API GUIDE 6.4.0

VERSION 2.0

APRIL 2020

Copyright

Trademark

Table of Contents

Overview for Developers

The FortiSOAR™ application uses a Representational State Transfer (RESTful) API with a Service Oriented Architecture (SOA) to allow for scalability, flexibility, and ease of use. The characteristics of this design include:

- Simplicity of interface
- Scalability, especially when combined within the SOA microservices concepts
- Maintainability, while the application code may change underneath the API, the API definition may remain in place
- Performance, component interactions dominate the user-perceived performance and efficiency

The API gives you access to the primary data available to the core system, which are also consumed by the Angular.js client for front end interaction.

FortiSOAR™ uses the Hydra Specification for implementing the APIs.

Note: While direct access to the module data models and endpoints is provided, we recommend that you consider the Playbooks system as the primary method for working to get data input and output from the FortiSOAR™ platform. The Playbook system makes development simpler, easier, and more flexible for both ingest and export of data via RESTful methods without requiring debugging or maintenance when new releases are available. However, there may be cases where the Playbooks system does not allow for the required functionality, and custom development on the direct API is needed.

Concepts

JSON-LD

The API uses the JSON for Linking Data (JSON-LD) format to present information. JSON-LD empowers humans and machines to read and write across the core API without pre-existing knowledge of the API structure, other than the root API endpoint. JSON-LD uses embedded links to determine the Context of the API endpoints quickly and easily in a human-readable format.

Example: API endpoints for `"@context": "/api/3/contexts/"` and `"@id": "/api/3/"` are standard pieces of the JSON-LD format.

Hydra Vocab

Additionally, the JSON-LD structure is combined with the Hydra API definition, which applies a standard Core Vocabulary to all data within the API. The Hydra definition is a

combination of the Core Vocabulary and the JSON-LD structure to completely define the entirety of the API within itself, making the API standardized and discoverable at the same time.

Note: Any API information described as `"@vocab"` is a part of the core Hydra definition.

A full list of all of the endpoints and the context around them can be found at `/api/3/apidoc`.

UUIDs

Universally unique identifiers (UUIDs) are used throughout the system to ensure all records have a unique identifier. The canonical displayed versions are 36 characters using hexadecimal text with inserted hyphen characters in the following format: `123e4567-e89b-12d3-a456-426655440000`

When creating new records directly in the API, you may optionally include a generated UUID if you wish to retain that UUID in an external system for reference. This must be a differentiated UUID from any other record in the system, or it will fail.

If you do not include a UUID during a POST, a UUID will be generated for you. This is the recommended method as the response for the POST will contain the IRI with included UUID that you may retrieve and save.

IRIs

All foreign key references use International Resource Identifiers (IRIs) to link records within the system. The format for this is essentially identical to a URI, forming a relative link within the application, though there are differences in the formats. The identifier uses a UUID associated with the record appended to the record location from the API root.

```
/api/3/alerts/{uuid}
```

IRIs will be seen in the API typically at the top of a record using the key `"@id"`.

Tip: The IRI is automatically generated when inserting the record and is not intended to be used during any POST activities. POST activities should use the `"uuid":` identifier with a plain 36-character UUID.

Organization

The REST API is organized around the core Modules defined within the platform. Note that most Modules are defined on a 1:1 basis with a database table, though this is not always the case. The root API may be found at `https://FortiSOAR_ROOT.URL/api/3`, but can only be accessed with the proper authentication. See the `Authentication API` section in the *API Methods* chapter for more information.

All Modules and models are made available in the core API, Modules representing a specific data table accessible within the UI. New Modules may be added via the Module Editor in the Administration menu, and every new Module will add a new API endpoint for accessing the records within that Module.

Authentication

Authentication methods for using the API are available via two methods:

1. **Hash-based message authentication code (HMAC)** - this method involves computing a specific cryptographic hash via a cryptographic key within the parameters specified from FortiSOAR™. This is the required method for permanent API access. See the HMAC Authentication section for more details.
   Note: For HMAC authentication the timestamp must be in UTC format.
2. **Token-based authentication** - this method is a temporary access mechanism by using a SHA-256 token, available within the browser resources of the UI. This token expires on a periodic basis and must be refreshed, however, it is useful for temporary access when doing specific API actions during development where generating an HMAC key is not possible, e.g., REST clients and CURL methods.
   The token must be placed in the Header with the `Authorization` key and the format for identifying the user a `Bearer %token%`

FortiSOAR™ utilizes a sessionless security model, meaning the server does not track individual user sessions as objects. Users must be authenticated with a unique token that has a limited lifespan, typically configured at 30 minutes though that may be adjusted. For more information, see the `Authentication API` section in the *API Methods* chapter.

Using the HMAC method does not require a session because each message is individually signed with the cryptographic key, but this must be computed, and the server clocks must be in sync for proper operation.

Note: Basic authentication and No authentication webhooks are allowed in the context of the Custom API Endpoint for Playbooks. This is purposeful to reduce the overall risk profile of allowing for less secure methods on the core API.

Authorization

Every API action is reviewed against user authorization privileges. These are determined by your Team membership and Role assignment as described in the "Administration Guide.

Tip: We recommend that you always use a limited privilege set scoped to the desired operations to ensure both security and guard against potential data loss due to accidental actions.

Operation

Exercising the API is possible via any valid REST client, such as Postman, or using CURL combined with a user token. All valid CRUD operations allowed via your user privileges may be invoked, but not all endpoints support all HTTP methods.

Supported HTTP Methods

- **GET [Read]**
- **POST [Create]**
- **PUT [Update]**
- **DELETE [Delete]**

URI Encoding

The most common format for traversing the API involves using the Module names in the following format.

```
/api/3/{module name}
```

An example endpoint for Alerts is as follows:

```
/api/3/alerts
```

The body response will include a list of the Alert records found within the database.


```
{
    "@context": "/api/3/contexts/Alert",
    "@id": "/api/3/alerts",
    "@type": "hydra:PagedCollection",
    "hydra:nextPage": "/api/3/alerts?%24page=2",
    "hydra:totalItems": 1,
    "hydra:itemsPerPage": 30,
    "hydra:firstPage": "/api/3/alerts",
    "hydra:lastPage": "/api/3/alerts?%24page=2",
    "hydra:member": [
        {
        "@id": "/api/3/alerts/028b37fa-bb35-4e3b-8afb-a3274a8cb34
3",
        "@type": "Alert",
        "name": "Malicious Attachment - Malware.Binary.Vbs",
        "sourceId": null,
        "source": "FireEye EX - Email MPS",
        "origin": "",
```

```
        "dueDate": null,
        "description": null,
        "type": null,
        "severity": null,
        "status": null,
        "assignedTo": null,
        "createUser": {
        "@id": "/api/3/appliances/432ba9fe-0955-4379-9177-68b0d87
e8caf",
        "@type": "Appliance",
        "name": "Workflow",
        "userId": "7c23b46e-5b84-47c0-8f7f-fe0498156c81",
        "createUser": null,
        "createDate": null,
        "modifyUser": "/api/3/people/bcb46d5a-0ad8-4480-ac68-9ebc
28502a30",
        "modifyDate": 1457479171,
        "id": 502,
        "@settings": "/api/3/user_settings/432ba9fe-0955-4379-917
7-68b0d87e8caf"
        }]
    "hydra:search": {
    "@type": "hydra:IriTemplate",
    "hydra:template": "/api/3/alerts{?}",
    "hydra:variableRepresentation": "BasicRepresentation",
    "hydra:mapping": []
    }
}
```


Syntax

FortiSOAR™ uses the CamelCase format for all data labels, e.g., keys for the key-value pairs in the JSON structure. CamelCase is the practice of writing a compound word or phrase structures where the first letter of the first word is lowercase and subsequent first letters of words following are capitalized.

Tip: We recommend you also use CamelCase format during your development to ensure consistency and readability.

Examples

```
"hydra:variableRepresentation"
"sourceId"
"dueDate"
```

API Endpoints Reference

This gives a summary of the API endpoints available within FortiSOAR™ organized via category.

Category	Description
Modules	General access for CRUD operations on records within the Modules defined in the system database
Playbooks	Access to both Playbook models, Step types (core system), Steps, and Custom Endpoints (Custom API Endpoint Triggers)
Configuration	General management of the configuration of the system, including Users / Appliances (actors), View Templates, and Module / Model definitions

Query Parameters

Most endpoints allow for query parameters to be appended to the URL for scoping the specific data set being returned. Parameters are used to filter results from the query and change the result format.

Syntax

1. The beginning of the query string is denoted by a `?` appended to the end of a valid endpoint URL.
2. Operational parameter values should start with a `$`.
3. Filter values should not have a `$`.
4. Parameters specify the specific value using `=`.
5. Parameters are chained using the `&` character.

Sample URL

```
https://FortiSOAR.ROOT.URL/api/3/{collection}/{uuid}?parameter1=value&parameter2=value
```

Operations

The standard query parameters are given in the following table.

Parameter	Valid Values	Action
$relationships	true, yes, false (default)	Show or hide full field and value context for

		related records
$export \| true, yes, false (default) \| Display record in exportable/importable JSON format, e.g., `"@id"` is turned from an IRI to a simple `"uuid"` value \| \| {fieldName} \| Any valid format for field type with url-encoded for special characters \| Filter results by `fieldName` using the equivalence operator. A `fieldName` can contain double underscores to specify a key within an object. For example, `status__itemValue` or `assignedToPerson__firstname`. \| \| {fieldName}${operator}	Any valid format for field type with url-encoded for special characters	Filter results by `fieldName` using any operator. Operators that are supported are found in the *Query API Reference* chapter. A `fieldName` can contain double underscores to specify a key within an object.
$orderby	Any valid `{fieldName}`, optional `-{fieldName}` operand	Sort the returned data set by the values in a specific field, may be chained and accepts `-value` for inverse ordering
$limit	An integer between 1 and 214748364	Maximum number of records to return in a query (default is 30)

Examples

```
/api/3/alerts?name=Full Alert Name
/api/3/alerts?name$like=%partial name%
/api/3/alerts?eventCount$gte=10&eventCount$lt=20
/api/3/alerts?status__itemValue=Open
/api/3/tasks?assignedToPerson__firstname$like=%doe%
/api/3/alerts?status__itemValue$in=Open|Resolved|In Progress
/api/3/alerts?$limit=100&$orderby=-createDate&$relationships=true
```

Module Endpoints

Module endpoints are grouped here by the Component to which they belong. At the root of the API, the models are not organized by Component. The component definition only exists within the system view metadata defining the navigation.

Note: An important concept to grasp is that the Modules within the UI generally refer to a 1:1 relationship with a table. Beyond the UI, many of the Modules in the Role Editor are actually composed of multiple data models within the API. This section refers to the Modules that have a 1:1 data model relationship.

There are 3 optional endpoint formats defined within all modules. Each of these endpoints supports a specific set of standard operations.

Module Root

```
/api/3/{plural_module_name}
```

The Module Root displays a paginated listing of all records within that module. The default sorting is by last modified date.

```
/api/3/{plural_module_name}?{$filter_query}
```

The Module Root may be filtered to a specific set of criteria as opposed to generally listing all models within the database.

Supported Operations

All Module Root endpoints support the following operations.

Operation	Description	Formats
GET	Allows you to read the list of records	-
POST	Insert a new record of the specified module type	JSON
PUT	Update an existing record, requires `"@id"`	JSON
DELETE	Delete an existing record, requires `"@id"`	-

Module Records

```
/api/3/{plural_module_name}/{record_uuid}
```

Supported Operations

All Module Record endpoints support the following operations.

Operation	Description	Formats
GET	Allows you to read the list of records	-

Playbook Endpoints

Custom Endpoints (Custom API Endpoint Trigger)

Endpoints Available

```
https://FortiSOAR.ROOT.URL/api/triggers/1/{name}
OR
https://FortiSOAR.ROOT.URL/api/triggers/1/deferred/{name}
```

These are user-defined endpoints generated based on the specified arbitrary name included in the Playbook trigger.

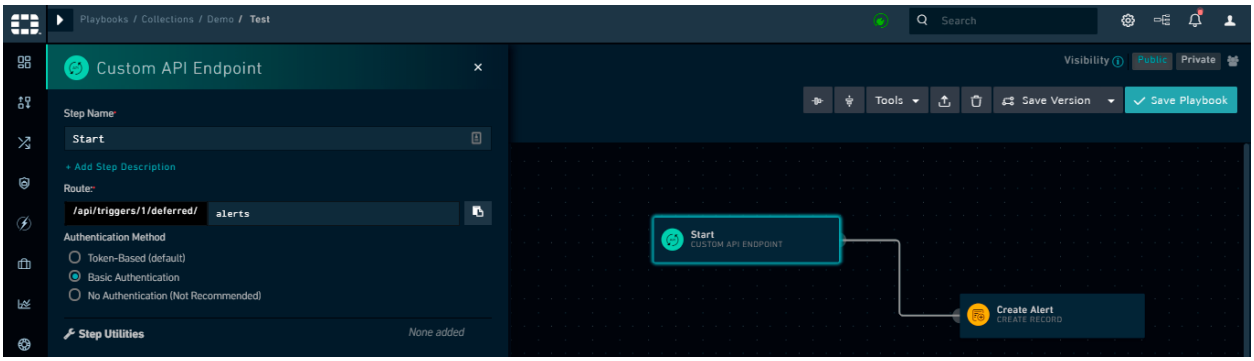

Figure 1. *Custom API Endpoint Trigger*

The `deferred` endpoints allow for alternate authentication means, i.e., no HMAC (token) requirement, for external systems that may only offer Basic or non-authenticated webhooks (No Authentication).

Configuration Endpoints

The Configuration endpoints primarily govern system and user interaction. Generally, it should not be necessary to use these endpoints outside of the UI, but they are provided here as a reference.

Model Metadatas

Warning: Modifications to the Model Metadatas and Attributes are not recommended due to the potential for database manipulation leading to a potentially difficult recovery.

`Model Metadata` describes every piece of customizable data models in our system. We have two pieces of metadata: `Model` and `Attribute`. `Models` are comprised of any number of `Attributes`. Attributes can be of any type, ranging from any JSON primitive type to relationships with existing entities. These relationships can be of type toMany or toOne.

ModelMetadata: `/api/3/apidoc/ModelMetadata`

- Type [`type`]
  Metadata type, JSON primitive for attributes or internal types for models. See `type` for `AttributeMetadata` for valid JSON primitives.
- Parent Type [`parentType`]
  The type of the parent, if set to `null`, will default to the `BaseEntity` class.
- Table Name [`tableName`]
  The table name for ORM, this will be the table that is created in the database.
- Ownable [`ownable`]
  Determines if this class should have the ownable fields (`owners`)
- Trackable [`trackable`]
  Determines if this module should have the trackable fields (`createDate`, `createUser`, `modifyDate`, `modifyUser`)
- Indexable [`indexable`]
  Determines if this class should have the indexable fields (`id`). This gives the object a global sequence id field.
- Attributes [`attributes`]
  Key-value list of the attributes for this class. These attributes can be described here: `/api/3/apidoc/AttributeMetadata`
- Display Template [`displayTemplate`]
  Template for the display name of record as an AngularJS template.
- Descriptions [`descriptions`]
  Map of description type => description, example:

```
{
    "singular" : "Alert",
    "plural" : "Alerts"
}
```

AttributeMetadata: `/api/3/apidoc/AttributeMetadata`

- Id [`id`]
  The unique object id for this attribute, will be auto generated if not specified.
- Type [`type`]
  Metadata type, JSON primitive for attributes or iri/internal types for models.
  Supported primitives are:
  `integer`,
  `string`,
  `id`,
  `boolean`,
  `number`,
  `null`
  `object`,
  `array`

Note: Adding an array field type is currently not supported through the FortiSOAR™ UI.
`object` and `array` are special types. See the *Extended Data* section below.
Note: `null` is supported in theory but might not make sense to use.

- Name [`name`] (**required**, **unique**) The name of the attribute.
  Note: DO NOT use underscores in the field name. Must be Camel-Case.
- Length [`length`]
  The length of the attribute. For anything less than 255, it will created in the SQL database as a VARCHAR with a defined length.
- FormType [`formType`]
  The type of form to use to render this attribute. Supported types are:
  `integer`,
  `text`,
  `textarea`,
  `checkbox`,
  `manyToMany`,
  `datetime`,
  `picklist`,
  `lookup`,
  `oneToMany`,
  `file`,
  `manyToOne`,
  `richtext`,
  `phone`,
  `email`,
  `json`
- Data Source [`dataSource`]
  The source of the data, not defined explicitly. An example of picklist lookups:

```
{
    "module" : "picklists",
    "query": {
        "logic": "AND",
        "filters": [
            {
              "field": "listName__name",
              "operator": "eq",
              "value": "Criticality"
            }
        ],
        "sort": [
            {
              "field": "orderIndex",
              "direction": "ASC"
            }
        ]
```

```
        }
    }
```

- An example of a relationship for all items in a collection, `query` is null because no query should be made to limit the lookup:

```
    {
        "module" : "alerts",
        "query" : null
    }
```

- Searchable [`searchable`]
  Determines if this attribute should be searchable within a grid in the FortiSOAR™ UI. This does not affect record indexing in ElasticSearch. The field will yet be displayed in the global search results in the UI even if it is not marked as searchable.
- Grid Column [`gridColumn`]
  Determines if this attribute should show up in a grid by default.
- Mapped By [`mappedBy`]
  Field name on the referenced type, if it is a collection and not the owner of a relationship.
- Inversed By [`inversedBy`] (**validation enforced**)
  Field name on the referenced type, if it is a collection and is the owner.
- Owns Relationship [`ownsRelationship`] (**validation enforced**)
  Whether this model owns the relationships, if is collection
- Validation [`validation`]
  Validation object, a list of validation attributes with boolean values or condition sets.
  Examples:
  `{ "required" : true }`
  `{ "required" : false }`
  Example of conditional validation: Only require `resolved` if `status` equals `Resolved`, only

```
    {
        "required": [
            {
                "logic": "AND",
                "result": true,
                "filters": [
                    {
                        "field": "status",
                        "operator": "eq",
                        "value": "/api/3/picklists/97e9bc03-5a4c-43c6-
b3a3-47422b42d288"
                    }
                ]
            }
        ]
    }
```

- Visibility [`visibility`
  Determines if this attribute should be visible during data entry.

This can be the following:
`true`, the default, will always be shown.
`false`, will not show up in the application.
Query object, see above, will be visible if the condition matches.

- Readable [`readable`]
  Determines if this attribute should be returned during READ operations.
- Writeable [`writeable`]
  Determines if this attribute should be modified during UPDATE operations.
- Identifier [`identifier`]
  Determines if this attribute is a unique identifier for this class.
- Unique [`unique`]
  Determines if the attribute is unique.
- Display Name [`displayName`]
  Display name as a template using field names, example:
  ```
  {
      "displayName" : "{{ sourceId }}"
  }
  ```
- Descriptions [`descriptions`]
  Map of description type => description, example
  ```
    {
        "singular" : "Source ID"
    }
  ```

Examples

For the following examples, we will use `alerts`.

Viewing Model Metadata

To view a model, use `GET` on `/api/3/model_metadatas/7ab10952-b2ea-4910-9762-eb685bdd29eb`. You should receive a result like this:


```
{
  "@context": "/api/3/contexts/ModelMetadata",
  "@id": "/api/3/model_metadatas/7ab10952-b2ea-4910-9762-eb685bdd
29eb",
  "@type": "ModelMetadata",
  "parentType": "/api/3/model_metadatas/base_entities",
  "tableName": "alerts",
  "ownable": true,
  "trackable": true,
  "indexable": true,
  "displayName": "{{ name }}",
  "descriptions": {
    "singular": "Alert",
```

```
    "plural": "Alerts"
  }
}
```

Model Attributes

All models will have attributes to describe the fields on the json. These can be retrieved by using:

`GET /api/3/model_metadatas/<uuid>?$relationships=true`


```
{
    "@id": "/api/3/model_metadatas/08854338-6440-4b66-a07f-9788086fa088",
    "@type": "ModelMetadata",
    "attributes": [
        {
            "@id": "/api/3/attrib_model_metadata/8582cab2-ed5a-4daa-abaa-0277cba104e9",
            "@type": "AttribModelMetadata",
            "sattrib": "/api/3/model_metadatas/08854338-6440-4b66-a07f-9788086fa088",
            "type": "string",
            "name": "firstname",
            "length": 255,
            "formType": "text",
            "dataSource": [],
            "searchable": true,
            "system": true,
            "encrypted": false,
            "gridColumn": true,
            "collection": false,
            "inversedField": null,
            "ownsRelationship": false,
            "validation": {
                "required": true,
                "minlength": 0,
                "maxlength": 255
            },
            "bulkAction": {
                "allow": false
            },
            "defaultValue": "",
            "dataSourceFilters": [],
            "tooltip": "",
            "visibility": true,
            "readable": true,
            "writeable": true,
            "identifier": false,
```

```
            "unique": false,
            "peerReplicable": true,
            "displayName": "{{ firstname }}",
            "descriptions": {
                "singular": "First Name"
            }
        }
    ],
    "type": "people",
    "parentType": "actors",
    "tableName": "actors",
    "ownable": false,
    "trackable": true,
    "peerReplicable": false,
    "defaultSort": [
        {
            "field": "firstname",
            "direction": "ASC"
        },
        {
            "field": "lastname",
            "direction": "ASC"
        }
    ],
    "indexable": true,
    "writable": true,
    "displayName": "{{ firstname }} {{ lastname }}",
    "descriptions": {
        "singular": "Person",
        "plural": "People"
    }
}
```


Exporting a Model

To export a model, use `GET` on `/api/3/model_metadatas/<uuid>?$relationships=true&$export=true`. You should receive a result like this:


```
{
  "@type": "ModelMetadata",
  "uuid": "7ab10952-b2ea-4910-9762-eb685bdd29eb",
  "type": "alerts",
  "tableName": "alerts",
  "attributes": [
    {
      "@type": "AttributeMetadata",
      "uuid": "16c855d3-29b4-4d29-90a8-8f322b556067",
```

```
      "name": "name",
      "length": 255,
      "searchable": true,
      "gridColumn": true,
      "validation": {
        "required": true
      },
      "displayName": "{{ name }}",
      "descriptions": {
        "singular": "Name"
      }
    },
    ...
  ]
}
```

Updating Model Metadata

To update a model, first, you should `GET` the model you would like to update. Then, use `PUT` with the fields you want to change:

`PUT /api/3/model_metadatas/7ab10952-b2ea-4910-9762-eb685bdd29eb`

```
{
    "ownable": true
}
```

To update attributes on a model, first, you should `GET` the model. Then use `PUT` on the `attributes` field with the modifications. You need to post all the attributes as a whole.

`PUT /api/3/model_metadatas/7ab10952-b2ea-4910-9762-eb685bdd29eb`

```
{
  "attributes": [
    {
      /* Existing with @id */
    },
    {
      /* New without @id */
    }
  ]
}
```

You can update an attribute. For example, going to `/api/3/attribute_metadatas/56d50d6a5a4ef3f3348b4574` will return a single metadata.

Creating New Models

To create a new model, use the same syntax as you have used for updating the models; except use `POST` and specify a `type` and `parentType`.

`POST /api/3/model_metadatas`

```
{
    "type": "malware",
    "parentType": "/api/3/model_metadatas/base_entities",
    "descriptions": {
        "singular": "Malware",
        "plural": "Malware"
    },
    "tableName": "malware"
}
```

The plural description will be used as the Module description. TableName is required for Postgres implementations of this model.

View Template

Create List View

`POST /api/3/system_view_templates`

```
{
    "id": "modules/malwares-list",
    "type": "rows",
    "config": {
        "rows": [{
            "columns": [{
                "widgets": [{
                    "type": "grid",
                    "config": []
                }]
            }]
        }]
    }
}
```

Create Detail View

`POST /api/3/system_view_templates`

```
{
    "id": "modules/malwares/detail",
    "type": "rows",
    "config": {
        "rows": [{
            "columns": [{
                "widgets": [{
                    "type": "form",
                    "config": []
                }]
            }]
        }, {
            "columns": [{
                "widgets": [{
                    "type": "relationship.subtab",
                    "config": []
                }]
            }]
        }]
    }
}
```

Extended Data

The attribute types that are type: `object` or `array` are often referred to as: 'Extended Data'. These are schema-less json blobs that have minimal input validation.

Attribute metadata for these types will look like this:

```
[
    {
        "@type": "AttributeMetadata",
        "uuid": "60e00773-2de6-44aa-8af0-174e4fbdec17",
        "name": "extendedData",
        "type": "object",
        "formType": "object",
        "displayName": "{{ extendedData }}",
        "descriptions": {
            "singular": "Extended Data"
        }
    },
    {
        "@type": "AttributeMetadata",
        "uuid": "338040a1-6965-4e56-b71d-80cc83ecbeb7",
        "name": "extendedDatas",
        "type": "array",
        "formType": "array",
        "displayName": "{{ extendedDatas }}",
```

```
        "descriptions": {
            "singular": "Extended Data (Array)"
        }
    }
]
```

Examples of Valid Data

```
{
    "extendedData": {
        "test": 1234,
        "key2": [
            "randomdata"
        ]
    }
}
{
    "extendedDatas": [
        1234,
        {
            "nested": "object"
        }
    ]
}
```

Examples of Bad Data

```
{
    "extendedData": [
        "array",
        "on object",
        "is bad"
    ]
}
{
    "extendedDatas": {
        "do not": "post object to array"
    }
}
```

Indexing of Extended Data

Because of the limitations on ElasticSearch, extended data needs to be transformed into a string before it is indexed. Searches performed on this data will act like a traditional text search.

System Update

After making changes to the metadata, it is time to publish. If you have the UPDATE permission to the Application module, make a PUT command to `/api/publish`. You will get a response like this:

```
{
    "@type": "Publish",
    "status": "started"
}
```

This will indicate that a system update has begun. This will make every request to the API return a 503 until the publish is complete. The progress will look something like this:

```
{
  "@type": "SystemUpdate",
  "code": 503,
  "message": "Clearing cache",
  "progressPercent": 22,
  "startTime": 1460678007
}
```

Modules List

- Route: modules/{{ module }}
- Template: modules/{{ module }}/list
- State: main.modules.list
- Parameters:
    - module: name of module

Model-dependent page that will be able to utilize the state parameters to build widgets.

Simple Example

```
{
  "id": "modules/incidents/list",
  "type": "rows",
  "config": {
    "rows": [{
      "columns": [{
        "widgets": [{
          "type": "grid"
        }]
      }]
    }]
```

```
  }
}
```

Modules Detail

- Route: modules/{{ module }}/{{ id }}
- Template: modules/{{ module }}/detail
- State: main.modulesDetail
- Parameters:
    - module: name of the module
    - id: id of the given record

Model-dependent page that will be able to utilize the state parameters to build widgets.

Simple Example

```
{
  "id": "modules/incidents/detail",
  "type": "rows",
  "config": {
    "rows": [{
      "columns": [{
        "widgets": [{
          "type": "form"
        }]
      }]
    }, {
      "columns": [{
        "widgets": [{
          "type": "relationship.subtab",
          "config": []
        }]
      }]
    }]
  }
}
```

Modules Form

- Route: modules/{{ module }}/add or modules/{{ module }}/edit/{{ id }}
- Template: modules/{{ module }}/form
- State: main.modulesAdd or main.modulesEdit
- Parameters:
    - module: name of the module

- id: id of the given record (on edit only)

Module edit and module add both use the form template to load in a template that both modes can share to display fields and a form to the user. Model-dependent page that will be able to utilize the state parameters to build widgets.

Example


```
{
  "id": "modules/incidents/form",
  "type": "form",
  "config": {
    "rows": [{
      "columns": [{
        "widgets": [{
          "config": {
            "rows": [{
              "columns": [{
                "fields": [
                  "name"
                ]
              }]
            }],
            "size": "large"
          },
          "type": "formGroup"
        }, {
          "config": {
            "rows": [{
              "columns": [{
                "fields": [
                  "description"
                ]
              }]
            }]
          },
          "type": "formGroup"
        }, {
          "config": {
            "rows": [{
              "columns": [{
                "fields": [
                  "phase",
                  "status"
                ]
              }]
            }],
            "title": "Summary"
```

```
        },
        "type": "formGroup"
      }, {
        "config": {
          "rows": [{
            "columns": [{
              "fields": [
                "resolution"
              ]
            }]
          }],
          "title": "Resolved"
        },
        "type": "formGroup"
      }, {
        "config": {
          "rows": [{
            "columns": [{
              "fields": [
                "confirmation",
                "confidence",
                "severity",
                "category",
                "source",
                "originPoint",
                "incidentLead"
              ]
            }]
          }],
          "title": "Details"
        },
        "type": "formGroup"
      }, {
        "config": {
          "rows": [{
            "columns": [{
              "fields": [
                "dateOfIncident",
                "discoveredOn",
                "dwellTime",
                "containmentTime",
                "recoveryTime"
              ]
            }]
          }],
          "title": "Dates"
        },
        "type": "formGroup"
      }]
```


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

Application

Template: app

The application template is a unique widget that is loaded when you login to exo. This template describes header and navigation details.

Simple Example

```
{
  "id": "app",
  "type": "app",
  "config": {
    "header": {
    },
    "navigation": [{
      "title": "Home",
      "icon": "fa fa-home",
      "require": [],
      "state": {
        "name": "main.dashboard",
        "parameters": []
      }
    }, {
      "title": "Vulnerability Management",
      "icon": "fa fa-bug",
      "items": [{
        "title": "Vulnerabilities",
        "require": {
          "module": "vulnerabilities",
          "action": "canRead"
        },
        "state": {
          "name": "main.modules.list",
          "parameters": {
            "module": "vulnerabilities"
          }
        }
      }, {
        "title": "Assets",
        "require": {
          "module": "assets",
```

```
          "action": "canRead"
        },
        "state": {
          "name": "main.modules.list",
          "parameters": {
            "module": "assets"
          }
        }
      }]
    }]
  }
}
```

Header

Header property is the collection of the header component details.

Navigation

Navigation property is an array of links to build the left-hand navigation bar.

Links

A single link can be created like below.

```
{
  "title": "Home", // Title of the page
  "icon": "fa fa-home", // Icon used for the link
  "require": {  // Restriction of the link. or collection of link
s
    "module": "vulnerabilities", // What module to restrict on
    "action": "canRead" // Restrited action (defaults to canRead)
.
  },
  "state": { // State and it's parameters
    "name": "main.dashboard",
    "parameters": []
  }
}
```

A collection of links will look like the following:

```
{
  "title": "Vulnerability Management", // Title of collection
  "icon": "fa fa-bug", // Icon used for the collection
  "items": [{ // array of links (same as single link)
    "title": "Vulnerabilities",
```

```
    "require": {
      "module": "vulnerabilities",
      "action": "canRead"
    },
    "state": {
      "name": "main.modules.list",
      "parameters": {
        "module": "vulnerabilities"
      }
    }
  }, {
    "title": "Assets",
    "require": {
      "module": "assets",
      "action": "canRead"
    },
    "state": {
      "name": "main.modules.list",
      "parameters": {
        "module": "assets"
      }
    }
  }]
}
```

HMAC Authentication

How FortiSOAR™ Authorizes Payloads with HMAC

To successfully authenticate our payloads, we are required to get a fingerprint as part of the request and compare it locally by recreating the fingerprint. This is subjective and can be defined by the HMAC provider (FortiSOAR™). FortiSOAR™ fingerprinting will be based on hashing the private key against the identifier (NOT THE ACTUAL PAYLOAD).

How to Create an Identifier

The FortiSOAR™ identifier is created in such a way that some pieces will be gathered from the request and others will have to be provided by the requester. Our current identifier simply is: `ALGO.VERB.TIMESTAMP.FULL_URI.HASHED_PAYLOAD`

The identifier units are separated with periods. Here is a brief explanation of each unit:

- ALGO: The Algorithm the hashing is being done with, for a full list of hashing algorithms available in PHP you can use the hash_algos function.
- VERB: The current request action, GET, POST, PUT, PATCH, DELETE
- TIMESTAMP: The timestamp the fingerprint is created. This is also sent as part of the request.
  Note: Timestamp must be in UTC format.
- FULL_URI: The full URI of the request, including any parameters
- HASHED_PAYLOAD: The string that is to be fingerprinted. Hashed with the requested ALGO

Once the above is created the fingerprint is created with `hash_hmac(algo, identifier, private_key, false)`. The fingerprint is expected to be represented as a string as opposed to raw binary.

PHP Example

```
$privateKey = "our_private_key";
$publicKey = "our_public_key"; // used in the example following t
his

  $testAlgo = "sha256";
     $testVerb = "GET";
     $testTimestampObj = new \DateTime();
     $testTimestamp = $testTimestampObj->format('Y-m-d H:i:s');
     $testFullUri = "http://example.com/api/1/data/1";
```

```
    $testPayload = "This is a test string"; // this could be

our_public_key on GET requests
    $hashedPayload = hash($testAlgo,$testPayload,false);

    $rawFingerprint =
         $testAlgo.'.'.
         $testVerb.'.'.
         $testTimestamp.'.'.
         $testFullUri.'.'.
         $hashedPayload;

    $hashedFingerprint =
         hash_hmac(
             $testAlgo,
             $rawFingerprint,
             $privateKey,
             false
         );
```

Python Example

```
import base64
import hashlib
import hmac
from datetime import datetime

DEFAULT_ALGORITHM = "sha256"
with open("/opt/cyops-integrations/integrations/configs/APPLIANCE
_PUBLIC_KEY", 'r') as public_key_file:
   public_key = public_key_file.read()
with open("/opt/cyops-integrations/integrations/configs/APPLIANCE
_PRIVATE_KEY", 'r') as private_key_file:
   private_key = private_key_file.read()

def generate_hmac(method, full_uri, payload, private_key, public_
key):
    if method == 'GET':
        payload = public_key
    timestamp = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")
    payload = payload if type(payload) is bytes else payload.enco
de()
    digest_method = hashlib.new(DEFAULT_ALGORITHM)
    digest_method.update(payload)
    hashed_payload = digest_method.hexdigest()
    raw_fingerprint = "{0}.{1}.{2}.{3}.{4}".format(DEFAULT_ALGORI
THM,
                                                   method,
```

```
                                                        timestamp,
                                                        full_uri,
                                                        hashed_payload
)

    hashed = hmac.new(private_key.encode(), raw_fingerprint.encod
e(),
                      hashlib.sha256)
    hashed_fingerprint = hashed.hexdigest()
    header = base64.b64encode(
        '{0};{1};{2};{3}'.format(DEFAULT_ALGORITHM, timestamp, pu
blic_key,
                                 hashed_fingerprint).encode())
    return 'CS {}'.format(header.decode())
```

Using the Signature

After creating our identifier, we can send the payload to our API. However, the fingerprint alone is not enough. We'll also need to send our public key, the timestamp used to hash our fingerprint, and the algorithm being used in the request. These pieces should be concatenated using semicolons and included in the authorization header base64 encoded.

PHP Example

```
$header =
base64_encode($testAlgo.';'.$testTimestamp.';'.$publicKey.';'.$ha
shedFinge
rprint);

curl_setopt($ch, CURLOPT_HTTPHEADER, array(
 "Authorization: CS " . $header,
 "X-CS-Data: " . $testPayload
));
```

Python Example

```
import requests
import json
auth_header = generate_hmac(method,full_url, json.dumps(payload),
 private_key, public_key)
headers =
{
  Authorization': auth_header
}
try:
```

```
  req = requests.request(method, full_url, data=payload, headers=
headers, verify=False)
  print (req)
except Exception as e:
  print(e)
```

If the fingerprint is authorized the request will be delivered to the API; otherwise, a 401 error will be returned.

API Methods

CRUD API

Authentication API

Authenticate

This API is used to generate a JWT authentication token for FortiSOAR™ authentication using username and password. It returns a JSON response containing the JWT token which you can then use to call other FortiSOAR™ APIs by passing it into the Authorization header. For example, `Authorization: "Bearer {Token}"`.

Request

```
METHOD: POST
URL: https://{HOSTNAME}/auth/authenticate
BODY:
{
  "credentials":
  {
      "loginid": "{{username}}",
      "password":"{{password}}"
  }
}
```

Response

```
{
    token:  “VALID JWT TOKEN"
}
```

You can deploy the FortiSOAR™ enterprise license by running the `csadm` command as a *root* user: `csadm license --deploy-enterprise-license <License File Path>` . For more information on the FortiSOAR™ Admin CLI (`csadm`), see the *FortiSOAR™ Admin CLI* chapter in the “Administration Guide.” For more information on licensing, see the *Licensing FortiSOAR™* chapter in the “Deployment Guide.”

Deploying the FortiSOAR™ license using the API

Before you can deploy the license file using the API, you require to generate the authorization token. Use the following `curl` call to get the token:

```
curl -X POST \
  https://<fortisoar_host>/auth/authenticate \
  -H 'Content-Type: application/json' \
  -d '{
    "credentials":
    {
        "loginid":"<loginid>",
        "password":"<password>"
    }
}
```

To deploy the license file using the API, make the following REST call:

```
curl -X POST
  'https://<hostname>/api/auth/license
  -H 'authorization: AUTH_TOKEN'
  -H 'content-type: application/json'
  -d '{
        "license_key" : "LICENSE_KEY"
    }'
```

The `LICENSE_KEY` value is the contents of the license file generated during the license generation step. As this API is an authenticated API, it can be used only when a valid license is already deployed. A successful deployment returns an `HTTP 200 OK` status.

Retrieving the license details

Once you have successfully deployed a FortiSOAR™ license, you can retrieve the details of your license at any time using the following API:

```
curl -X GET
  'https://<fortisoar_host>/auth/license/?param=license_details'

  -H 'authorization: AUTH_TOKEN'
```

The API displays a message similar to the following:

```
{
  "expired": false,
  "expired_nodes": [],
  "remaining_days": 180,
  "details": {
    "is_distributed": false,
    "role": [
      "master"
    ],
    "entitlements": {}
  },
```

```
  "nodes": {
    "6d1a908cc06887910ed3237e0c233fb1": {
      "message": "License Details: Max Users: 10, Expiring on: 20
19-11-03, Issued for 180 days",
      "details": {
        "total_users": 10,
        "remaining_users": 9,
        "expiring_on": "2019-11-03",
        "issue_time": "2019-05-07 06:26:13.368238",
        "total_days": 180,
        "remaining_days": 180,
        "customer_name": "abc",
        "is_distributed": false,
        "entitlements": {},
        "role": [
          "master"
        ]
      },
      "node": {
        "nodeId": "6d1a908cc06887910ed3237e0c233fb1",
        "nodeName": "fortisoar.localhost",
        "status": "active",
        "role": "primary",
        "currentState": "primary server"
      }
    }
  }
}
```


Model Operations

Create Individual Model Records

Request

```
METHOD: POST
URL: /api/{version}/{collection}
BODY:
{
  "field1": "value1",
  "field2": "value2"
}
```

Example

Request

```
METHOD: POST
URL: /api/3/assets
BODY:
{
  "ip": "8.8.8.8",
  "hostname": "google-public-dns-a.google.com"
}
```

Response

```
STATUS: 201
BODY:
{
  "@id": "/api/3/assets/01199609-d60f-356b-a762-129a6e1b353b",
  "@type": "Asset",
  "ip": "8.8.8.8",
  "hostname": "google-public-dns-a.google.com"
}
```

Retrieve Collection of Model Records

The responses adhere to the Hydra Spec for pagination. Records that match the specified query (see Filtering for more info) are returned as objects within the "hydra:member" array.

Request

```
METHOD: GET
URL: /api/{version}/{collection}
```

Response

```
STATUS: 200
BODY:
{
  "@context": "/api/{version}/contexts/{collection type}",
  "@id": "/api/{version}/{collection}",
  "@type": "hydra:PagedCollection",
  /* Contains some Hydra pagination properties */
  "hydra:member": [
    { "@id": "/api/{version}/{collection}/{uuid}", ... },
    { "@id": "/api/{version}/{collection}/{uuid}", ... },
    { "@id": "/api/{version}/{collection}/{uuid}", ... }
  ]
}
```

Example

Request

```
METHOD: GET
URL: /api/3/assets
```

Response

```
STATUS: 200
BODY:
{
  "@context": "/api/3/contexts/Asset",
  "@id": "/api/3/assets",
  "@type": "hydra:PagedCollection",
  "hydra:totalItems": 2,
  "hydra:itemsPerPage": 30,
  "hydra:firstPage": "/api/3/assets",
  "hydra:lastPage": "/api/3/assets",
  "hydra:member": [
    {
      "@id": "/api/3/assets/01199609-d60f-356b-a762-129a6e1b353b"
,
      "@type": "Asset",
      "ip": "8.8.8.8",
      "hostname": "google-public-dns-a.google.com"
    },
    {
      "@id": "/api/3/assets/017d3845-e204-367c-b74d-6bb37a8239b7"
,
      "@type": "Asset",
      "ip": "8.8.4.4",
      "hostname": "google-public-dns-b.google.com"
    }
  ]
}
```

Retrieve Single Model Records

Request

```
METHOD: GET
URL: /api/{version}/{collection}/{uuid}
```

Response

```
STATUS: 200
BODY:
{
  "@id": "/api/{version}/{collection}/{uuid}",
  "@type": "{collection type}",
  "field1": "value1",
  "field2": "value2"
}
```

Example

Request

```
METHOD: GET
URL: /api/3/assets/01199609-d60f-356b-a762-129a6e1b353b
```

Response

```
STATUS: 200
BODY:
{
  "@id": "/api/3/assets/01199609-d60f-356b-a762-129a6e1b353b",
  "@type": "Asset",
  "ip": "8.8.8.8",
  "hostname": "google-public-dns-a.google.com"
}
```

Update Single Model Records

Request

```
METHOD: PUT
URL: /api/{version}/{collection}/{uuid}
BODY:
{
  "field1": "value1",
  "field2": "value2"
}
```

Example

Request

```
METHOD: GET
URL: /api/3/assets/01199609-d60f-356b-a762-129a6e1b353b
```

```
BODY:
{
  "ip": "8.8.8.8"
}
```

Response

```
STATUS: 200
BODY:
{
  "@id": "/api/3/assets/01199609-d60f-356b-a762-129a6e1b353b",
  "@type": "Asset",
  "ip": "8.8.8.8",
  "hostname": "google-public-dns-a.google.com"
}
```

Delete Single Model Records

Danger: Deleting models can result in system instability and data model. We advise that you do not delete models after creation.

Request

```
METHOD: DELETE
URL: /api/{version}/{collection}/{uuid}
```

Example

Request

```
METHOD: DELETE
URL: /api/3/assets/01199609-d60f-356b-a762-129a6e1b353b
```

Response

```
STATUS: 204
REASON: No Content
```

Insert records in bulk

You can use the `Bulk Insert API` to insert records in bulk, in a module that you specify.

Note: It is recommended that you insert up to 100 records in single request and loop over your dataset.

Request

```
METHOD: POST
URL: {{YourFortiSOARHostname}}/api/{version}/insert/{moduleType}
BODY: {
        "data”: [{record 1}, {record 2}]
      }
}
```

Example

Request

```
METHOD: POST
URL: {{YourFortiSOARHostname}}/api/3/insert/alerts
BODY: {
        "data": [
        {
            "name": "bulk record 1",
            "description": "<p>bulk record 1 description</p>",
            "severity": "/api/3/picklists/58d0753f-f7e4-403b-953c
-b0f521eab759"
        },{
            "name": "bulk record 2",
            "description": "<p>bulk record 2 description</p>",
            "severity": "/api/3/picklists/58d0753f-f7e4-403b-953c
-b0f521eab759"
        },{
            "name": "bulk record 3",
            "description": "<p>bulk record 3 description</p>",
            "severity": "/api/3/picklists/58d0753f-f7e4-403b-953c
-b0f521eab759"
        }
    ]
}
```

Response: If all the records are inserted successfully, then you will get a `200 OK` response. If not all the records can be inserted, then you will get a `207` (Partially Deleted) response.

Example of a complete response:

```
{
   “@context”: “/api/3/contexts/Alert”,
   “@id”: null,
   “@type”: “hydra:Collection”,
   “hydra:member”: [
       {
           inserted record data
```

```
        }
    ]
}
```

Delete records in bulk

You can use the `Bulk Delete API` to delete records in bulk, in a module that you specify.

Request

```
METHOD: DELETE
URL: {{YourFortiSOARHostname}}/api/{version}/delete/{moduleType}
BODY: {
        "ids": ["value1", "value2", "value3"]
      }
}
```

Example

Request

```
METHOD: DELETE
URL: {{YourFortiSOARHostname}}/api/3/delete/alerts
BODY: {
        "ids": ["13d7115a-2d4d-4bfb-bdb8-efabfce547f011",
                "13d7115a-2d4d-4bfb-bdb8-efabfce547f012",
                "13d7115a-2d4d-4bfb-bdb8-efabfce547f013"]
      }
}
```

Response: If all the records are deleted successfully, then you will get a `200 OK` response. If not all the records can be deleted, if for example, you have provided an ID of an already deleted record, then you will get a `207` (Partially Deleted) response.

Update records in bulk

You can use the `Bulk Update API` to update records in bulk, in a module that you specify.

Request

```
METHOD: PUT
URL: {{YourFortiSOARHostname}}/api/{version}/update/{moduleType}
BODY: {
    “data” : [
        {
            "field1": "value1",
```

```
            "field2": "value2",
            "field3": "value3"
        },
        {
            "field1": "value1",
            "field2": "value2",
            "field3": "value3"
        }
    ]
}
```

Example

Request

```
METHOD: PUT
URL: {{YourFortiSOARHostname}}/api/3/update/alerts
BODY: {
    “data” : [
        {
            “@id”: “/api/3/alerts/a3abeb23-f323-45a9-856b-2bf9d10
386cb”,
            “source”: “1238765”,
            "description": "Test 1"
        },
        {
            “@id”: “/api/3/alerts/8c086c21-2c79-4488-ae22-d0e1164
69fe2”,
            “source”: “097353”,
            “description”: “Test 2”,
        }
    ]
}
```

Response: The response will contain all the fields specified in the metadata of the module. Following is a snippet extract of a successful response:

```
STATUS: 200
BODY: {
    “data” : [
        {
            “@id”: “/api/3/alerts/a3abeb23-f323-45a9-856b-2bf9d10
386cb”,
            “source”: “1238765”,
            "description": "Test 1"
        },
        {
            “@id”: “/api/3/alerts/8c086c21-2c79-4488-ae22-d0e1164
```

```
69fe2”,
            “source”: “097353”,
            “description”: “Test 2”,
        }
    ]
}
```

Notes:

- If you provide the body of the request appropriately, then all the records get successfully updated to the specified module, and you get a `200 OK` response.
- If you do not provide an `id` for an object (record), then the module is partially updated with the records that have an associated id. Records that do not have an id will not be updated in the specified module. In this case, you get a `207` response, which will list the records that are not updated.
- If you specify a `null` value for any field, those records get successfully updated.

Upsert records

Important: That upsert works only if you have defined uniqueness for records in the module you want to upsert records. If you have not defined record uniqueness, the upsert operation will always insert records. Also, note that when you are upserting records, fields that are marked as non-editable also get updated and the upsert behavior for uniqueness will not work for fields that are marked as encrypted.

Single Upsert API

You can use the `Upsert API` to either update an existing record, if any record matches the unique list of fields you have specified for that module using the Module Editor, or insert a new record.

Request

Following is a sample request for a single upsert request:

```
METHOD: POST
URI: {{YourFortiSOARHostname}}/api/3/upsert/{moduleType}
BODY:
      {
        "name": "value1",
        "sourceId": null,
        "severity": "value2"
      }
```

Bulk Upsert API

You can use the `Bulk Upsert API` to either bulk update multiple records, if the records match the unique list of fields you have specified for that module, or bulk insert new records.

Request

Following is a sample request for a bulk upsert request:

```
METHOD: POST
URI: {{YourFortiSOARHostname}}/api/3/bulkupsert/{moduleType}
BODY:[
       {
        "name": "value1",
        "sourceId": "null",
        "severity": "value2"
       },
      {
        "name": "value1",
        "sourceId": "null",
        "severity": "value2"
       }
]
```

Response: If all the records are upserted successfully, then you will get a `200 OK` response. If some records are upserted successfully and some records are not upserted, then you will get a `207` (Partial Upsert) response. If none of the records are upserted successfully, then you will get a `400` response.

Relationship Operations

For these examples, we will expect that an Incident has already been created in the system. All these operations will affect the `assets` field of an Incident.

Get Relationship

Request

```
METHOD: GET
URL: /api/3/incidents/bbdc13f8-015f-4615-8c63-4cebb6ec991b/assets
```

Response

```
{
  "@context": "/api/3/contexts/Asset",
```

```
  "@id": "/api/3/incidents/bbdc13f8-015f-4615-8c63-4cebb6ec991b/a
ssets",
  "@type": "hydra:PagedCollection",
  "hydra:totalItems": 1,
  "hydra:itemsPerPage": 30,
  "hydra:firstPage": "/api/3/incidents/bbdc13f8-015f-4615-8c63-4c
ebb6ec991b/assets",
  "hydra:lastPage": "/api/3/incidents/bbdc13f8-015f-4615-8c63-4ce
bb6ec991b/assets",
  "hydra:member": [
    {
      "@id": "/api/3/assets/2ac5eafd-e7dd-465e-8ddd-6b9aeb7125c0"
,
      "@type": "Asset",
      "macAddress": "as:se:t2",
      "id": 639
    }
  ]
}
```

Append Existing Object to Relationship

Request

```
METHOD: POST
URL: /api/3/incidents/bbdc13f8-015f-4615-8c63-4cebb6ec991b/assets
BODY:
{
  "@id": "/api/3/assets/e0a3521f-67de-473f-a9a0-7f2d44626567"
}
```

Response

```
{
  "@id": "/api/3/assets/e0a3521f-67de-473f-a9a0-7f2d44626567",
  "@type": "Asset",
  "macAddress": "as:se:t1",
  "id": 638
}
```

Append New Object to Relationship

Request

```
METHOD: POST
URL: /api/3/incidents/bbdc13f8-015f-4615-8c63-4cebb6ec991b/assets
```

```
BODY:
{
  "macAddress": "as:se:t3",
  "assetType": "laptop"
}
```

Response

```
{
  "@id": "/api/3/assets/5434a6e1-e9cd-4ecf-ba6b-f8ac82c3effc",
  "@type": "Asset",
  "macAddress": "as:se:t3",
  "id": 639
  "assetType": "laptop"
}
```

Delete Relationship

Request

```
METHOD: DELETE
URL: /api/3/incidents/bbdc13f8-015f-4615-8c63-4cebb6ec991b/assets
/
e0a3521f-67de-473f-a9a0-7f2d44626567
```

Response

```
STATUS: 204
REASON: No Content
```

Audit Log Purge Operations

You can use the Audit Log Purge API to purge audit logs on an automated as well as on-demand basis.

Stop Automatic Purging of Audit Logs

To stop automatic purging of audit logs, run the following API:

Request

```
Method: DELETE
URL:  /gateway/audit/activities/ttl
Content-Type: application/json
Authorization: <Bearer Token>
```

Example:

```
DELETE /gateway/audit/activities/ttl
Content-Type: application/json
Authorization: <Bearer Token>
```

Response: If the API is successfully run you will get a `200 OK` response, which stops the automatic purging of audit logs.

Delete audit logs before a particular date

To delete audit logs before a particular date, run the following API:

Request

```
Method: DELETE
URL:  /gateway/audit/activities
Authorization: <Bearer Token>
Cache-Control: no-cache
```

Example:

```
DELETE /gateway/audit/activities?uptoDate=1524477805
Authorization: <Bearer Token>
Cache-Control: no-cache
```

Response: If the API is successfully run you will get a `200 OK` response and then audit logs that exist before the specified date will be purged.

Query API Reference

API Routes

Apply Basic Query

```
GET /api/3/{collection}?param1=value&param2=value
```

For more information, see *API Endpoints Reference* chapter.

Apply Ad Hoc Query

```
POST /api/query/{collection}
BODY:
{ QUERY OBJECT }
```

- The `collection` parameter matches the collection route in the typical CRUD API (/api/3/{collection})

Example

```
POST /api/query/incidents
BODY:
{
  "logic": "AND",
  "filters": [
    {
      "field": "status.itemValue",
      "operator": "eq",
      "value": "Open"
    }
  ]
}
```

Apply Persisted Query

```
GET /api/query/{collection}/{queryId}
```

- The `collection` parameter matches the collection route in the typical CRUD API (`/api/3/{collection}`)
- The `queryId` parameter matches a Query Object UUID stored in `/api/3/query_objects`

Example

```
GET /api/query/incidents/2e77a714-f0c1-45ca-bd49-b71efbd9328c
```

Query Objects

Filter

A filter contains a field, operator, and a value.

```
{
"field": "{fieldName}",
"operator": "{operator}",
"value": {value}
}
```

Field

The field can be any valid field in the module. To access sub-elements of a relationship or picklist field, you can use dot notation or double-underscore notation.

```
"field": "name"
"field": "status.itemValue"
"field": "assignedToPerson.email"
```

Operator

The operator can be any of the available operators for a specific field type.

```
"operator": "eq"
"operator": "like"
```

For the complete list of supported operations, see the Supported Field Operators section.

Value

The value is the object of the filter or comparison. For example, if you are using the `is like` operator, then the value will be a pattern to match the record field.

```
"operator": "eq",
"value": "Alert 123: Repeated login failures - device xxx"
```

```
"operator": "like",
"value": "alert ___: Repeated login failures%"
```

For the complete list of supported operations, see the Supported Field Operators section.

Supported Field Operators

Operator	Value Syntax	Description
`eq`	string	**Equals**: Field is an exact match.
`neq`	string	**Not Equals**: Field is not an exact match.
`lt`	number	**Less Than**: Field has a lesser than value.
`lte`	number	**Less Than or Equal To**: Field has a lesser than or equal to value.
`gt`	number	**Greater Than**: Field has a greater than value.
`gte`	number	**Greater Than or Equal To**: Field has a greater than or equal to value.
`in`	value1|value2|...	**In**: Field is in the given list of values.
`nin`	value1|value2|...	**Not In**: Field is not in the given list of values.
`contains`	string	**Contains**: Object contains the key. Only applicable for field type *Object*.
`like`	string pattern	**Like**: Field matches the given pattern. The pattern can contain text, percent (`%`), or an underscrore (`_`) `%` represents zero or more of any characters.`_` represents one character.
`notlike`	string pattern	**Not Like**: Field does not match the given pattern. The pattern can contain text, percent (`%`), or an underscrore (`_`) `%` represents zero or more of any characters.`_` represents one character.
`isnull`	boolean	**Is Null**: If value is set as true, then this checks if the field is null. If value is set as false, then this checks if the field is not null.

Logic

AND

A collection of filters can be applied in conjunction using `"logic": "AND"`

Example

Query all records where `record.assignedToUser !== null && record.status === "Open"`

```
{
  "logic": "AND",
  "filters": [
    {
      "field": "assignedToUser",
      "operator": "isnull",
      "value": false
    },
    {
      "field": "status",
      "operator": "eq",
      "value": "Open"
    }
  ]
}
```

OR

A collection of filters can be applied in disjunction using `"logic": "OR"`

Example

Query all records where `record.status === "Open" || record.status === "Pending"`

```
{
  "logic": "OR",
  "filters": [
    {
      "field": "status",
      "operator": "eq",
      "value": "Open"
    },
    {
      "field": "status",
      "operator": "eq",
      "value": "Pending"
    }
  ]
}
```

Nested Logic

Nesting logic filters allows you to build queries with nested logic like `conditionX && (conditionY || conditionZ)`

Example

Query all records where `record.assignedToUser !== null && (record.status === "Open" || record.status === "Pending")`

```
{
  "logic": "AND",
  "filters": [
    {
      "field": "assignedToUser",
      "operator": "isnull",
      "value": false
    },
    {
      "logic": "OR",
      "filters": [
        {
          "field": "status",
          "operator": "eq",
          "value": "Open"
        },
        {
          "field": "status",
          "operator": "eq",
          "value": "Pending"
        }
      ]
    }
  ]
}
```

Sort

Records can be sorted by field/direction using the `"sort": []` list. This list contains objects with `"field"` and `"direction"` keys. The `"field"` key identifies which field (or association) to sort by. The `"direction"` key identifies whether to sort ascending (`"ASC"`) or descending (`"DESC"`). Order is maintained when applying these sorts.

```
{
  "logic": "AND",
  "filters": [],
  "sort": [
    {
      "field": "createDate",
      "direction": "DESC"
    }
  ]
}
```

Aggregation

Records can be aggregated using the `"aggregates": []` list. This list contains objects with `"operator"`, `"field"`, and `"alias"` keys. The `"operator"` key identifies which aggregate operator to apply to the specified field. The `"field"` key identifies the field to apply it to. The `"alias"` key identifies what the returned field looks like.

Supported Aggregate Operators

- `fields`
- `select`
- `count`
- `countdistinct`
- `groupby`
- `distinct`
- `sum`
- `max`
- `min`
- `median`
- `avg`

Example

Count all records in each status grouping.

```
{
  "logic": "AND",
  "filters": [],
  "aggregates": [
    {
      "operator": "groupby",
      "field": "status",
      "alias": "status"
    },
    {
      "operator": "countdistinct",
      "field": "*",
      "alias": "total"
    }
  ]
}
```

Associations

Wherever a `"field"` key is specified in a query object, you can use dot-notation to query against an associated entity using the same operators that are available for the root record.

Example

Query all records assigned to a user named Jeff.

```
{
  "logic": "AND",
  "filters": [
    {
      "field": "assignedToUser.firstname",
      "operator": "eq",
      "value": "Jeff"
    }
  ]
}
```

Model Type

Use this API to search for records from different models like Incident, Alerts or any other custom models based on multiple valid fields in the model.

The API is: `https://{{hostname}}/api/query/{{modelType}}`. For example, `https://{{hostname}}/api/query/incidents` where incidents is a model in which you want to search for records. This API uses POST as the request method.

Example

```
BODY:
{
    "aggregates": [
        {
            "operator": "groupby",
            "field": "phase.itemValue",
            "alias": "phase"
        },
        {
            "operator": "avg",
            "field": "dwellTime",
            "alias": "maxDwellTime"
        }
    ],
    "logic": "AND",
    "filters": []
}
```

Response: JSON list of records matching the filter criteria provided in the `BODY` of the `modelType`.