "relation" } } } + <platform_host>/entities/
<b>API response 2</b>	{ "data": { "data": { "source": "<new_entity_id>", "source_type": " <entity_type>", "target": "<entity_id_to_relate>", "target_type": " <entity_type_to_relate>", "type": "relation" } } }
<b>API call 3</b>	POST + "Content-Type: application/json" + "Accept: application/json" + "Authorization: Bearer <token>" + { "data": { "data": { "source": "<new_entity_id>", "source_type": "<entity_type>", "target": " <superseded_entity_id>", "target_type": " <entity_type_same_as_source>", "type": "relation", "subtype": "stix_update_of" } } } + <platform_host>/entities/

**API response 3**

```
{ "data": { "data": { "source": "<new_entity_id>", "source_type": "<entity_type>", "target": "<superseded_entity_id>", "target_type": "<entity_type_same_as_source>", "type": "relation", "subtype": "stix_update_of" } } }
```

## Relationship request

To create a relationship between two entities, for example between a sighting and an incident, you need to make three consecutive API calls to perform the following actions:

- Create a new version of the sighting. Adding a relationship updates the sighting, and therefore it is necessary to create a new version of this entity to reflect the change.
- Create a relationship between the new version of the sighting and the incident.
- Create a relationship between the new version of the sighting, i.e. the one that is now related to the incident, and the previous one, i.e. the one that is not related to the incident. This updates the sighting to the new version.

The keys holding entity version information for the process are inside the nested `data` JSON object:

- `source`: holds the `id` value of the *new* sighting you want to add a relationship to.
- `target`: holds the `id` value of the target entity of the relationship. In other words, this is the entity you want to relate the sighting to. In this example, it is an incident.
- `type`: when creating relationships, its value is always `relation`.

## Relationship response

A successful relationship creation returns three responses:

- *API call 1* returns a JSON object with the new version of the sighting.
- *API call 2* returns a JSON object containing the data about the relationship between the sighting and the incident. This object represents the relationship between the entities.
- *API call 3* relates the previous version of the sighting without a relationship to the new one with a relationship. This call updates the sighting to include the relationship information.

## Relationship examples



## cURL API request — creates new version of sighting — relationship 1/3

The first API call creates a new version of the sighting.

```
$ curl -X POST
      -v
      --insecure
      -i
      -H "Content-Type: application/json"
      -H "Accept: application/json"
      -H "Authorization: Bearer <token>"
      -d '{ "data": { <new_entity> } }'
      https://platform.host/api/entities/
```

## HTTP request message body

It contains the data of the sighting you want to relate to another entity (for example, to an incident).

```
{
  "data": {
    "meta": {
      "source_name": "test",
      "source_type": "group",
      "source": "091bde3a-4a07-4e0f-b834-4038d2041932",
      "title": "test-eclecticiq-sighting"
    },
    "data": {
      "type": "eclecticiq-sighting",
      "title": "test-eclecticiq-sighting"
    }
  }
}
```

## API response — successful sighting creation — relationship 1/3

```
HTTP 201 OK CREATED
```

It returns a new version of the sighting, with a new `id`. You are using this `id` to relate the sighting to another entity.

```
{
  "data": {

    "data": {
      "title": "test-eclecticiq-sighting",
      "type": "eclecticiq-sighting"
    },

    "group_id": "7ae5bfc3b-5a20-4fc9-9b2d-ad655a3d5408",
    "id": "85939e81-99b3-4476-b9b5-5b8e17cea2b5",

    "meta": {
      "estimated_observed_time": "2016-03-31T12:14:04.398700+00:00",
      "estimated_threat_start_time": "2016-03-31T12:14:04.398700+00:00",
      "ingest_time": "2016-03-31T12:14:04.398700+00:00",
      "is_outdated_version": false,
      "source": "091bde3a-4a07-4e0f-b834-4038d2041932",
      "source_name": "test",
      "source_type": "group",
      "title": "test-eclecticiq-sighting"
    },

    "source": "091bde3a-4a07-4e0f-b834-4038d2041932",
    "type": "entities"
  }
}
```

### cURL API request — creates relationship — relationship 2/3

The second call establishes a relationship between the new version of the sighting you just created with the previous call, and another, different entity; for example, an incident.

```
$ curl -X POST
      -v
      --insecure
      -i
      -H "Content-Type: application/json"
      -H "Accept: application/json"
      -H "Authorization: Bearer <token>"
      -d '{ "data": { "data": { "source": "<updated_sighting_id>", "source_type":
"eclecticiq-sighting", "target": "<incident_id>", "target_type": "incident", "type":
"relation" } } }'
      https://platform.host/api/entities/
```

## HTTP request message body

The request message body of the relationship creation call is a `data` JSON object with the following format:

```
{
  "data" : {
    "source"      : "<id_of_the_entity_you_want_to_create_a_relationship_for>",
    "source_type" : "eclecticiq-sighting",
    "target"      : "<id_of_the_entity_you_want_to_relate_to_the_source>",
    "target_type" : "incident",
    "type"        : "relation"
  }
}
```

In your API request, make sure you wrap the `data` object in the `{ "data" : { ... } }` wrapper.

```
{
  "data" : {
    "data": {
      "source": "85939e81-99b3-4476-b9b5-5b8e17cea2b5",
      "source_type": "eclecticiq-sighting",
      "target": "dcb550e7-4178-4d1f-ad59-072e6d7aa1c7",
      "target_type": "incident",
      "type": "relation"
    }
  }
}
```

## API response — successful relationship creation — relationship 2/3

```
HTTP 201 OK CREATED
```

A successful response returns a JSON object containing the details of the newly established relationship between the two different entities.

```
{
  "data": {

    "data": {
      "source": "85939e81-99b3-4476-b9b5-5b8e17cea2b5",
      "source_type": "eclecticiq-sighting",
      "target": "dcb550e7-4178-4d1f-ad59-072e6d7aa1c7",
      "target_type": "incident",
      "type": "relation"
    },

    "group_id": "decde301-d4c4-4387-a8fd-7f423cb6ca2c",
    "id": "2522daa2-638d-4e11-bac0-b4c18bf2f394",

    "meta": {
      "is_outdated_version": false
    },

    "source": null,
    "type": "entities"
  }
}
```

### cURL API request — updates sighting — relationship 3/3

The third call updates the sighting, in this example the source entity, by creating a `stix_update_of` relationship between the previous version — the one without a relationship to the incident — and the new one — the one that bears a relationship to the incident.

```
$ curl -X POST
  -v
  --insecure
  -i
  -H "Content-Type: application/json"
  -H "Accept: application/json"
  -H "Authorization: Bearer <token>"
  -d '{ "data": { "data": { "source": "<updated_entity_id_with_relationship>",
"source_type": "eclecticiq-sighting", "target": "
<superseded_entity_id_no_relationship>", "target_type": "eclecticiq-sighting",
"type": "relation", "subtype": "stix_update_of" } } }'
  https://platform.host/api/entities/
```

### HTTP request message body

The request message body of the update call is a `data` JSON object with the following format:

```
{
  "data" : {
    // new version of the sighting related to incident
    "source"      : "<updated_entity_id>",
    "source_type" : "eclecticiq-sighting",
    // previous version of the sighting not related to incident
    "target"      : "<superseded_entity_id>",
    "target_type" : "eclecticiq-sighting",
    "type"        : "relation",
    "subtype"     : "stix_update_of"
  }
}
```

In your API request, make sure you wrap the `data` object in the `{ "data" : { ... } }` wrapper.

```
{
  "data": {
    "data": {
      "source": "85939e81-99b3-4476-b9b5-5b8e17cea2b5",
      "source_type": "eclecticiq-sighting",
      "target": "c38b51d0-ce71-49f6-918e-13b6b299f4ee",
      "target_type": "eclecticiq-sighting",
      "type": "relation",
      "subtype": "stix_update_of"
    }
  }
}
```

### API response — successful sighting update — relationship 3/3

```
HTTP 201 OK CREATED
```

A successful response returns a JSON object containing the details of the version relationship between the two versions of the same entity.

```
{
  "data": {
    "data": {
      "source": "85939e81-99b3-4476-b9b5-5b8e17cea2b5",
      "source_type": "eclecticiq-sighting",
      "subtype": "stix_update_of",
      "target": "c38b51d0-ce71-49f6-918e-13b6b299f4ee",
      "target_type": "eclecticiq-sighting",
      "type": "relation"
    },
    "group_id": "e73c48be-75ea-40ff-b478-890a9b8b1329",
    "id": "6e4ffcad-f62a-4d21-a20a-bc85a42be274",
    "meta": {},
    "source": null,
    "type": "entities"
  }
}
```

## Error handling

When an error occurs, the API handles it by returning a **4xx (input/client error)** or **5xx (server issue)** HTTP **response code** (<https://httpstatuses.com/>), and by including a short explanatory message in the JSON response body.

The key/value pairs in the `errors` JSON object provide more details about the possible cause of the error, which should help you solve it.

- `status`: the HTTP response code.
- `title`: the name of the error.
- `detail`: a short explanation of the issue with more context, when available.

```
{
  "errors" : [
    {
      "detail" : "The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.",
      "status" : 404,
      "title" : "Not Found"
    }
  ]
}
```

